Алгоритмы программирования

Кафедра Медицинской кибернетики и информатики МБФ

Кафедра медицинской кибернетики и информатики РНИМУ им. Н.И. Пирогова Минздрава РФ

Отделение медицинской кибернетики МБФ и кафедра медицинской кибернетики и информатики основаны в 1973 году заслуженным деятелем науки РФ, профессором С.А. Гаспаряном.

Заведующий кафедрой - д.м.н. профессор Зарубина Т.

Завуч кафедры и зав. курсом Медицинская информатика – к.м.н. доцент Николаиди Е.Н.

Инструктаж по технике безопасности и противопожарной безопасности

- **требования,** обязательные для исполнения студентами в помещениях кафедры:

1. В помещениях кафедры запрещается курение и использование неисправных электроприборов.
2. К учебе на кафедре допускаются студенты, прошедшие инструктаж по технике безопасности и противопожарной безопасности, и лично расписавшиеся в журнале регистрации инструктажа студентов.
3. Перед началом занятий студенты обязаны ознакомиться со схемой эвакуации из учебных классов.
4. Учебная работа в компьютерных классах без преподавателя запрещена.
5. Включение и выключение оборудования может производиться только после разрешения преподавателя.

Действия при пожаре:

- Студент, обнаруживший пожар или его признаки (задымление, запах гари или тления различных материалов, повышение температуры и т.п.) обязан оповестить о пожаре сотрудников кафедры и затем покинуть здание.
- При срабатывании системы оповещения при пожаре студент обязан немедленно покинуть здание.

Специальные требования:

- Студент обязан выполнять только ту работу, которая поручена и соответствует теме занятия.
- Не отвлекаться самому и не отвлекать других посторонними разговорами.
- перевести мобильные телефоны в беззвучный режим работы.
- Не оставлять без присмотра включенное оборудование.
- При обнаружении неисправностей или поломок студент должен сообщить преподавателю.

Категорически запрещается:

- Работать на неисправном оборудовании.
- Приносить в учебные классы еду и
- напитки.
- Изменять настройки компьютеров.
- Загромождать проходы сумками и верхней одеждой.
- Класть на рабочие части устройства посторонние предметы.
- Искать и устранять самостоятельно неисправности приборов и устройств.
- В случае внезапного прекращения подачи электроэнергии немедленно сообщить преподавателю или дежурному инженеру.
- После окончания работы студент должен убрать за собой рабочее место.

Что бывает ЗА НАРУШЕНИЕ

- **Лица, нарушающие требования инструкции:**

1. **отстраняются от работы**
2. **направляются на повторный инструктаж с последующим оповещением деканата,**
3. **при повторном нарушении несут административную ответственность.**

Общие требования к учащимся на кафедре Медицинской кибернетики и информатики

1. Наличие белых халатов
2. Выполнение правил техники безопасности
3. Наличие конспектов лекций и теоретического материала занятий (обязательно)
4. **На занятиях по модульному контролю использование любых электронных средств (мобильные телефоны, планшеты и т.п.) НЕДОПУСТИМО**

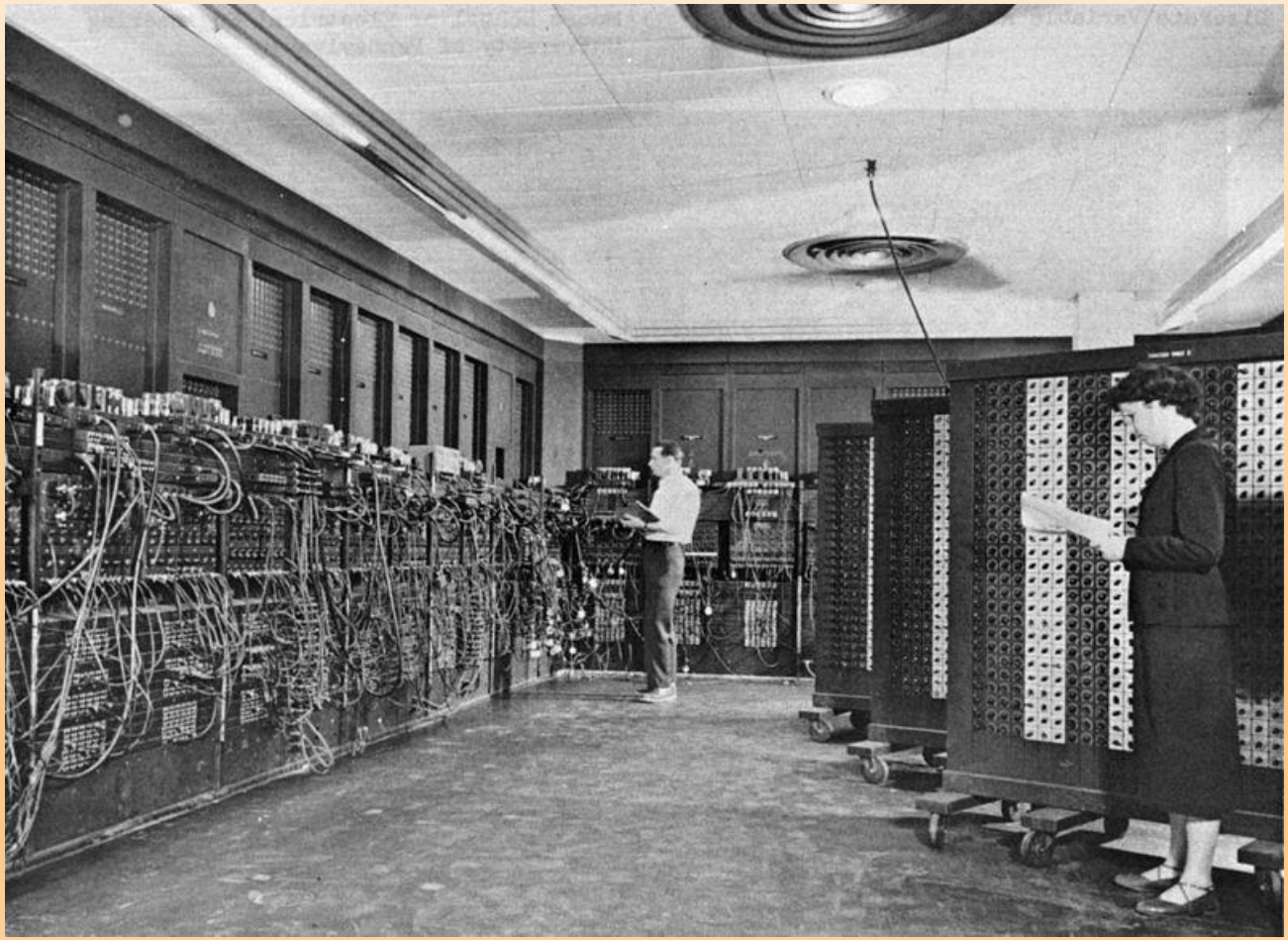
Алгоритмы программирования.

1.Принципы архитектуры ЭВМ фон Неймана

1. **Использование двоичной системы счисления в вычислительных машинах.**
2. **Программное управление ЭВМ.**
3. **Память компьютера используется не только для хранения данных, но и программ.**
4. **Ячейки памяти ЭВМ имеют адреса, которые последовательно пронумерованы.**
5. **Возможность условного перехода в процессе выполнения программы.**

Схема архитектуры фон Неймана

ЭВМ Эниак 1946г.



Компьютер 2019

2.Области применения компьютеров:

1. Обработка и хранение информации
2. Анализ данных в различных областях
3. Работа с Big Data
4. Обработка изображений
5. Мультимедиа – аудио-, видео- и звук
6. Телекоммуникации – сети и связь и т.д.
7. Робототехника
8. Исследования космоса
9. Использование в цифровой медицине и электронное здравоохранение и.т.д.


perfectsm!)e
PLANAR
iTero

Новейшие информационные медицинские технологии - Телемедицина

Единая государственная система здравоохранения РФ

ИЭМК-Интегрированная электронная медицинская карта»
ФХД-Финансово-хозяйственная деятельность
Задание 1 – весь состав ЕГИСЗ.

Компьютерная томография

Основные направления IT будущего:

- По данным на 2019-2021гг ведущих аналитических изданий Cnews и Gartner, в целом для IT-сферы, наиболее перспективными “горячими технологиями” являются направления:

1. аналитика больших данных
2. искусственный интеллект
3. облачные решения
4. интернет вещей
5. сети 5G
6. автономные системы (дроны, роботы, беспилотные автомобили и т. д.)
7. виртуальная и дополненная реальность
8. квантовые технологии
9. периферийные вычисления
10. цифровые двойники предприятий, организаций процессов и систем.

Темы докладов:

1. Системы искусственного интеллекта в мед и здравоохранении
2. Использование облачных решений в различных областях
3. интернет вещей в медицине
4. квантовые технологии и перспективы
5. цифровые двойники предприятий, организаций процессов и систем.
6. Суперкомпьютеры – виды, особенности, характеристики и использование

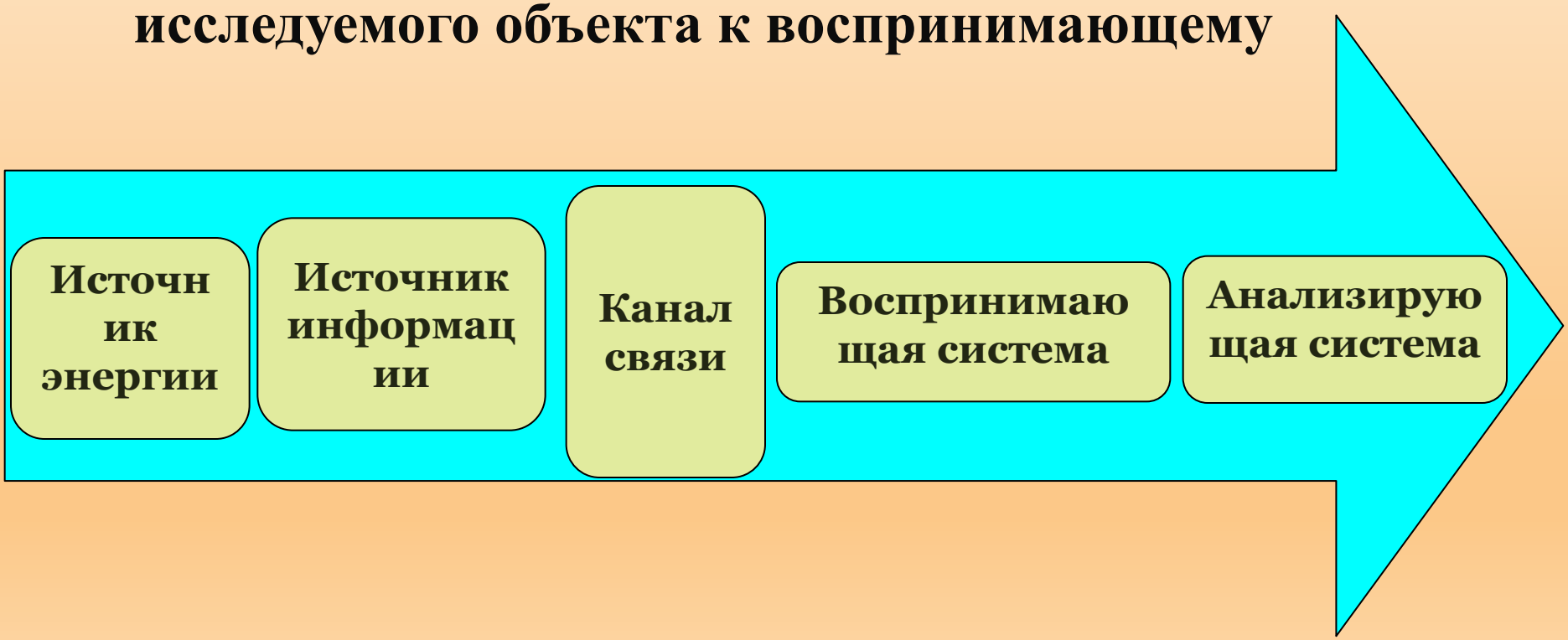
3. Основные понятия IT

Опр. 1. Информация – это сведения об объектах и явлениях окружающей среды, их параметрах, свойствах и состоянии, которые воспринимаются информационными системами (живыми организмами, машинами) в процессе жизнедеятельности и работы.

Принцип классификации	Виды информации
По способу восприятия	1. Визуальная 2. Аудиальная 3. Тактильная 4. Обонятельная 5. Вкусовая
По форме представления	1. Графика 2. Текст 3. Число 4. Звук
По значимости для пользователя	1. Массовая, специальная, личная; 2. Научная, управленческая, политическая итд.

Опр. 2.Информационный процесс

это перенос и восприятие информации от исследуемого объекта к воспринимающему



4. Основные понятия

- Опр. 3 . Компьютер – это комплекс тех. И программных средств, предназначенных для обработки информации в процессе решения вычислительных и информационных задач.
- Опр. 4. Информационная система – это система, предназначенная для хранения, поиска и обработки информации и соответствующие организационные ресурсы (чел., тех, фин.), которые обеспечивают работу ИС
- Опр. 4. Информационная технология -IT – это технология, использующая совокупность тех. И прогр. Средств, методов сбора, обработки, хранения и передачи данных, для получения качественно новой информации о состоянии объекта, процесса или явления.

Хранение информации в памяти компьютера Так человек видит изображение:

Компьютер видит набор «0» и «1» (первые две строчки файла):

1111 1111 1101 1000 1111 1111 1110 0000
0000 0000 0001 0000 0100 1010 0100 0110
0100 1001 0100 0110 0000 0000 0000 0001
0000 0001 0000 0000 0000 0000 0000 0001
0000 0000 0000 0001 0000 0000 0000 0000
1111 1111 1101 1011 0000 0000 0100 0011
0000 0000 0000 0011 0000 0010 0000 0010
0000 0011 0000 0010 0000 0010 0000 0011

Процессор — электронный блок, либо интегральная схема (микропроцессор), исполняющая машинные инструкции (код программ), главная часть аппаратного обеспечения компьютера

Общая схема компьютера

5.Единицы измерения информации

Опр. 8. Бит (bit) – базовая единица измерения информации, может содержать только одну двоичную цифру. Бит может принимать только два значения: «0» или «1».

Опр. 9. Байт (byte) – также единица количества информации, один байт равен восьми битам (1 Байт = 8 бит).

6. Основные направления теории кодирования

П.6 Алгоритмы и их свойства?

- **Опр. Алгоритм** – это совокупность действий, приводящих к достижению результата за конечное число шагов.

Свойства алгоритмов:

1. **Дискретность–** это разбиение алгоритма на ряд отдельных законченных действий-шагов.
2. **Детерминированность -** любое действие алгоритма должно быть строго и недвусмысленно определено в каждом случае. Например, алгоритм проезда к другу, если к остановке подходят автобусы разных маршрутов, то в алгоритме должен быть указан конкретный номер маршрута 5, указать точное число остановок.
3. **Конечность –** каждое действие в отдельности и алгоритм в целом должны иметь возможность завершения.
4. **Массовость –** один и тот же алгоритм можно использовать с разными исходными данными.
5. **Результативность –** алгоритм должен приводить к достоверному решению.

Основная цель алгоритмизации – составление алгоритмов для ЭВМ с дальнейшим решением задачи на ЭВМ.

Существует несколько способов записи алгоритмов

На практике наиболее распространены следующие формы представления алгоритмов:

1. словесная (запись на естественном языке);
2. псевдокоды (полуформализованные описания алгоритмов на условном алгоритмическом языке, включающие в себя как элементы языка программирования, так и фразы естественного языка, общепринятые математические обозначения и др.);
3. графическая (изображения из графических символов – блок-схема);
4. программная (тексты на языках программирования – код программы).

Виды алгоритмов

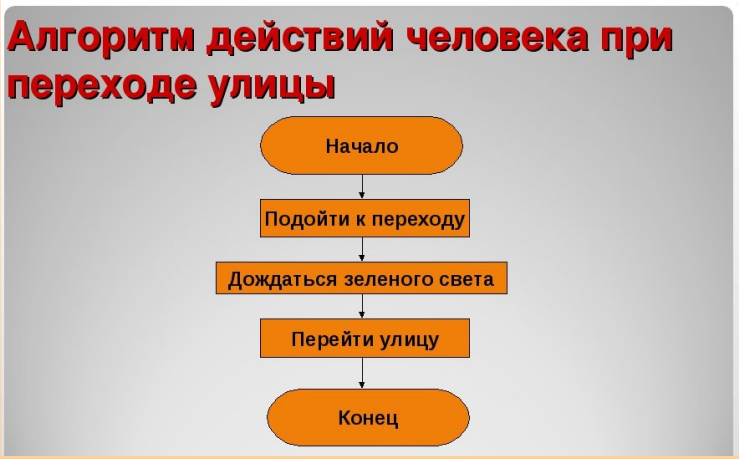
Три основных вида алгоритмов:

1. линейный алгоритм,
2. разветвляющийся алгоритм,
3. циклический алгоритм.

Словесный алгоритм.
1. Линейный алгоритм

Алгоритм перехода через улицу на светофоре:

1. Подойти к переходу
2. Дождаться включения зеленого света
3. Перейти улицу.



Задание

1.Написать словесный алгоритм включения компьютера

2. Написать словесный алгоритм измерения площади учебного стола прямоугольной формы

2.Разветвляющийся алгоритм – алгоритм с вариантами (с условиями)

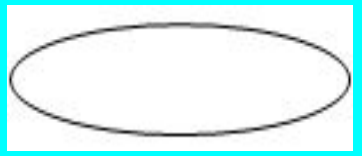
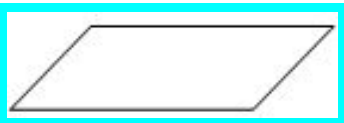
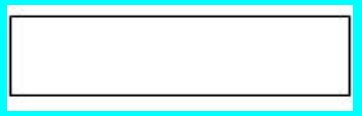
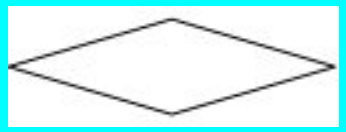
- Пример 1. Алгоритм перехода через улицу
- (какие варианты)?
- На доске

Задание

- 1. Написать словесный алгоритм с условием для покупки книги
- 2. Написать словесный алгоритм с условием для нахождения площади треугольника(в зависимости от того, что дано а) высота и основание, б) две стороны и угол между ними, в) три стороны

П.7. Графическая форма представления алгоритма. Блок-схемы.

Таб 2. Основные виды блоков представлены в блок-схемах

Назначение блока	Форма блока
1. начало и конец блок-схемы	
2. блок ввода данных	
3. блок выполнения действия	
4. блок условия	
5. блок вывода данных	

Пример 1. Александр хочет позвонить Пете по городскому телефону.
Надо: составить блок-схему, описывающую порядок действий Александра.

- **Пример 2.**
- Ученику требуется купить учебник.

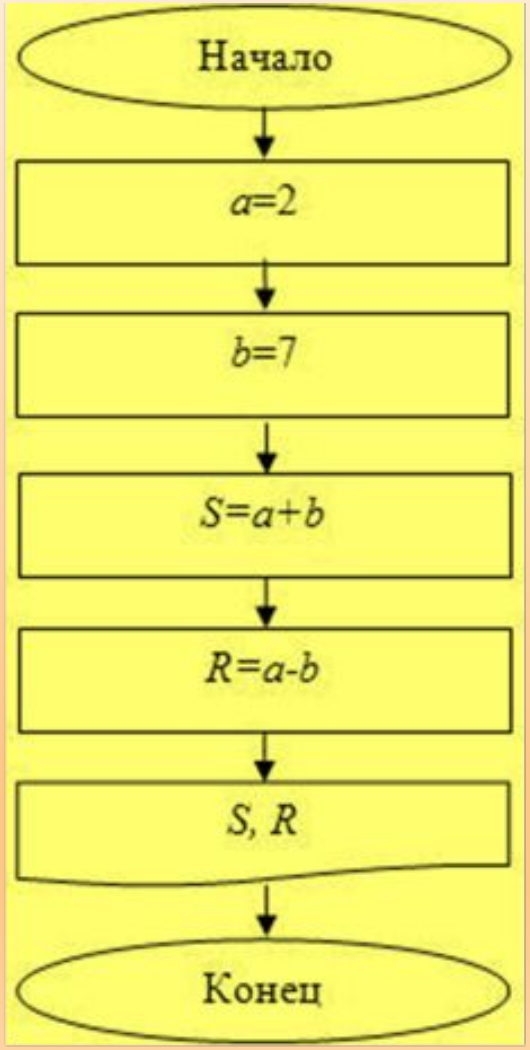
- Составить блок-схему, описывающую порядок действий ученика.

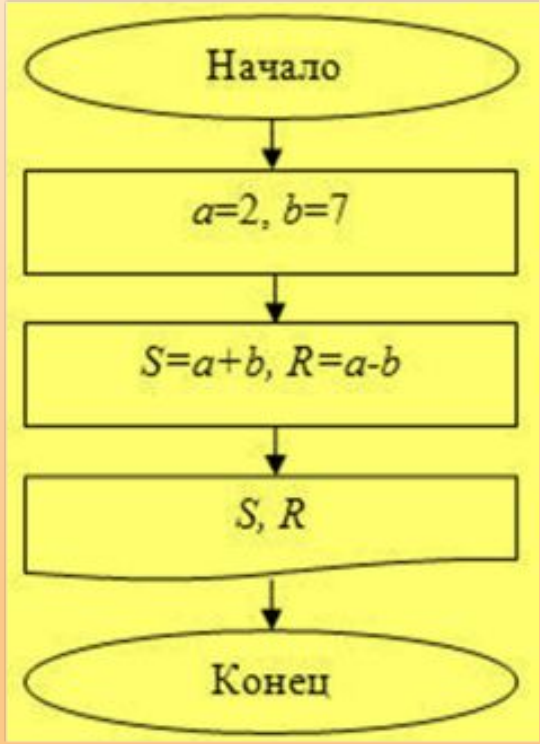
Блок-схема для примера 2

- **Пример 3.**

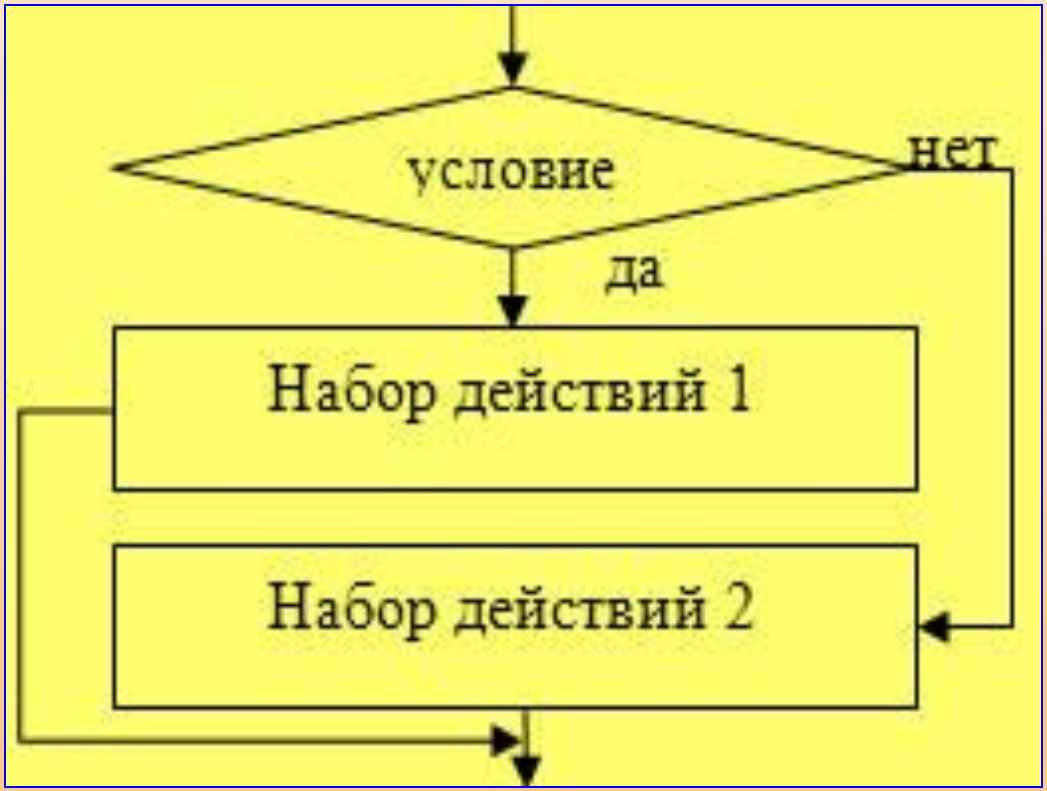
- Даны числа a=2, b=7 .
- Вычислить сумму **S** и разность **R** чисел.

Блок-схема к примеру 3





Графическая реализация разветвляющегося алгоритма



Пример 4.

Джон звонит Полу по городскому телефону, но трубку может взять не только Пол.

Составить блок-схему, описывающую действия Джона в этом случае.

Блок-схема для примера 4.

- **Пример 5**.
  Ученику требуется купить учебник. В магазине в наличии оказался нужный учебник в жесткой и мягкой обложке.

Составить блок-схему, описывающую действия ученика.

Блок-схема для примера 5

3.Циклические алгоритмы. Блок-схемы с циклом.

Циклы бывают двух видов:

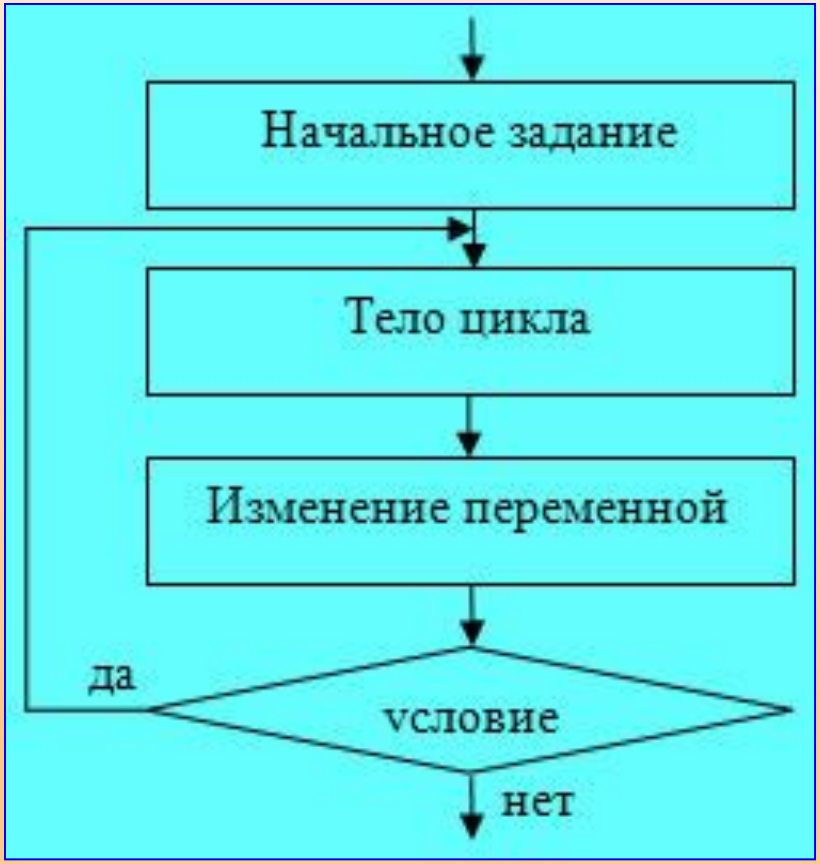
1. **с предусловием**
2. **с постусловием.**

1. В **цикле с предусловием сначала проверяется условие входа в цикл**, а затем выполняется тело цикла, если условие верно.

2. В **цикле с постусловием сначала выполняется тело цикла**, а потом проверяется условие.

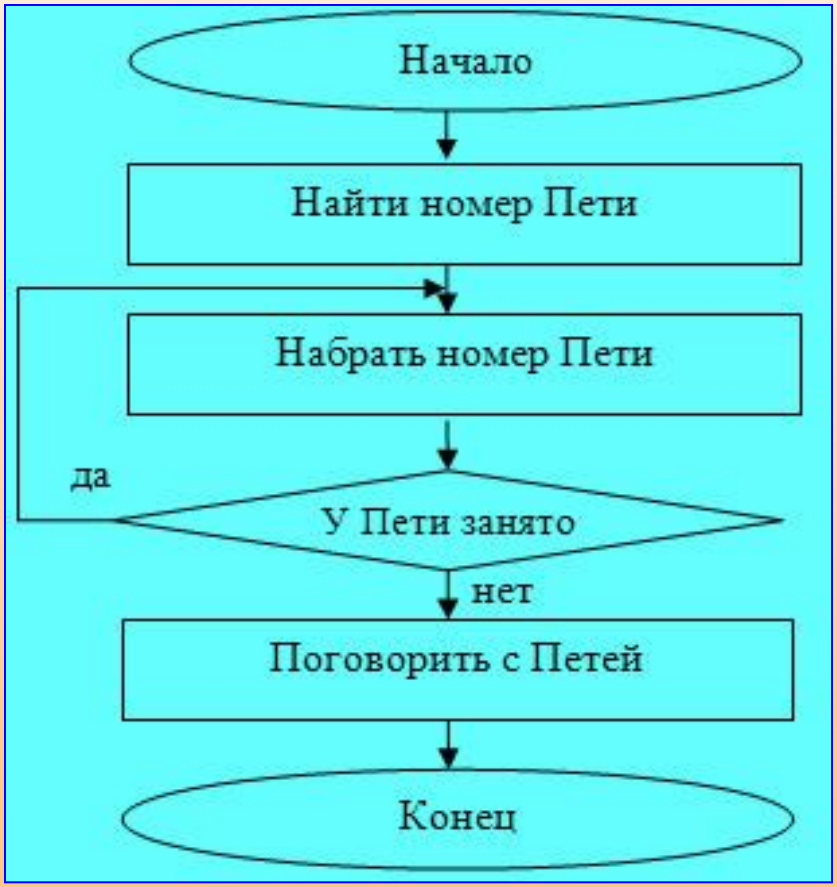
1.Цикл с предусловием

2.Цикл с постусловием.



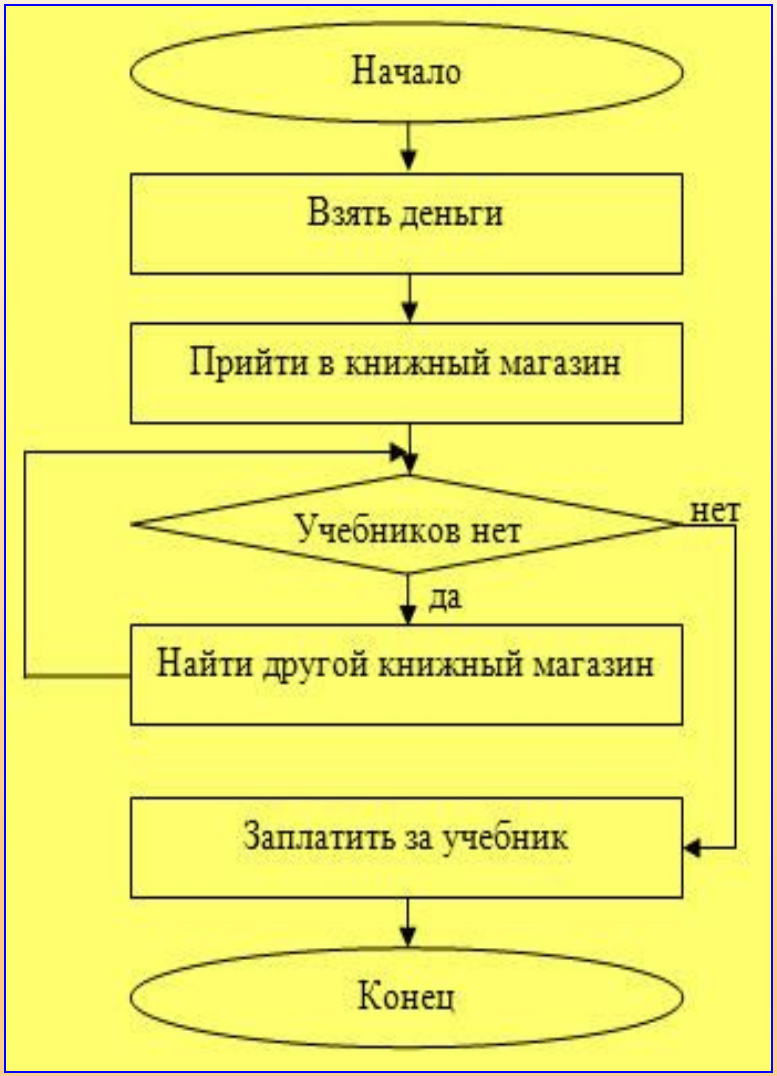
- **Пример 3.** Вася звонит Пете, но у Пети может быть занята линия. Составить блок-схему действий Васи в этом случае.

- Решение. Когда телефонная линия занята, то необходимо снова и снова набирать номер, пока Петя не закончит предыдущий разговор, и телефонная линия не окажется вновь свободной.

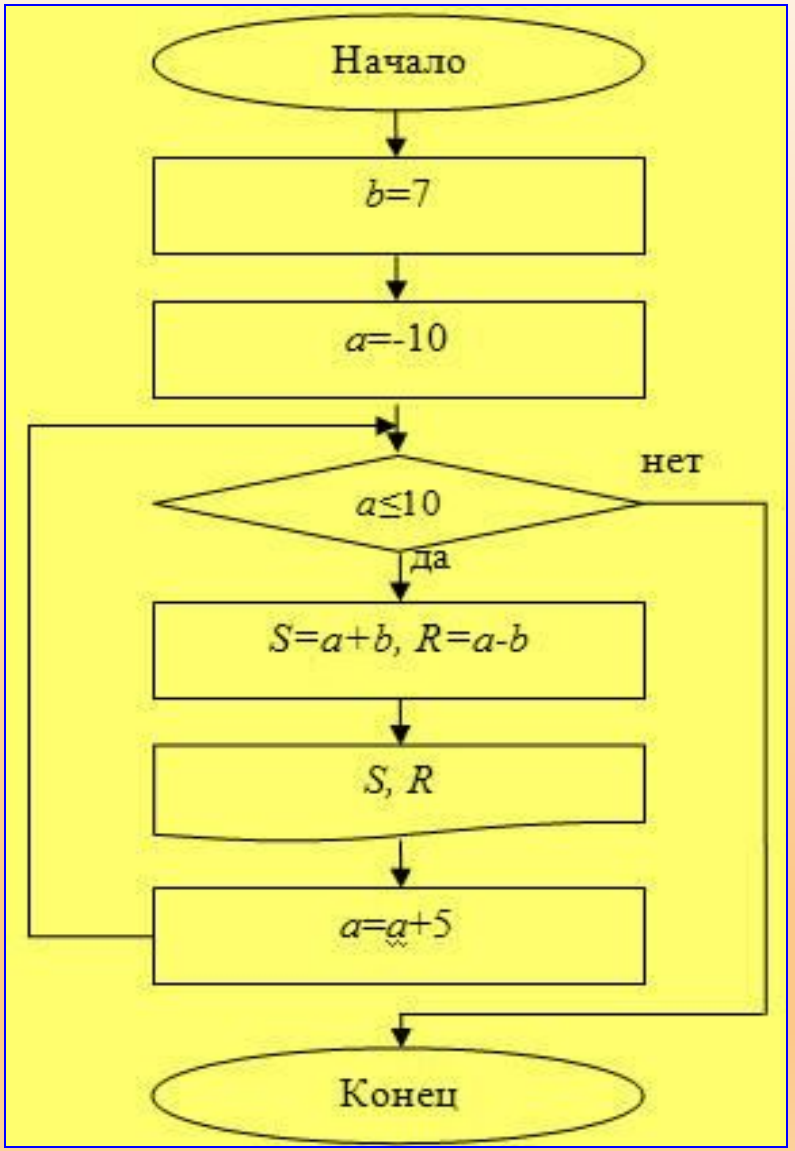


- **Пример 4.** Ученику требуется купить учебник. Составить блок-схему, описывающую действия ученика в случае, если учебника нет в ряде магазинов.

- Решение. Действия ученика: когда он приходит в первый и любой последующий магазины, то возможны два варианта – учебник имеется в наличии или учебника нет в продаже. Если учебника нет в продаже, то ученику следует пойти в другой книжный магазин и спросить данный учебник, и т.д. пока учебник не будет куплен.



- **Пример 5.** Даны числа a,b. Известно, что число a меняется от -10 до 10 с шагом 5,
  b=7 (не изменяется).
- Вычислить сумму S и разность R чисел a, b и для всех значений a и b.
- Решение. В отличие от примеров 3 и 4 здесь число a меняется от -10 до 10 с шагом 5. Это означает, что число a является переменной цикла.
- Сначала a=-10 – это первоначальное задание переменной цикла.



Проверочная работа 2.

Занятие 3.Циклические алгоритмы

- **Опр. . Тело цикла** – это набор инструкций, предназначенный для многократного выполнения.
- **Опр. . Итерация** – это единичное выполнение тела цикла.
- **Опр. . Переменная цикла**(счетчик цикла) – это величина, меняющаяся на каждой следующей итерации(повторении) выполнения цикла.

- **Каждый цикл должен содержать следующие необходимые элементы:**
- первоначальное задание переменной цикла (счетчика),
- проверку условия,
- выполнение тела цикла,
- изменение переменной цикла(счетчика).

Виды циклов:

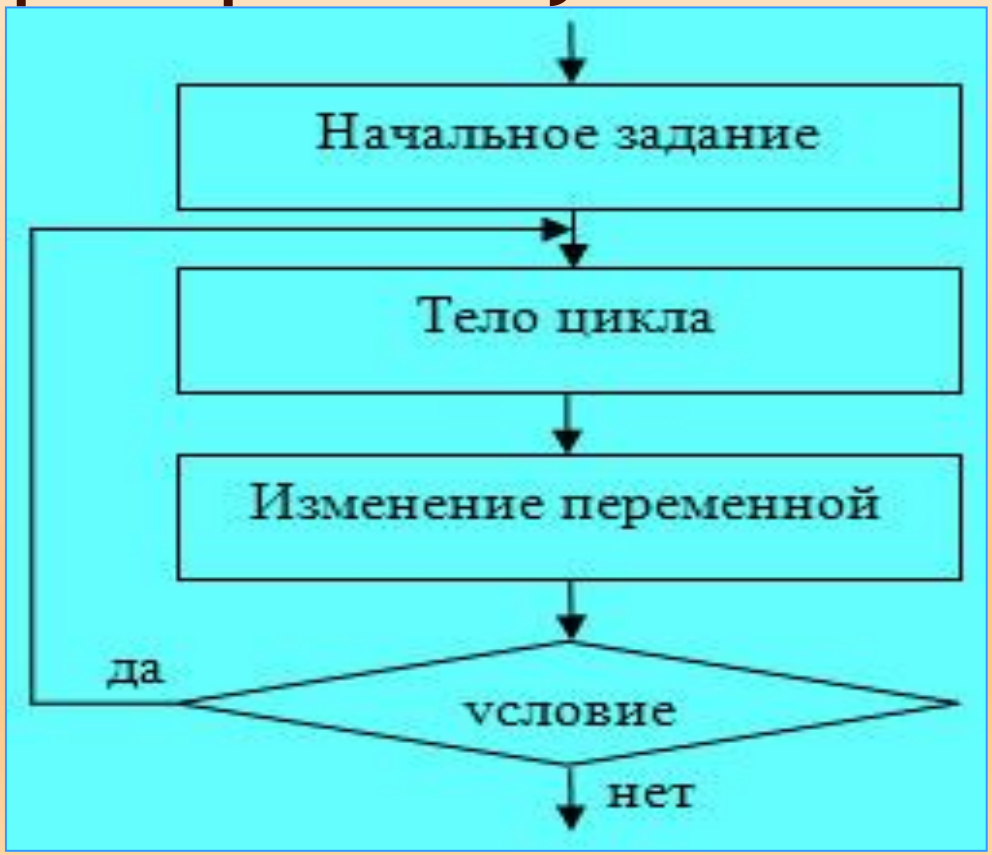
- 1. С предусловием – если условие верно, то выполняется тело цикла
- 2. С постусловием – пока условие верно выполняется тело цикла
- 3. Со счетчиком – указываем сколько раз конкретно выполнится тело цикла

Цикл с предусловием- сначала проверяется условие, потом выполняется тело цикла

Задача 1

- Найти сумму всех чисел от 5 до 45.

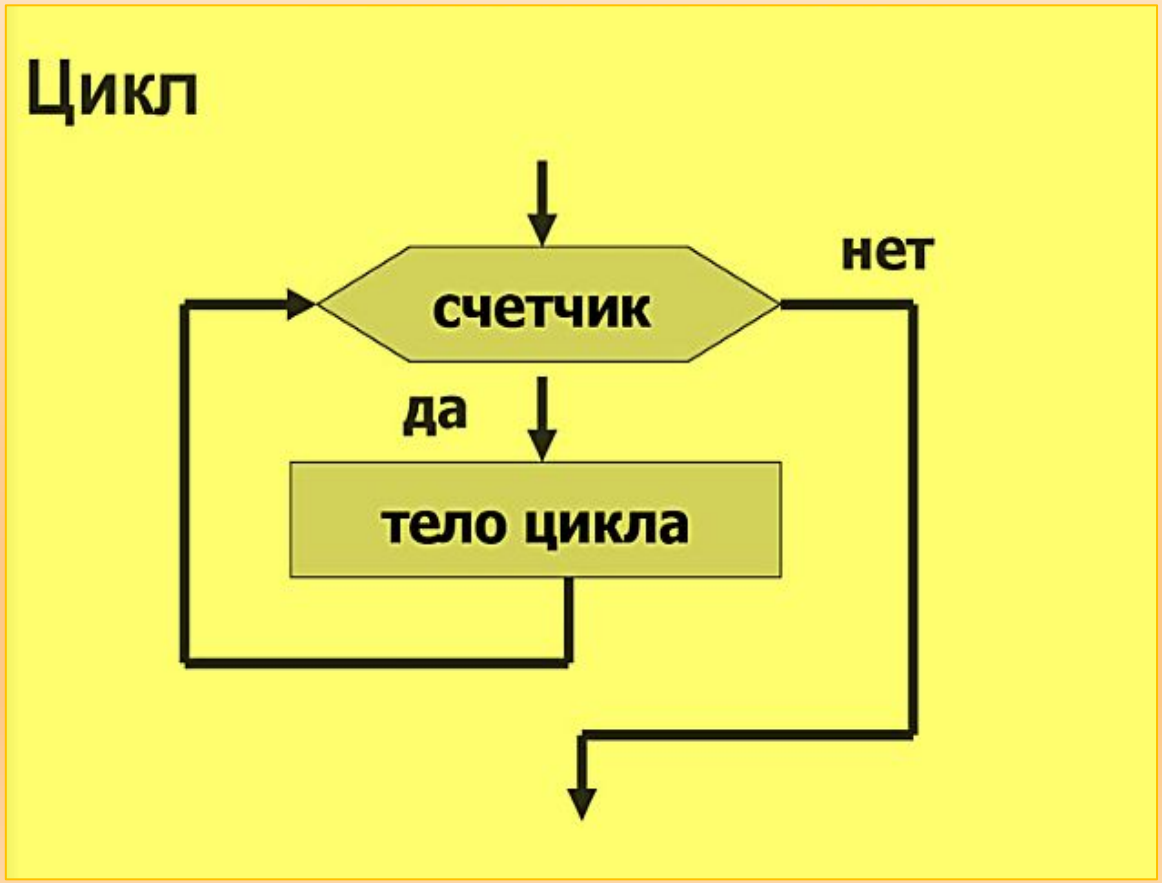
Цикл с постусловием
сначала выполняется тело цикла, затем проверяется условие



Задача 2

- Вычислить и вывести сумму степеней числа 2, пока она не станет больше 2500

Цикл со счетчиком. Выполнение цикла фиксированное число раз.



Задача 3

- Найти произведение m чисел

Расчет по алгоритму на тестовых данных.

- Задача 4. Произвести расчет по алгоритму задачи 3, представленному в графическом виде(блок-схеме) при m=7

Глава 2. Оценка эффективности алгоритмов. Псевдокод.

П. 1. **Оценка эффективности алгоритмов.**

Этапы:

1.Теоретический анализ

2.Проверочные испытания: измерение производительности

3. Реализация алгоритма

4.Измерение использования ресурсов. Измерения обычно выражаются как F(n) - функция от размера входных данных n.

Измерение использования ресурсов.

- **Измерения обычно выражаются как функция от размера входных данных** n**.**
- **Группа параметров А:**
- **1.Время работы алгоритма.**
- **Теоретический анализ -** используется асимптотический анализ временной сложности алгоритма, чтобы оценить время работы как функцию от размера входных данных. Результат обычно выражается в терминах О большое.
- **Практический анализ -** ***используются сравнительные тесты времени работы алгоритма.***
- Этот вид тестов существенно зависит также от выбора языка программирования, компилятора и его опций, так что сравниваемые алгоритмы должны быть реализованы в одинаковых условиях.
- **2. Память:** количество памяти для кода и количество памяти для данных, с которыми код работает. Для анализа алгоритма обычно используется анализ пространственной сложности алгоритма, чтобы оценить необходимую память времени исполнения как функцию от размера входа.
- Результат обычно выражается в терминах О большое

- **Группа Б.** Для компьютеров, питающихся от батарей (например, ноутбуков) или для очень длинных/больших вычислений (суперкомпьютеры), измеряют также:

1. Прямое потребление энергии: энергия, необходимая для работы компьютера.
2. Косвенное потребление энергии: энергия, необходимая для охлаждения, освещения, и т. п.

- **Группа С. Доп . измерения**:
    1. Размер передачи: пропускная способность канала может оказаться ограничивающим фактором. Для уменьшения количества передаваемых данных используют сжатие данных.
    2. Внешняя память: память, необходимая на диске или другом устройстве внешней памяти. Эта память может использоваться для временного хранения или для будущего использования
    3. Время отклика: параметр важен для приложений, работающих в реальном времени, когда компьютер должен отвечать быстро на внешние события.
    4. Общая стоимость владения: параметр важен, когда предназначен для выполнения одного алгоритма.
- **В результате, для решения конкретной задачи выбирается алгоритм, оптимальный (эффективный) в соответствии с выбранным критерием оптимальности**

- **Опр. Вычислительной сложностью алгоритма** называют функцию зависимости объёма работы, которая выполняется некоторым алгоритмом, от размера входных данных.
- **Опр. Временной сложностью алгоритма** называют функцию максимального количества элементарных операций, проделываемых алгоритмом для решения экземпляра задачи указанного размера, в зависимости от размера входных данных.

- Временная сложность алгоритма-

в худшем, наилучшем или среднем случае.

- Рассмотрение входных данных большого размера и оценка порядка роста времени работы алгоритма приводят к понятию **асимптотической сложности** алгоритма.
- Для записи асимптотической сложности алгоритмов используются асимптотические обозначения.

- В оценке нас интересует не само количество операций, а то, как оно будет расти, если мы начнём увеличивать N. Скажем, при сложении значений одномерного массива мы, задав любое N, получаем столько же операций. Такая зависимость записывается как O(N).
- O – это от слова Order, то есть порядок. O(N) значит, что для выполнения выбранного нами алгоритма с параметром N потребуется порядка N операций.

Обозначение	Интуитивное объяснение
	ограничена сверху функцией g(N) (с точностью до постоянного множителя) асимптотически

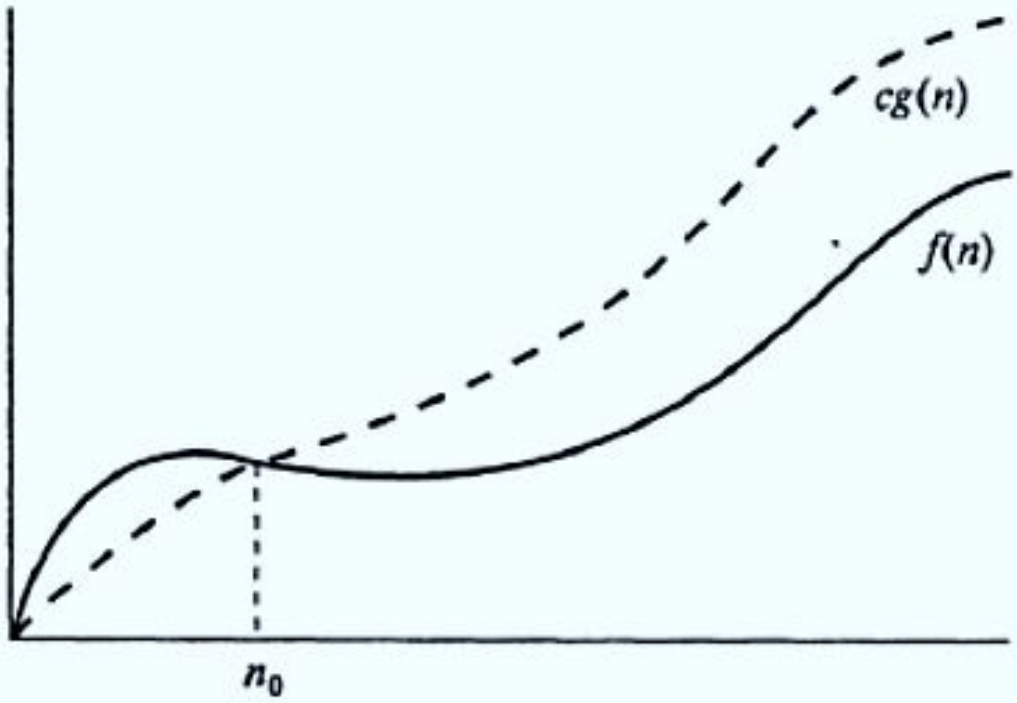


Рис. 1.3. Наглядное изображение нотации большого O. Функция f(n) есть O(g(n)), так как f(n)≤ cg(n) при n ≥ n0

Логарифмическая	Линейная	Квадратичная	Полиномиальная	Показательная
$O(\log n)$	$O(n)$	$O(n^2)$	$O(n^k)(k \geq 1)$	$O(a^n)(a > 1)$

Классы функций

Различные виды нотаций

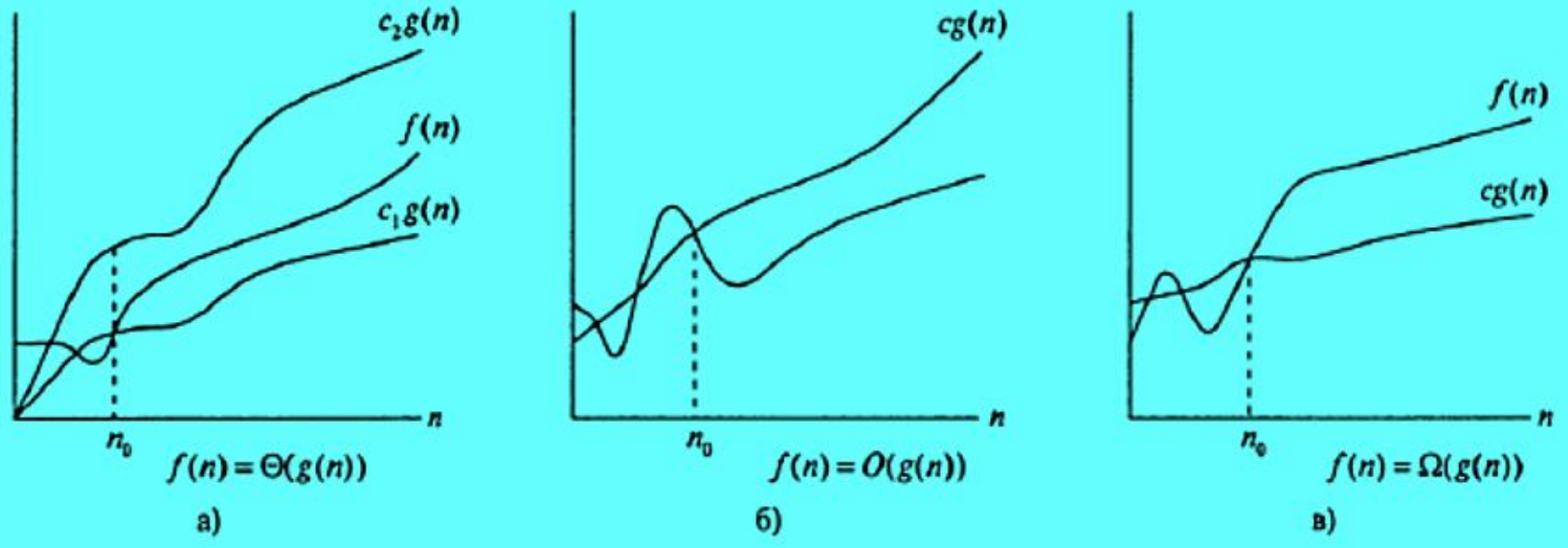


Рис. 1.4. Графические примеры Θ, O и Ω, обозначений; в каждой части рисунка в качестве n_0 используется минимально возможное значение, т.е. любое большее значение также сможет выполнить роль n_0

Проверочная работа

- Задача 1. Разработать блок-схему алгоритма :
- Вычислить сумму нечетных чисел от 1 до n Использовать цикл с предусловием.
- Задача 2. Сделать расчет по алгоритму задачи 1 для n=5
- Задача 3. Разработать блок-схему для задачи:
- Найти количество натуральных чисел, сумма которых не превышает 100. Цикл с постусловием

П. Основные понятия теории множеств

Опр.1. Множеством называется совокупность некоторых элементов, объединенных каким-либо общим признаком. Элементами множества могут быть числа, фигуры, предметы, понятия и т.п.

Опр.2.Объекты, из которых состоит множество, называются его элементами (например, буква К – элемент множества букв русского алфавита).

Опр.3. Множества, состоящие из одних и тех же элементов, называются равными (одинаковыми). Пишут А=В.

- **Опр.4. Множество, которое не содержит ни одного элемента, называется пустым и обозначается символом ∅.**
- **Опр.5. Множество B, состоящее из некоторых элементов данного множества A (и только из них), назы вается подмножеством (частью) этого множества.**

Это записывается так: $B \subset A$ или $A \supset B$. Говорят, что «B – подмножество A»

Основные числовые множества:

N={1,2,3,4,...} – множество натуральных чисел;

Z={...,-4,-3,-2,-1,0,1,2,3,4,...} – множество целых чисел (содержит все натуральные числа и числа, им противоположные), $N \subset Z$;

Q={x| , где $p \in Z$, $q \in N$} – множество рациональных чисел (состоит из чисел, допускающих представление в виде дроби), $N \subset Z \subset Q$;

R=$(-\infty;+\infty)$ – множество действительных чисел, $Q \subset R$ (кроме всех рациональных чисел, содержит иррациональные числа, содержащие в своей записи знаки радикалов:).

Основные свойства отношений включения между множествами:

1. $\varnothing \subset A$ для любого множества A;
2. $A \subset A$ для любого множества A (рефлексивность);
3. из того, что $B \subset A$ не следует $A \subset B$ (несимметричность);
4. если $A \subset B$ и $B \subset A$, то $A=B$ (антисимметричность);
5. если $A \subset B$ и $B \subset C$, то $A \subset C$ (транзитивность).

Операции над множествами

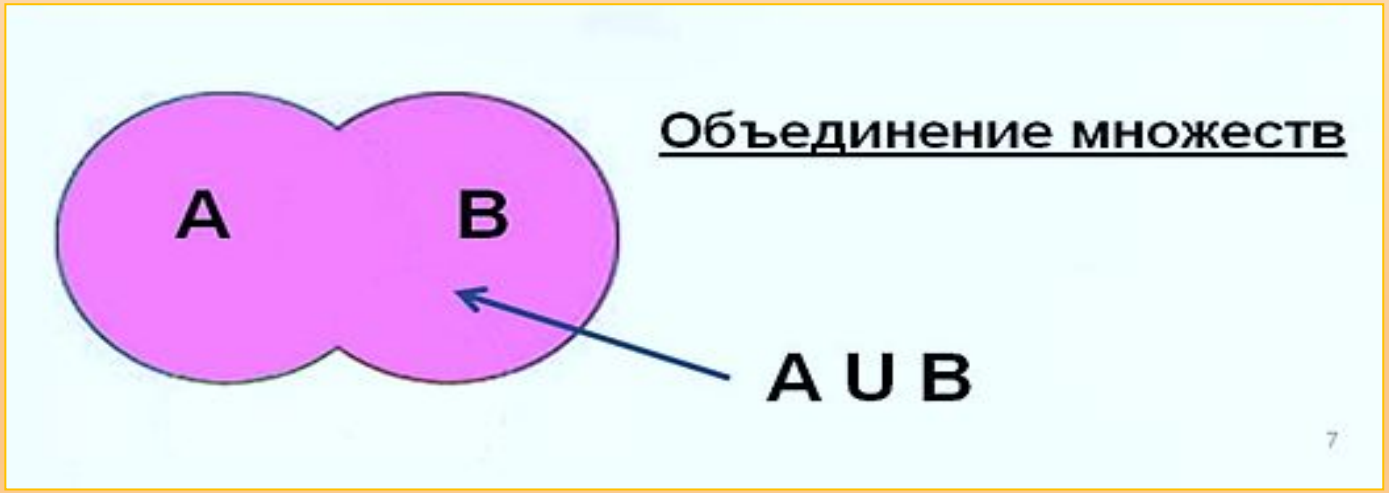
Опр.6 Пересечением множеств A и B называется множество C, состоящее из всех тех и только тех элементов, которые принадлежат каждому из данных множеств: $C=\{x \mid x \in A$ и $x \in B\}$. Обозначается, $A \cap B$.

1.Свойства операции пересечения:

1. $A \cap A = A$;
2. $A \cap \varnothing = \varnothing$;
3. $A \cap A' = \varnothing$;
4. $A \cap U = A$;
5. $A \cap B = B \cap A$.

Операция объединения множеств

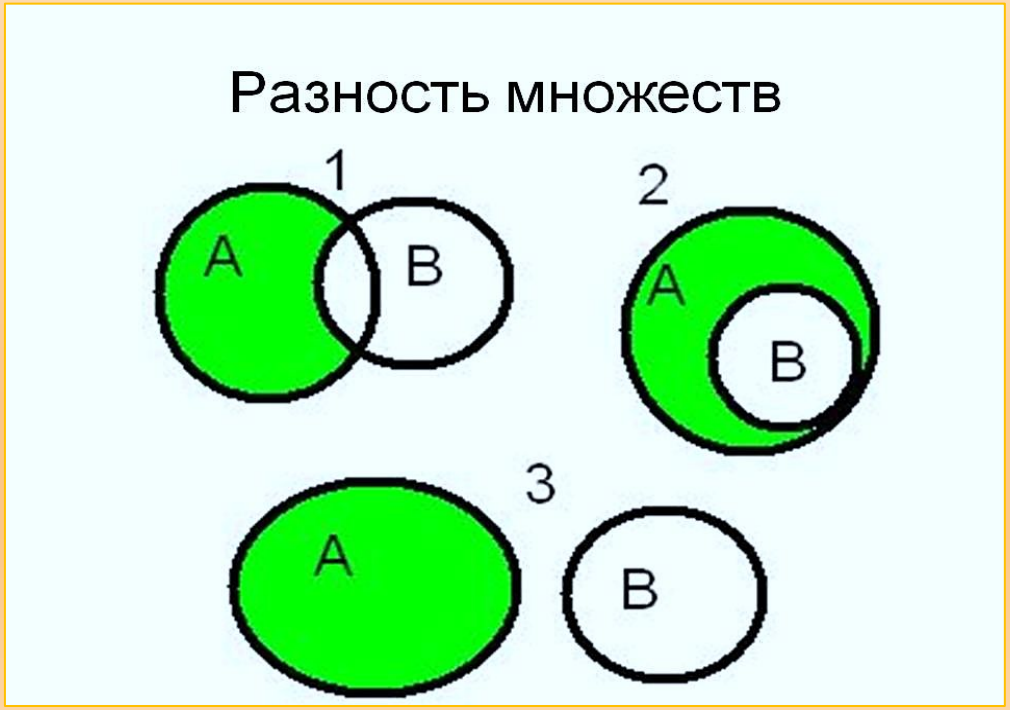
Опр. 7 Объединением множеств A и B называется множество C, которое состоит из всех элементов данных множеств A и B и только из них: $C=\{x \mid x \in A$ или $x \in B\}$. Обозначается, $A \cup B$.



2.Свойства операции объединения:

1. $A \cup A = A$;
2. $A \cup \varnothing = A$;
3. $A \cup A' = U$;
4. $A \cup U = U$;
5. $A \cup B = B \cup A$.

Опр.8 Разностью множеств A и B называется множество C, состоящее из всех элементов множества A, не принадлежащих множеству B: $C=\{x \mid x \in A \text{ и } x \notin B\}$. Обозначается, $A \setminus B$.



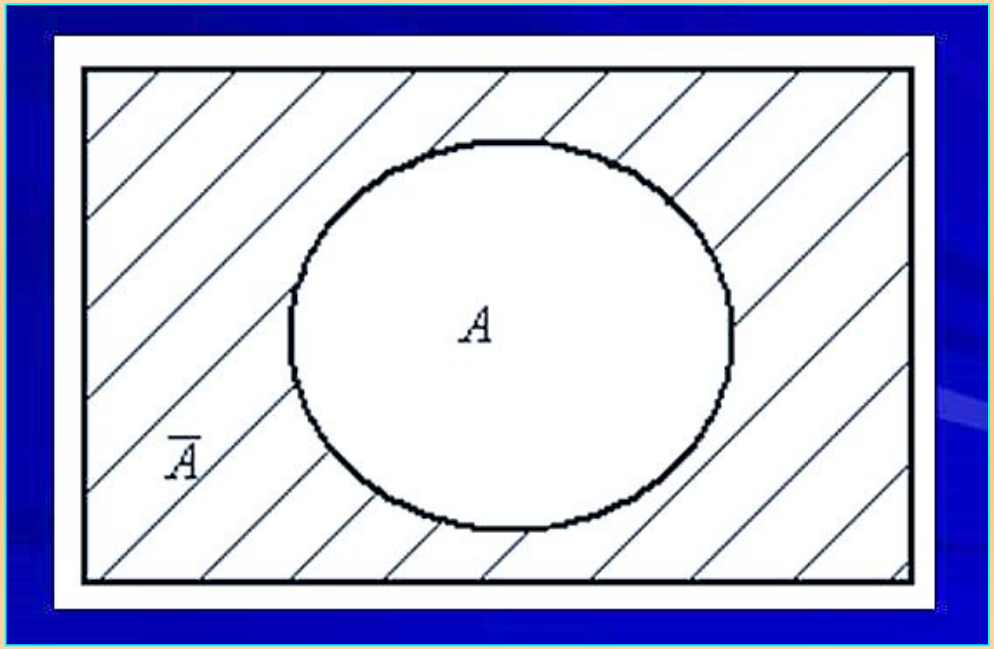
3.Свойства операции разности:

1) $A \setminus A = \varnothing$;
2) $A \setminus \varnothing = A$;
3) $A \setminus A' = A$;
4) $A \setminus U = \varnothing$;
5) $U \setminus A = A'$;
6) $\varnothing \setminus A = \varnothing$;
7) $A \setminus B \neq B \setminus A$.

Дополнение множества А

Опр.9 дополнением множества А называется разность $\overline{A}$ =U/A и читается «не-А» .

Дополнением множества А называется множество, состоящее из всех элементов, не принадлежащих множеству А.



Приоритет выполнения операций

Порядок выполнения операций над множествами:

1. **Равноправные действия слева направо,**
2. **Сначала Скобки,**
3. **далее Операция дополнения,**
4. **затем Пересечение,**
5. **затем равноправные операции Объединения и Вычитания.**

Задачи по теории множеств.

Проверочная работа по теории множеств.

- На листочках

П.2. Псевдокод Приведем основные управляющие структуры псевдокода

Таб 1. Базовые управляющие структуры псевдокода	
Название структуры	**Псевдокод**
1.Присваивание	переменная = число
2. Ввод	ввод(переменная)
3. Вывод	вывод(переменная),вывод("фраза")
4. Ветвление	если **условие** то **действие1** иначе **действие2**
5. Повторение	пока условие начало Пока действие конец пока

Пример 1 хорошего начального псевдокода

1. Если номер счета и пароль подходят, то показать основную информацию счета.

Пример псевдокода 2.

Программа должна заменять сочетание букв "to" в текстовом файле на сочетание “on”.

Программа прочитает каждую строку в этом файле, поищет нужное сочетание в каждой строке и заменит его на другое.

Первоначальный набросок псевдокода может выглядеть так:

1. Начало
2. открыть файл
3. в каждой строке файла:
   1) найти сочетание
   2) удалить сочетание
   3) вставить другое сочетание
4. закрыть файл
5. Конец

Этапы написания псевдокода

1. **Сначала опишите цель процесса.** Если с помощью псевдокода можно решить задачу, он считается завершенным.
2. **Напишите первые шаги, которые подготовят вас к инструкциям.** Обычно в первой части кода определяются переменные и другие элементы, которые делают алгоритм рабочим.

3. **В процессе описания алгоритма решения задачи на псевдокоде, пишите только по одному обращению в строке.** Каждое обращение в псевдокоде должно задавать компьютеру лишь одно действие.
4. **Пишите большими буквами первое слово основной функции.** Важными ключевыми словами могут быть READ, WRITE, IF, ELSE, ENDIF, WHILE, ENDWHILE, REPEAT и UNTIL.
5. **Когда пишите - отделяйте шаги блоками.** С помощью блоков можно упорядочивать информацию или объединять ее.

Переход от псевдокода к программе.

1. Выполните расчет решения задачи по вашему псевдокоду с тестовыми входными данными и разберитесь в том, как он работает.

2. Исправьте ошибки.

3. На базе вашего алгоритма в псевдокоде напишите программу на нужном вам языке программирования.

Команды языков программирования в псевдокоде

Важными ключевыми словами могут быть:

1. READ,
2. WRITE,
3. IF,
4. ENDIF,
5. ELSE,
6. WHILE,
7. ENDWHILE,
8. REPEAT
9. UNTIL

Конструкции языков в псевдокоде

1. **IF Условие THEN Команды**.
   Это означает, что отдельная инструкция будет срабатывать лишь в том случае, если будет выполняться отдельное условие.

2. **WHILE Условие DO Команды**. Это означает, что инструкцию нужно повторять снова и снова, пока условие не перестанет выполняться.

3. **DO Команды WHILE Условие**. Эта конструкция похожа на while CONDITION do INSTRUCTION. В первом случае условие проверяется до того, как начинает

4. **REPEAT Команды UNTIL Условие**

5. **FOR a = NUMBER1 TO NUMBER2 DO** Команды.
   Это означает, что переменная "**a**" будет автоматически принимать значение **NUMBER1**. "a" будет увеличиваться на единицу в каждом шаге, пока значение переменной не достигнет **NUMBER2**. Для обозначения переменной можно использовать любую другую букву.

Структуризация алгоритма в псевдокоде и программе

BLOCK1

BLOCK2

BLOCK3

BLOCK2

BLOCK3

BLOCK1

Задача 1. Алгоритм решения квадратного уравнения на псевдокоде

Решение

Псевдокод (пример)

```
ввод a, b, c
D = b2 - 4ac
если D < 0 то
  вывод Корней нет
иначе
  если D > 0 то
    x1 = …
    x2 = …
    вывод x1, x2
  иначе
    x1 = …
    вывод x1
  конец если
конец если
```

Задача 2. Написать алгоритм в псевдокоде для расчета значения выражения **a=k+b/n. k,b,n вводятся по одному.**

Задача 3. Сделать расчет по алгоритму на тестовых данных k=2, b=3, n=5.

Проверочная работа 2.

Задача 1.Посчитать в тексте кол-во вхождений последовательности “шин” во входной строке символов. Признак конца строки – пробел.

Начертить блок-схему для алгоритма решения задачи

Задача 2. Написать псевдокод для алгоритма решения задачи 1. Можно использовать команды и структуры языков программирования, рассмотренные на занятии.

Задача 3.Сделать построчный тестовый расчет по псевдокоду для текста примера 1 на слайде:

Машина Тьюринга — абстрактная вычислительная машина. Была предложена Аланом Тьюрингом в 1936 для формализации понятия алгоритма

Глава 3. Основные понятия формальных определений алгоритма.

П.1. Основные понятия

А

Опр. 1. Символ — любой печатный знак

Опр.2. Алфавит — конечное множество символов (русский алфавит, греческий, др.)

Греческий алфавит

α альфа	β бета	γ гамма	δ дельта	ε эпсилон	ζ дзета	η эта	θ тета
ι йота	κ каппа	λ лямбда	μ мю	ν ню	ξ кси	ο омикрон	π пи
ρ ро	σ сигма	τ тау	υ ипсилон	φ фи	χ хи	ψ пси	ω омега

Опр. 3. Слово — конечная последовательность символов из алфавита:

АННА

Опр. 4. Длина слова — количество символов в слове - 11 символов.

Кибернетика

1.Объектами алгоритма являются **слова** и только они.

2.Алгоритм — описание способа преобразования одного (входного) слова в другое (выходное).

- **Опр 5.** Алгоритмы B и C эквивалентны в алфавите A (на множестве A*), если к каждому слову из A* они – либо одновременно неприменимы, – либо одновременно применимы и выдают одинаковое выходное слово.

- **Теорема 1. Не существует алгоритма, который для заданных двух алгоритмов по их записи определяет, являются они эквивалентными (в заданном алфавите) или нет. (рассматриваем без доказательства)**

Опр.6. Алгоритмические языки это формальные языки используемые для записи, реализации или изучения алгоритмов.

Опр. 7. Языки программирования это формальные языки, предназначенный для записи компьютерных программ.

Описание языков

- **Описание алгоритмического языка** – это описание, которое содержит:
  1. **алфавит — набор символов, используемых при записи программ,**
  2. **синтаксис — правила, указывающие, какие тексты допустимы в языке, а какие — нет.**
  3. **семантику — правила, определяющие смысл синтаксически правильного текста.**
  4. **Прагматику.**

- **Описание языка программирования** – это описание, которое содержит набор лексических, синтаксических и грамматических правил, определяющих внешний вид программы и действия, которые выполнит исполнитель (обычно — компьютер) под её управлением.

ОСНОВНЫЕ СПОСОБЫ КОМПОЗИЦИИ АЛГОРИТМОВ:

1) **Суперпозиция алгоритмов**. При суперпозиции двух алгоритмов A и B выходное слово первого алгоритма рассматривается как входное слово второго алгоритма.

2) **Объединение алгоритмов.** Объединением алгоритмов A и B в одном и том же алфавите называется алгоритм C, преобразующий любое слово P, содержащееся в пересечении областей определения алгоритма A и B в записанные рядом слова

3) **Разветвление алгоритмов**. Разветвление представляет собой композицию D трёх алгоритмов A,B и C, причём область определения алгоритма D является пересечением областей определения трёх алгоритмов A,B,C.

4) **Итерация (повтор).** Представляет собой такую композицию C двух алгоритмов A и B, что для любого входного слова P, соответствующего слова, получается в результате последовательного многократного применения алгоритма A до тех пор, пока не получится слово, образуемое алгоритмом B.

Рекурсия. Рекурсивная функция

- **Опр. 8.** В программировании рекурсия — вызов функции (процедуры) из неё же самой, непосредственно (*простая рекурсия*) или через другие функции (*сложная* или *косвенная рекурсия*),

- Опр.9. Количество вложенных вызовов функции или процедуры называется глубиной рекурсии.

- Рекурсивная программа позволяет описать повторяющееся или даже потенциально бесконечное вычисление, причём без явных повторений частей программы и использования циклов.

Пример – вычисление факториала.

Пример 1 – написать псевдокод для вычисления n!

- **Опр. 10.** Частично-рекурсивной функцией называется функция, которую можно построить из набора простых функций путем использования трех операций: **суперпозиции, примитивной рекурсии и минимизации**.

  Примеры простых функций:
- 1) Тождественно равная нулю функция **O**(x) = O
- 2) Набор функций $I_n^m(x_1, x_2, \ldots\ldots, x_n) = x_m$
- 3) Функция **S**(x) = x + 1.

Опр.11

Функция f с натуральными аргументами и значениями называется *вычислимой*, если существует алгоритм, её вычисляющий, то есть такой алгоритм A, что

- если $f(n)$ определено для некоторого натурального n, то алгоритм A останавливается на входе n и печатает $f(n)$;
- если $f(n)$ не определено, то алгоритм A не останавливается на входе n.

- **Понятие вычислений тесно связано с понятием алгоритма,** потому, что вычисления являются частным случаем процессов преобразования информации - информации, представленной в форме чисел.
- Оказывается, что к этому частному случаю может быть сведен любой алгоритм.
- ***1) Каждый алгоритм, определяя частичное отображение A* → A* ,***
  определяет вычислимую функцию

- ***2) Каждая вычислимая функция f(n): N → N определяет алгоритм - частичное отображение на множестве слов алфавита A: A* → A****

- ***ТЕЗИС ЧЕРЧА*** **-** совокупность всех частично вычислимых функций совпадает с совокупностью частично-рекурсивных функций.

Тезис Черча постулирует возможность получения значений любой вычислимой частичной функции процессом, использующим лишь операции суперпозиции, примитивной рекурсии, минимизации и вычисления функций **o**(x), **S**(x), **I** n m (x).

Термин **рекурсивная функция** в теории вычислимости используется для обозначения трёх классов функций:

1. примитивно рекурсивные функции;
2. общерекурсивные функции;
3. частично рекурсивные функции.

Алгоритм — описание способа преобразования одного (входного) слова в другое (выходное).

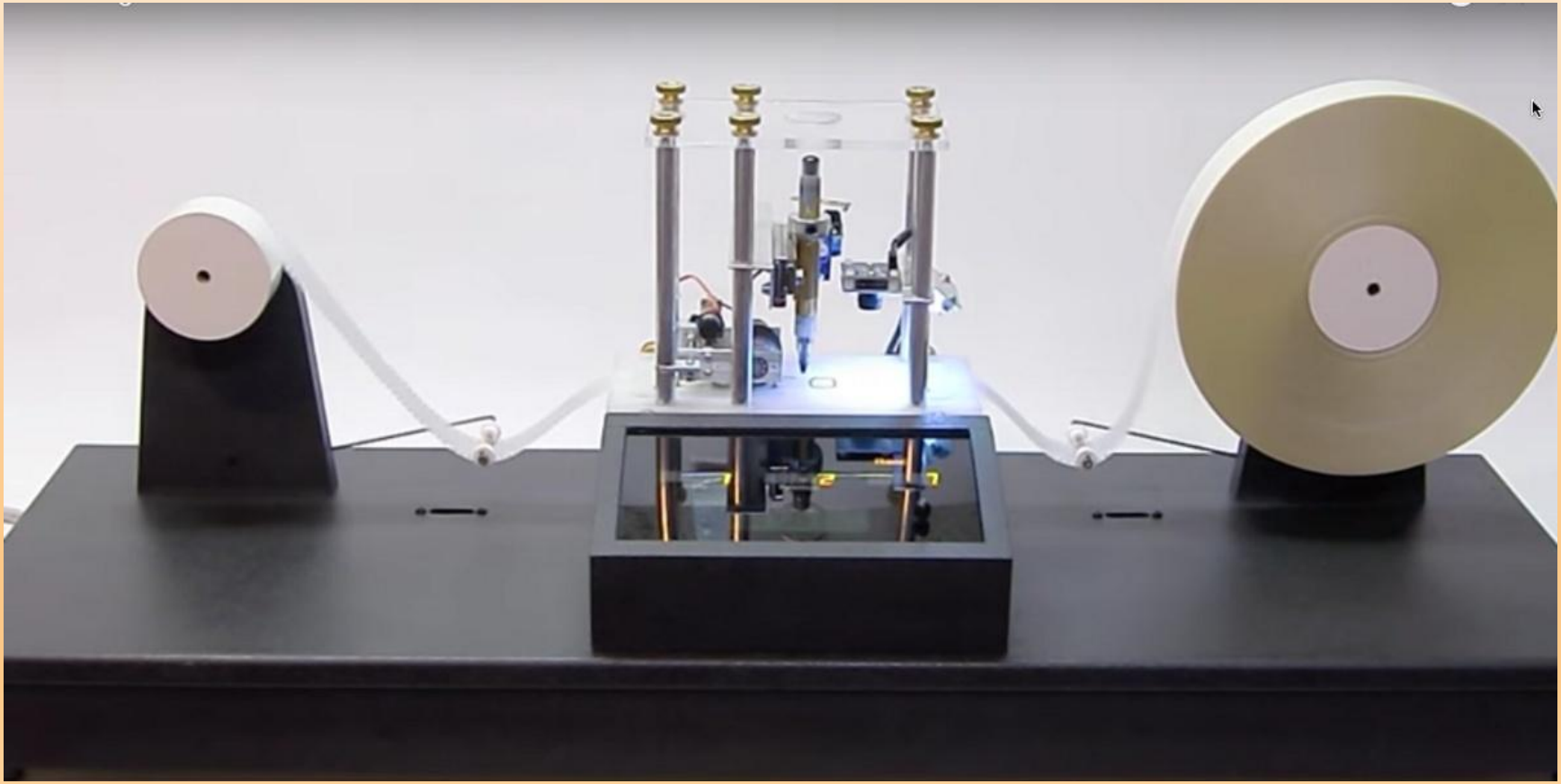
Рассмотрим алгоритмы, как процессы преобразования конечных слов в конечном алфавите.

Такие формализованные процессы мы будем считать эквивалентными общему понятию алгоритма.

Два подхода к формальному определению алгоритма:

1. **Машина Тьюринга**
2. **Нормальные алгоритмы Маркова.**

П.2.Формальное определение алгоритма
1). Машина Тьюринга.

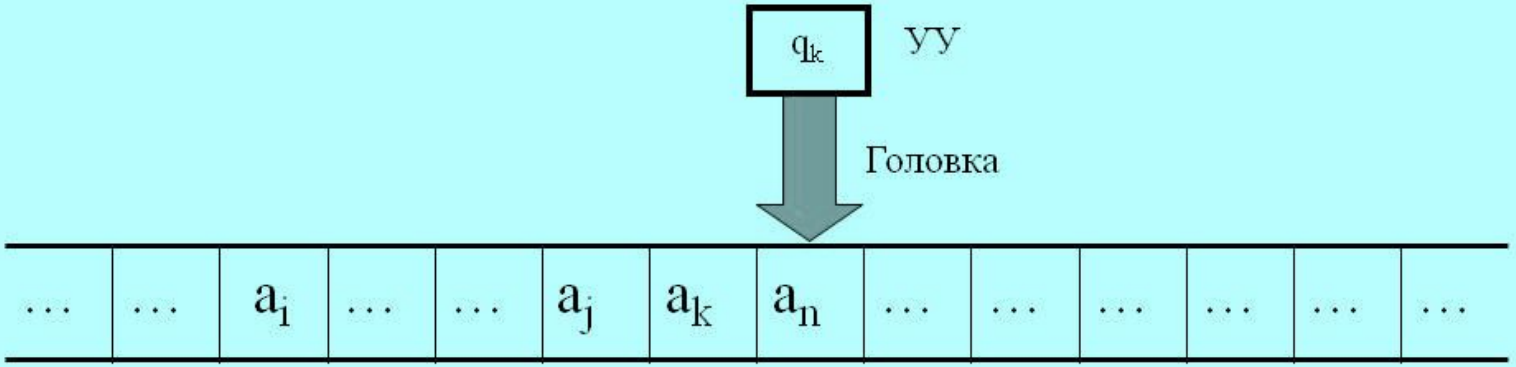


Машина Тьюринга

Это абстрактная машина, предложенная Тьюрингом в 1936 г. в качестве универсальной алгоритмической модели.

Она состоит из трех частей:

- лента;
- головка;
- управляющее устройство (УУ).



Пару из видимого символа (S) и текущего состояния автомата (q) будем называть ***конфигурацией*** и обозначать $\langle S, q \rangle$.

Автомат может выполнять три элементарных действия:

1) записывать в видимую клетку новый символ (менять содержимое других клеток автомат не может);

2) сдвигаться на одну клетку влево или вправо («перепрыгивать» сразу через несколько клеток автомат не может);

3) 3) переходить в новое состояние. Ничего другого делать автомат не умеет, поэтому все более сложные операции так или иначе должны быть сведены к этим трём элементарным действиям.

Такт работы машины Тьюринга

МТ работает ***тактами*** (*по шагам*), которые выполняются один за другим. На каждом такте автомат МТ выполняет три следующих действия, причем обязательно в указанном порядке:

1) записывает некоторый символ S' в видимую клетку (в частности, может быть записан тот же символ, что и был в ней, тогда содержимое этой клетки не меняется);

2) сдвигается на одну клетку влево (обозначение – L, от *left*), либо на одну клетку вправо (обозначение – R, от *right*), либо остается неподвижным (обозначение – N).

3) переходит в некоторое состояние q' (в частности, может остаться в прежнем состоянии).

Формально действия одного такта будем записывать в виде тройки:

$$S', [L,R,N], q'$$

Программа для машины Тьюринга

	S_1	S_2	…	S_i	…	S_n	Λ
q_1							
…							
q_j				S′, [L, R, N], q′			
…							
q_m							

Слева перечисляются все состояния, в которых может находиться автомат, сверху – все символы (в том числе и Λ), которые автомат может видеть на ленте. (Какие именно символы и состояния указывать в таблице – определяет автор программы.)

На пересечениях же (в ячейках таблицы) указываются те такты, которые должен выполнить автомат, когда он находится в соответствующем состоянии и видит на ленте соответствующий символ.

Правила выполнения программы:

К началу выполнения программы машина Тьюринга находится в начальной конфигурации:

1) на ленте записано входное

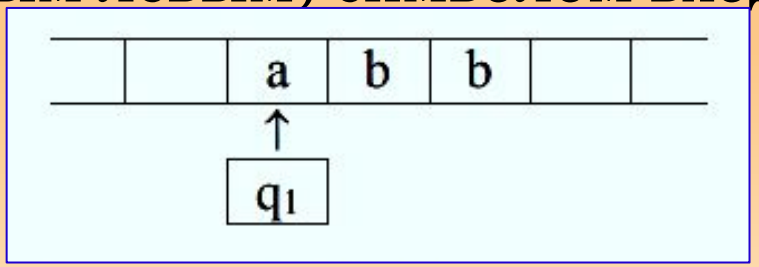
слово, к которому будет применена программа.

Входное слово – это конечная последовательность символов, записанных в соседних клетках ленты; внутри входного слова пустых клеток быть не должно, а слева и справа от него должны быть только пустые клетки.

Пустое входное слово означает, что все клетки ленты пусты.

2) автомат установлен в состояние q1

(указанное в таблице первым) и размещен под его первым (самым левым) символом входного слова:



3. В таблице отыскивается ячейка на пересечении первой строки
(т.к. автомат находится в состоянии q1) и того столбца, который соответствует первому символу входного слова (это необязательно левый столбец таблицы),
и выполняется такт, указанный в этой ячейке. В результате автомат окажется в новой конфигурации.
4. Действие 3. повторяется для новой конфигурации
5. Такт останова-это такт, который ничего не меняет: автомат записывает в видимую клетку тот же символ, что и был в ней раньше, не сдвигается и остается в прежнем состоянии

Окончание работы программы и результат

- В целом возможны два исхода работы МТ над входным словом:
- 1) Первый исход -когда в какой-то момент МТ останавливается (попадает на такт останова**).**
- **В таком случае говорят, что МТ применима к заданному входному слову. А то слово, которое к этому моменту получено на ленте, считается выходным словом, т.е. результатом работы МТ, ответом**
- 2) Второй исход – «плохой»- МТ зацикливается, никогда не попадая на такт останова
- **В этом случае говорят, что МТ неприменима к заданному входному слову.**
- **Результата нет.**

гипотеза Тьюринга – всякий разрешимый алгоритм может быть представлен в виде машины Тьюринга

Опр. Алгоритмически разрешимые задачи – это задачи которые могут решаться машиной Тьюринга за конечное число шагов.

Гипотеза Тьюринга подтверждается:

1. возможностью построения машин, реализующих конкретные алгоритмы

2. доказательством возможности построения машин, реализующих различные композиции алгоритмов и их фрагменты.

П.4. Нормальные алгоритмы Маркова

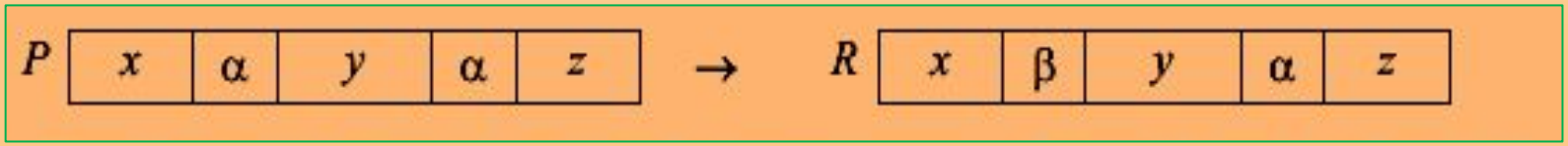
Формулой подстановки называется запись вида $\alpha \to \beta$ (читается «α заменить на β»), где α и β – любые слова (возможно, и пустые). При этом α называется левой частью формулы, а β – правой частью.

Сама ***подстановка*** (как действие) задается формулой подстановки и применяется к некоторому слову *P*. Суть операции сводится к тому, что в слове *P* отыскивается часть, совпадающая с левой частью этой формулы (т.е. с α), и она заменяется на правую часть формулы (т.е. на β). При этом остальные части слова *P* (слева и справа от α) не меняются. Получившееся слово *R* называют ***результатом подстановки***. Условно это можно изобразить так:

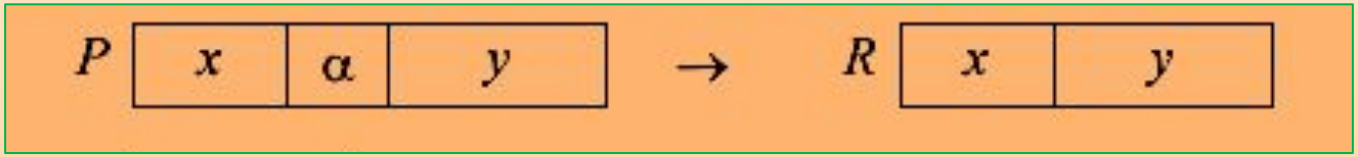
Уточним:

- 1. Если левая часть формулы подстановки входит в слово P, то говорят, что эта формула **применима к P.**
- Если α не входит в P, то формула считается **неприменимой** к P, и подстановка **не выполняется.**

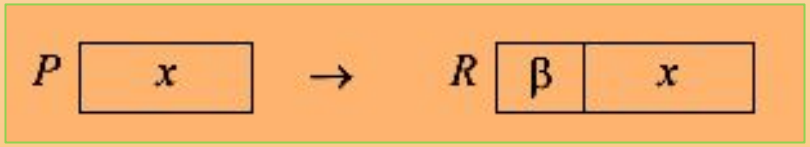
- 2.Если левая часть α входит в P несколько раз, то на правую часть β, по определению, заменяется только первое (самое левое) вхождение α в P:



- 3. Если правая часть формулы подстановки – пустое слово (в таком слове нет ни одного символа), то подстановка $\alpha \rightarrow$ сводится к вычеркиванию части α из P
- * (отметим ! - в формулах подстановки не принято как-либо обозначать пустое слово):



- 4. Если в левой части формулы подстановки указано пустое слово, то подстановка $\rightarrow \beta$ сводится, по определению, к приписыванию β слева к слову P:



- Правило!: формула с пустой левой частью применима к любому слову. (Пустое слово всегда присутствует перед любым словом).
- *Формула с пустыми левой и правой частями не меняет слово.*

Нормальным алгоритмом Маркова (НАМ) называется непустой конечный упорядоченный набор формул подстановки:

$$\begin{cases} \alpha_1 \to \beta_1 \\ \alpha_2 \to \beta_2 \\ \quad \ldots \qquad (k \geq 1) \\ \alpha_k \to \beta_k \end{cases}$$

В этих формулах могут использоваться два вида стрелок: обычная стрелка ($\to$) и стрелка «с хвостиком» ($\mapsto$). Формула с обычной стрелкой называется ***обычной формулой***, а формула со стрелкой «с хвостиком» – ***заключительной формулой***. Разница между ними объясняется чуть ниже.

Записать алгоритм в виде НАМ – значит предъявить такой набор формул.

Правило выполнения нормального алгоритма Маркова

- 1.На каждом шаге входящие в НАМ формулы подстановки просматриваются сверху вниз и выбирается первая из формул, применимых к входному слову Р, т.е. самая верхняя из тех, левая часть которых входит в Р.

- 2. Далее выполняется подстановка согласно найденной формуле. Получается новое слово Р′.

- 3.На следующем шаге это слово Р′ берется за исходное и к нему применяется та же самая процедура, (1,2) и получается новое слово Р′′.
- **И далее: Р → Р′ → Р′′ → …**

- !!! на каждом шаге формулы в НАМ всегда просматриваются начиная с самой первой.

- Необходимые уточнения:
- 1. Если на очередном шаге была применена обычная формула ($\alpha \to \beta$), то работа НАМ продолжается (переход к следующему шагу).

- 2. Если же на очередном шаге была применена заключительная формула ($\alpha \mapsto \beta$), то после её применения работа НАМ прекращается.
- **Слово, которое получилось в этот момент, и есть выходное слово, т.е. результат применения НАМ к входному слову.**

- 3. Если на очередном шаге к текущему слову неприменима ни одна формула, то и в этом случае работа НАМ прекращается, а выходным словом считается текущее слово.

Применимости НАМ:

- В обоих случаях говорят, что НАМ применúм к входному слову.
- Если НАМ никогда не остановится-тогда говорят, что НАМ неприменим к входному слову.

Вставка и удаление символов в НАМ

- 1.Вставка новых символов в слово – это замена некоторого подслова на подслово с бóльшим числом символов;
- в НАМ слово раздвигается автоматически.

- 2. Удаление же символов – это замена некоторого подслова на подслово с меньшим числом символов.
- В НАМ слово сжимается автоматически

например, удаление символа с – это с →

- Пример 1
- A={a,b,c,d}. В слове P требуется удалить все вхождения символа c, а затем заменить первое вхождение подслова bb на ddd. Например: abbcabbca → adddabba
- Решит ли алгоритм 1. нашу задачу?

$$\begin{cases} c \to & (1) \\ bb \to ddd & (2) \end{cases}$$

$$\text{abbcabbca} \xrightarrow{1} \text{abbabbca} \xrightarrow{1} \text{abbabba} \xrightarrow{2} \text{adddabba} \xrightarrow{2} \text{adddaddda} \to$$

Правильное решение:

$$\begin{cases} c \to & (1) \\ bb \mapsto ddd & (2) \end{cases}$$

Вот теперь наш алгоритм будет работать правильно:

$$\text{abbcabbca} \xrightarrow{1} \text{abbabbca} \xrightarrow{1} \text{abbabba} \xrightarrow{2} \text{adddabba}$$

Пример 2 (перестановка символов)

$A=\{a,b\}$. Преобразовать слово P так, чтобы в его начале оказались все символы a, а в конце – все символы b.

Например: babba → aabbb

Решение

$$\{ ba \to ab$$

babba → abbba → abbab → ababb → aabbb

- **Тезис Маркова** : всякий алгоритм в алфавите А представим в виде нормального алгоритма в этом же алфавите.
- Это тезис потому, что его невозможно доказать, т.к. в нем фигурируют с одной стороны, интуитивное расплывчатое понятие "всякий алгоритм", а с другой стороны - точное понятие "нормальный алгоритм".
-

Доп. Теорема 2 Маркова

- Существует такой универсальный алгоритм U, который для любого N и любого входного слова P из области определения алгоритма N

переводит **слово NuPu** , полученное приписыванием изображения слова P(Pu) к изображению алгоритма N(Nu), **в слово Ru**, являющегося изображением соответствующего выходного слова R=N(P), в которое алгоритм N перерабатывает слово P.

- Если же слово P выбирается так, что алгоритм N к нему не применим, то и универсальный алгоритм U не применим к слову NuPu .

- *Данная теорема имеет большое значение, так как из нее вытекает возможность построения машины, которая может выполнять работу любого нормального алгоритма, а значит в силу принципа нормализации – работу любого произвольного алгоритма.*

Глава 4. Типы и структуры данных.
П.1. формальные языки и языки программирования

Опр 1. Формальным языком в программировании называется множество слов, состоящих из слов конечного алфавита.

- Формальный язык может быть определён по-разному, например:

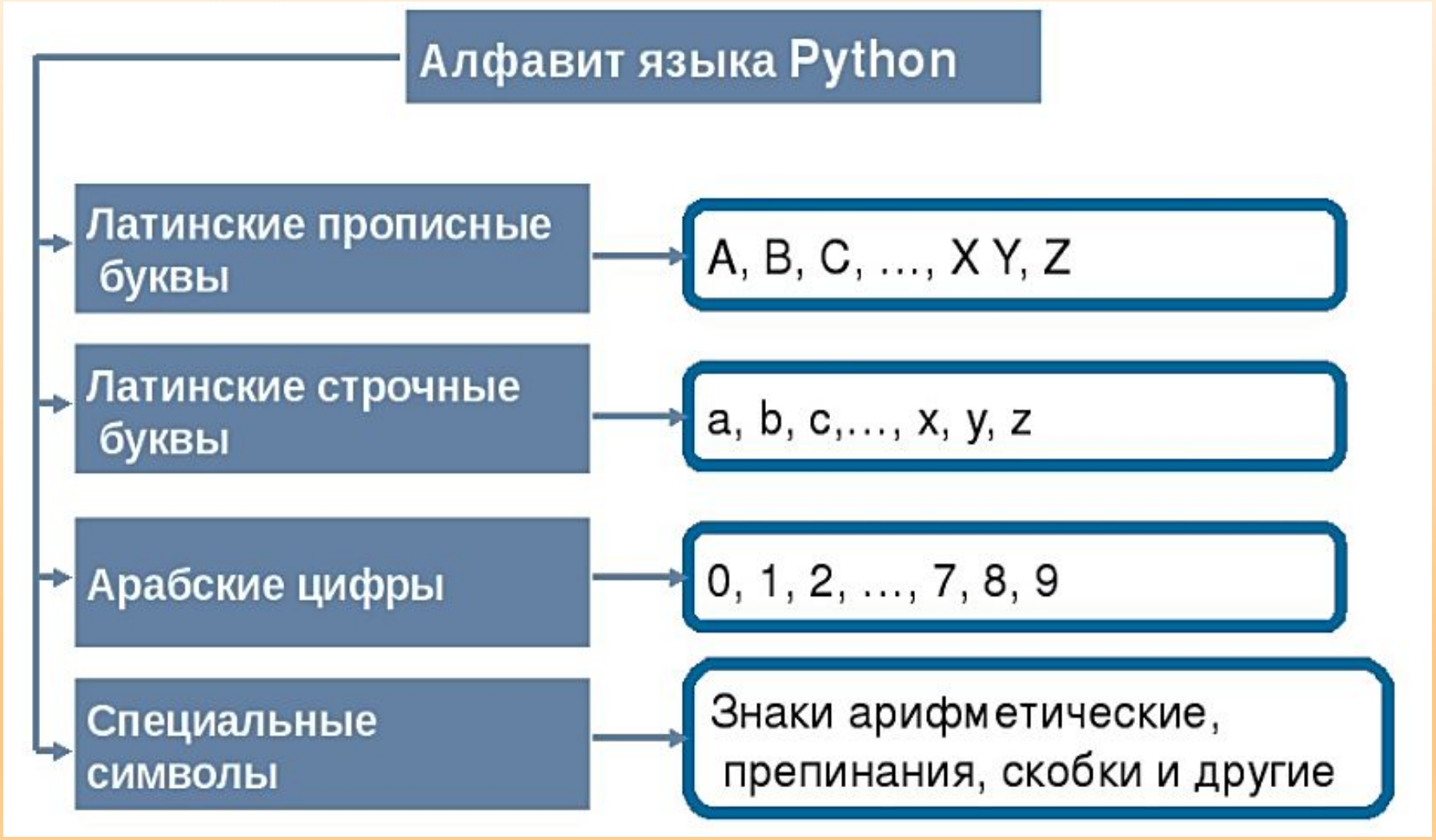
1. Простым перечислением слов, входящих в данный язык. Этот способ, в основном, применим для определения конечных языков и языков простой структуры.
2. Словами, порождёнными некоторой формальной грамматикой.
3. Словами, порождёнными нотациями Бэкуса-Наура.
4. Словами, порождёнными регулярным выражением.
5. Словами, распознаваемыми некоторым конечным автоматом.

Языки программирования.

Для описания языка программирования нужно определить:

1. набор лексических,
2. синтаксических,
3. грамматических правил,

определяющих внешний вид программы и действия, которые выполнит исполнитель (обычно — компьютер) под её управлением.


Алфавит языка Python
Латинские прописные буквы
A, B, C, …, X Y, Z
Латинские строчные буквы
a, b, c,…, x, y, z
Арабские цифры
0, 1, 2, …, 7, 8, 9
Специальные символы
Знаки арифметические, препинания, скобки и другие


Алфавит языка Паскаль
Латинские прописные буквы
A, B, C, …, X Y, Z
Латинские строчные буквы
a, b, c,…, x, y, z
Арабские цифры
0, 1, 2, …, 7, 8, 9
Специальные символы
Знаки арифметические, препинания, скобки и другие

Таблица ASCII:

символ ⟶ ASCII код ⟶ двоичный код

Таблица ASCII -название таблицы кодировки, в которой распространённым печатным и непечатным символам сопоставлены числовые коды.

Таблица была разработана и стандартизирована в США в 1963 году.

- Таблица ASCII определяет коды для символов:

1. Десятичных цифр
2. Латинского алфавита
3. Национального алфавита
4. Знаков препинания
5. Управляющих символов

Коды ASCII

Таблица 1.1. Базовая таблица кодировки ASCII

32	пробел	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	

Таблица Unicode

— стандарт кодирования символов, включающий в себя знаки почти всех письменных языков мира. 1991г.

В настоящее время стандарт является преобладающим в Интернете.

Стандарт состоит из двух основных частей:

1. универсального набора символов
2. семейства кодировок

Unicode

- Для кириллицы в UNICODE отведен диапазон кодов от 0x0400 до 0x04FF.

0x0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
400		Ё	Ђ	Ѓ	Є	S	I	Ї	J	Љ	Њ	Ћ	Ќ		Ў	Џ
410	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
420	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
430	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
440	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
450		ё	ђ	ѓ	є	s	і	ї	j	љ	њ	ћ	ќ		ў	џ
490	Ґ	ґ	Ғ	ғ			Җ	ж			Қ	қ				
4A0			Ң	ң											Ү	ү
4B0	Ұ	ұ	Ҳ	ҳ							Һ	һ				
4D0									Ә	ә						
4E0									Ө	ө						

Цикл работы команды программы

1. **Извлечение инструкции из памяти:** используя текущее значение счетчика команд, процессор извлекает некоторое количество байт из памяти и помещает их в буфер команд.
2. **Декодирование команды: п**роцессор просматривает содержимое буфера команд и определяет код операции и ее операнды. Длина декодированной команды прибавляется к текущему значению счетчика команд.
3. **Загрузка операндов: и**звлекаются значения операндов

Если операнд размещен в ячейках памяти – вычисляется исполнительный адрес.

4. **Выполнение операции над данными.**
5. **Запись результата: р**езультат может быть записан в том числе и в счетчик команд для изменения естественного порядка выполнения.

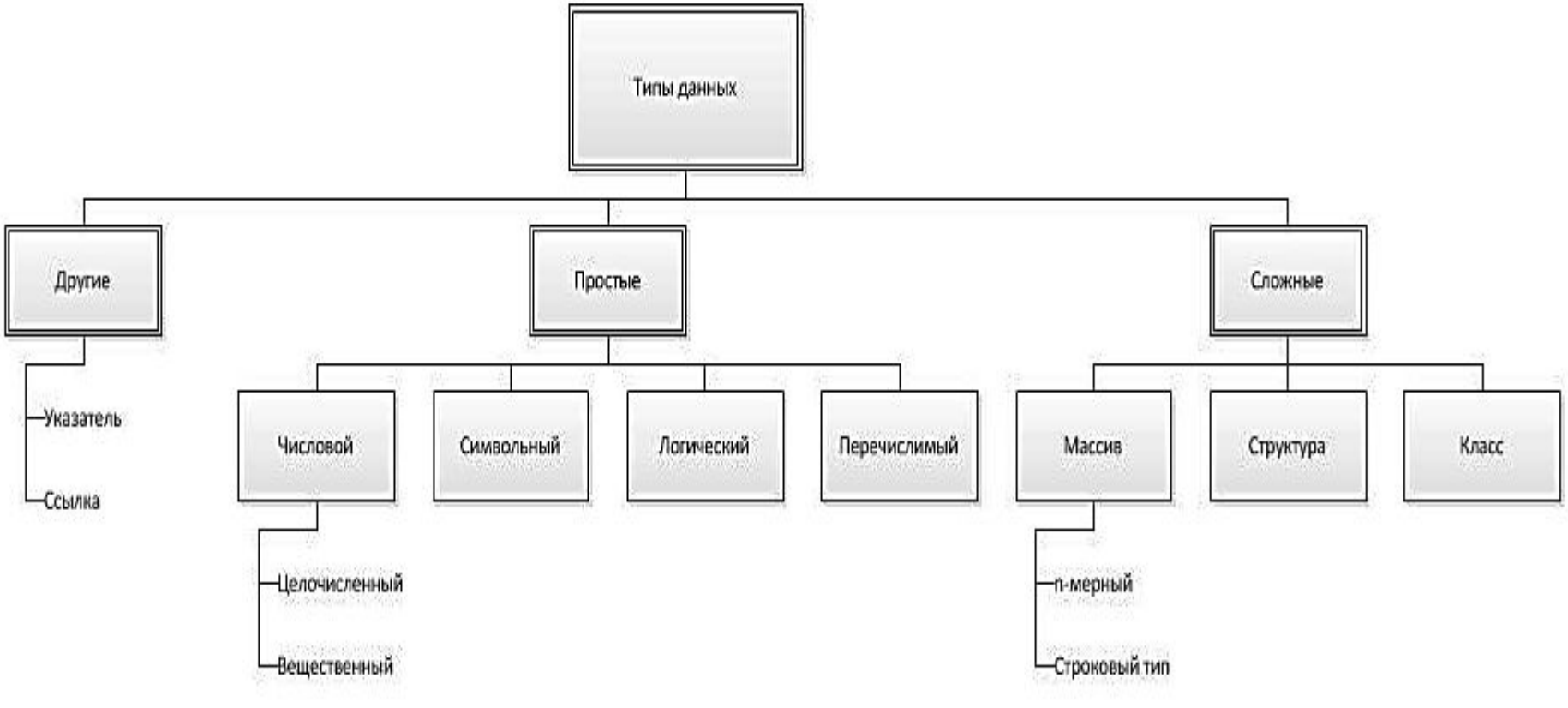
П.2. Типы данных.

Опр. 1. Типом данных называется класс данных, характеризуемый членами класса и операциями, которые могут быть к ним применены.

Тип данных определяет:

1. Набор возможных операций с переменными данного типа
2. Объем памяти для хранения данных данного типа

Классификация типов данных



1). Простые типы данных.

Существует 8 примитивных (простых) типов данных.

область значения	обозначение	требуемый обьем памяти, байт	диапазон	структура в битах	константы
целые числа	char	2	$[0, 2^{16}]$	-	'a', '\u00ff'
	byte	1	[-128, 127]	1-й бит указывает знак	
	short	2	$[-2^{15}, 2^{15}-1]$		
	int	4	$[-2^{31}, 2^{31}-1]$		
	long	8	$[-2^{63}, 2^{63}-1]$		3L, 4l
вещественные числа	float	4	$\sim[-3.4 \cdot 10^{38}; 3.4 \cdot 10^{38}]$ Infinity, NAN	знак: 1 порядок: 8 мантисса: 23	3.4F, .4f
	double	8	$\sim[-1.79 \cdot 10^{308}; 1.79 \cdot 10^{308}]$ Infinity, NAN	знак: 1 порядок: 11 мантисса: 52	3.4, 0.4d, 3.4D
логические константы	boolean	1 бит	{false, true}	-	true, false

2). Составные типы данных

1. **Массив —** это набор значений одного типа, хранящихся последовательно в памяти.
2. **Структуры** в ЯП позволяют хранить несколько полей и в одной переменной. В других языках могут называться записями или кортежами. Например, данная структура хранит в себе ФИО и диагноз**.**
3. **Типы указателей**
4. **Объединения —** это специальные структуры, которые позволяют различным полям разделять общую память. Таким образом в объединении может храниться только одно из полей. Размер объединения равен размеру наибольшего поля.
5. **Перечисления –** позволяющие определять в коде пользовательские типы.
6. **Указатели на функции** позволяют передавать одни функции в другие и реализуют механизм обратного вызова
7. **И др.**

Так в Питоне есть следующие типы данных

1. Числа
2. Логические
3. Строки
4. Байты
5. Массивы байтов
6. Списки
7. Кортежи
8. Множества
9. Словари

П.3.Числовой тип данных

ОПЕРАЦИИ с переменными и константами числового типа:

1. стандартные арифметические операции,
2. операции сравнения,
3. ввод,
4. вывод,
5. присваивание итд

Спецификация типа	Описание	Размер, байт	Допустимый диапазон чисел
char	Однобайтовое целое со знаком	1	-128 ÷ +127
unsigned char	Однобайтовое целое без знака или символ из набора символов системы ASCII	1	0 до 255
int	Целое значение естественного размера (машинное слово) со знаком	2	-32768 ÷ +32767
short int	Целое значение естественного размера (машинное слово) со знаком	2	-32768 ÷ +32767
unsigned int	Целое значение естественного размера без знака	2	от 0 до 65535
long int	Целое значение двойного естественного размера (два машинных слова) со знаком	4	-2147483648 ÷ +2147483647
float	Число в формате с плавающей запятой	4	±1,176E-38 ÷ ±3,40E+38
double	Не рекомендуется для использования	8	±1,7E-308 ÷ ±1,7E+308

П.4. Символьный тип данных

Операции:

1. Ввод символьных значений
2. Вывод
3. Присваивание
4. Все операции сравнения (по ASCII кодам) : =, <=,>=,<,>,<>.

•Примеры стандартных символьных функций:

- возвращает в программу символ с кодом N,
- возвращает код символа S,
- возвращает предыдущий символ
- возвращает следующий символ

Символьный тип данных

- Описание: ***Char***;
- Диапазон значений: ***любой символ*** – это буквы, цифры, знаки препинания и специальные символы.

  Каждому символу соответствует индивидуальный числовой код от 0 до $255_{(10)}$;

П.5. Логический тип данных.Boolean. Основные понятия алгебры логики.

Опр. 2. Логический тип данных - булевский тип, это тип данных, принимающий два возможных значения: **истина (true) и ложь (false).**

- **Опр.3. Логическая переменная** – это переменная, которая может принимать только два значения - TRUE(1) , FALSE(0)

Доступные операции с этим типом данных:

1. ***И* (логическое умножение) (AND, &, *),**
2. ***ИЛИ* (логическое сложение) (OR, | ,+),**
3. ***исключающее ИЛИ* (сложение с переносом) (xor, ^),**
4. **Равенство (*эквивалентность*) (EQV, =)**
5. ***НЕ - инверсия* (NOT, !)**
6. ***Сравнение* (>, <, <=, >=)**

Основные понятия Алгебры логики.

Опр. 4. Логическое высказывание — это любое повествовательное предложение, в отношении которого можно однозначно сказать, истинно оно или ложно.

- **Чтобы обращаться к логическим высказываниям, им назначают имена.**

- Пример 1. Пусть через **A** обозначено высказывание «Ольга поедет летом на море", а через **B** — высказывание «Ольга летом отправится в горы".

Тогда составное высказывание «Ольга летом побывает и на море, и в горах" можно кратко записать как : **A и B**.

- **Задание 1.** Пусть
- a = “это утро ясное”,
- b = “это утро теплое”.

  Выразите следующие формулы на обычном языке:

а)	$a \cdot \bar{b}$	г)	$\bar{a} \vee b$	ж)	$\overline{a \cdot b}$	к)	$a \to \bar{b}$
б)	$\overline{a \cdot b}$	д)	$\overline{a \vee b}$	з)	$\overline{a} \vee b$	л)	$\overline{a} \to b$
в)	$\overline{a} \cdot \overline{b}$	е)	$\overline{a} \vee \overline{b}$	и)	$\overline{a \cdot b}$	м)	$\overline{a \to b}$

Основные логические операции

1. *Логическое отрицание* (*инверсия*).
   Обозначается: $\bar{A}$, $\neg A$, **not** A, **не** A.
   Высказывание $\neg A$ истинно при ложном A и ложно при истинном A.

2. *Логическое умножение* (*конъюнкция*).
   Обозначается $A\&B$, A **and** B, $A \cdot B$, $A \wedge B$, AB, A **и** B.
   Высказывание $A \wedge B$ истинно тогда и только тогда, когда оба высказывания A и B истинны.

3. *Логическое сложение* (*дизъюнкция*).
   Обозначается: $A \vee B$, A **or** B, $A + B$, A **или** B.
   Высказывание $A \vee B$ ложно тогда и только тогда, когда оба высказывания A и B ложны.

Таблицы истинности- значение функции при всех наборах переменных

Отрицание

A	$\neg A$
0	1
1	0

Коньюнкция

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

Дизьюнкция

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

Дополнительные логические операции

4. *Логическое следование* (*импликация*).
   Обозначается: $A \to B$, $A \Rightarrow B$.
   Высказывание $A \to B$ ложно только тогда, когда A истинно, а B ложно.
   Важно: в операции импликации посылка A не обязана быть истинной, в отличие от логического оператора в языках программирования «если A то B».
   Импликация выражается через дизъюнкцию и отрицание:
   $A \Rightarrow B = \neg A \vee B$.

5. *Эквивалентность* (равносильность, необходимость и достаточность).
   Обозначается: $A \sim B$, $A \Leftrightarrow B$, $A \equiv B$.
   Высказывание $A \Leftrightarrow B$ истинно тогда и только тогда, когда значения A и B совпадают.
   Эквивалентность выражается через отрицание, дизъюнкцию и конъюнкцию:
   $A \Leftrightarrow B = (\neg A \vee B) \wedge (\neg B \vee A)$.

6. *Исключающее ИЛИ.*
   Обозначается: A **XOR** B, $A \oplus B$
   Высказывание A **XOR** B истинно, когда A и B не равны.

Таблицы истинности доп. лог. операций

Эквивалентность

A	B	$A \Leftrightarrow B$
0	0	1
0	1	0
1	0	0
1	1	1

Следование

A	B	$A \Rightarrow B$
0	0	1
0	1	1
1	0	0
1	1	1

Искл. ИЛИ - XOR

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Законы алгебры логики

Закон	Для ИЛИ	Для И
Коммутативный (переместительный): логические переменные можно менять местами	$X \vee Y = Y \vee X$	$X \wedge Y = Y \wedge X$
Ассоциативный (сочетательный): логические переменные в дизъюнкциях и конъюнкциях можно объединять в группы	$(X \vee Y) \vee Z = X \vee (Y \vee Z)$	$(X \wedge Y) \wedge Z = X \wedge (Y \wedge Z)$
Дистрибутивный (распределительный): одинаковые переменные в дизъюнкциях и конъюнкциях можно выносить за скобки	$(X \vee Y) \wedge Z =$ $= (X \wedge Z) \vee (Y \wedge Z)$	$(X \wedge Y) \vee Z =$ $= (X \vee Z) \wedge (Y \vee Z)$

Закон	Для ИЛИ	Для И
Закон непротиворечия: высказывание может быть только истинным или ложным, третьего не дано		$X \wedge \neg X = 0$
Закон исключенного третьего: высказывание может быть только истинным или ложным, третьего не дано	$X \vee \neg X = 1$	
Правила де Моргана	$\neg (X \vee Y) = \neg X \wedge \neg Y$	$\neg (X \wedge Y) = \neg X \vee \neg Y$
Законы склеивания	$(X \wedge Y) \vee (\neg X \wedge Y) = Y$	$(X \vee Y) \wedge (\neg X \vee Y) = Y$
Исключение констант	$X \vee 0 = X,\ X \vee 1 = 1$	$X \wedge 0 = 0,\ X \wedge 1 = X$

Двойного отрицания: двойное отрицание исключает отрицание	$\neg\neg X = X$	
Идемпотентности	$X \vee X = X$	$X \wedge X = X$
Контрапозиции	$A \to B = \neg B \to \neg A$	
Снятие импликации	$X \to Y = \neg X \vee Y$	
Снятие эквивалентности	$A \leftrightarrow B = (A \wedge B) \vee (\neg A \wedge \neg B)$ $A \leftrightarrow B = (A \vee \neg B) \wedge (\neg A \vee B)$	
Закон поглощения	$A \vee (A \wedge C) = A$	$A \wedge (A \vee C) = A$

Последовательность выполнения операций:

1. Скобки
2. логическое отрицание
3. логическое умножение
4. логическое сложение
5. исключающее ИЛИ
6. логическое следование
7. эквивалентность

Семинар 9.

Алгоритм построения таблицы истинности логической функции(выражения)

1) определить количество N используемых переменных в логическом выражении;
2) вычислить количество всевозможных наборов значений переменных $M = 2^N$, равное количеству строк в таблице истинности;
3) подсчитать количество логических операций в логическом выражении и определить число столбцов в таблице, которое равно сумме числа переменных и количества логических операций;

4) озаглавить столбцы таблицы названиями переменных и названиями логических операций;
5) заполнить столбцы логических переменных наборами значений, например, от 0000 до 1111 с шагом 0001 в случае для четырех переменных;
6) заполнить таблицу истинности по столбцам со значениями промежуточных операций слева направо;
7) заполнить окончательный столбец значений для функции *F*.

Пример 1.таблица истинности для выражения

$$(A \vee B) \rightarrow \neg C$$

A	B	C	(A V B)	¬C	(A V B) → ¬C
0	0	0	0	1	1
0	0	1	0	0	1
0	1	0	1	1	1
0	1	1	1	0	0
1	0	0	1	1	1
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	0	0

Задачи 2.Самост. реш

- 1) Построить таблицу истинности выражения: $x \vee \neg y$
- 2) Построить таблицу истинности выражения: $x \vee y \wedge z$

Пример 2. Составим таблицу истинности для формулы $x \cdot y \vee \overline{x \vee y} \vee x$

Перемен-ные		Промежуточные логические формулы					Формула
x	y	$\overline{x}$	$\overline{x \cdot y}$	$x \vee y$	$\overline{x \vee y}$	$x \cdot y \vee \overline{x \vee y}$	$x \cdot y \vee \overline{x \vee y} \vee x$
0	0	1	0	0	1	1	1
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	1
1	1	0	0	1	0	0	1

Оператор IF и логические выражения

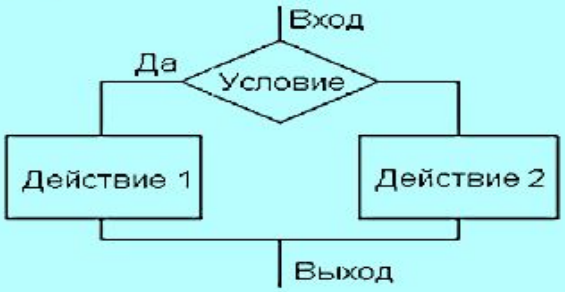
Условный оператор if…then…else полная форма

Синтаксис:

if <логическое выражение>
then <оператор1>
else <оператор2>;

Семантика:

1. Вычисляется значение **<логического выражения>**
2. Если **<логическое выражение>** истинно (TRUE), то выполняется оператор1, иначе выполняется оператор2.



Пример:
If2. Дано целое число N.
Если оно положительное, то прибавить к нему 1; если отрицательно вычесть из него 2.
Вывести полученное число.

```
program if_2;
 var  n: integer;
 begin
 write ('введите целое число n=');
 readln  (n);
 if n>0
   then n:=n+1
   else n:=n-2;
 writeln ('n=',n);
 end.
```

Использование лог выражений в языках программирования:

1). Присвоить A=TRUE; B=False

2). По таблицам истинности рассчитать, что будет выведено на экран:

Пример 1.

IF $A \wedge B \wedge C$ THEN “Верно”
ELSE “Неверно”

При каких значениях A B C будет напечатано **ВЕРНО?**

- **Пример 2.**

  IF A ∨ *B* ∨ *C* THEN “Верно”

  ELSE “Неверно”

При каких значениях A B C будет напечатано **НЕВЕРНО?**

Проверочная работа по теме: Алгебра логики 1.

- Построить таблицы истинности
- **Задача 1.** $\neg x \wedge y$
- **Задача 2.** $x \vee \neg y \wedge z$
- **Задача 3.** $x \vee \neg y \oplus y$

Задача 4. Написать псевдокод решения задачи с использованием конструкций IF-THEN-ELSE и логических операторов AND, OR, NOT и комментариев:

1.Вводятся через пробел значения массы тела и роста одного пациента.

2.Рассчитайте индекс массы тела (ИМТ) по формуле:

ИМТ = вес (в кг) / рост (в м)^2

Определите, является ли ИМТ пациента нормальным или пациент имеет дефицит/избыток массы тела, выведите на экран данное заключение.

1. Если ИМТ 18,5-24,99, то **ИМТ- Норма**
2. Если ИМТ вне диапазона вывести **Нарушение ИМТ**

Задание 5. Построчно просчитать решение по псевдокоду для значений (170 65) и (180 120), Написать, что будет напечатано на экране

Семинар 10.

Задание 2.

Написать псевдокод решения задачи с
Использованием конструкций IF-THEN-ELSE и логических операторов AND, OR, NOT и комментариев:
Вводятся через пробел значения роста и массы тела пациента.

Рассчитайте индекс массы тела (ИМТ) по формуле: ИМТ = вес (в кг) / рост (в м)^2
Определите, является ли ИМТ пациента нормальным или пациент имеет дефицит/избыток массы тела, выведите на экран данное заключение.

Если:

1. ИМТ<18,49, то напечатать **Недостаточная масса тела**
2. ИМТ 18,5-24,99, то **Норма**
3. ИМТ>25,00, то **Избыточный вес**

Построчно просчитать решение по псевдокоду для значений (160 60) и (170 110) и (170 50)
Вывести , что будет напечатано на экране

Задание 3.

Написать псевдокод решения задачи с использованием конструкций IF-THEN-ELSE и логических операторов AND, OR, NOT и комментариев:

Вводятся через пробел значения массы тела и роста пациента и код пациента. Рассчитайте индекс массы тела (ИМТ) по формуле: ИМТ = вес (в кг) / рост (в м)^2

Определите, является ли ИМТ пациента нормальным или пациент имеет дефицит/избыток массы тела, выведите на экран данное заключение.

Если:

ИМТ<18,49 или код равен 0, то напечатать Недостаточная масса тела

ИМТ 18,5-24,99 и код пациента равен 1, то Норма

ИМТ>25,00, то Избыточный вес

Построчно просчитать решение по псевдокоду для значений (160 60 1) и (160 60 0) и (170 50 1) (170 50 1)

Вывести, что будет напечатано на экране в каждом варианте, рассчитать таблицу истинности для данных значений и сравнить программный результат

Задача 4.

Псевдокод к задаче, используя цикл с предусловием и for:

Вводятся данные 7 пациентов: рост, вес и пол. Посчитать количество мужчин-пациентов, у которых ИМТ в норме.

Тип данных – Массивы.

Опр. Массив — это структура однотипных данных, занимающих непрерывную область памяти.

Характеристики массива данных:

1. Массив имеет размер — количество элементов в нем.

2. Каждый элемент массива имеет свой номер (индекс), обращение к элементу массива осуществляется путем указания его индекса.

3. В ЯП элементы нумеруются начиная с 0, поэтому последний элемент массива имеет номер на 1 меньше размера массива;
   если нумерация с 1- то последний элемента
   имеет номер = размеру массива.

Объявление массивов:

Массив в языках пр-я задается следующим образом:

1. тип_элементов идентификатор[размер элемента], где тип_элементов — произвольный тип данных языка пр-я, который будут иметь элементы массива, например, integer, real и т. д.;
2. идентификатор — имя массива;
3. размер — число элементов в массиве.

К элементу массива можно обращаться,
как Имя массива[индекс]
Например, если было сделано объявление массива real A[5];
то таким образом создается 5 элементов массива типа real: A[0],A[1], A[2], A[3], A[4]

Инициализация - заполнение массива:

1. `int leap[]={31,29,31,30,31,30,31,31,30,31,30,31};`

2. `int non_leap[]={31,28,31,30,31,30,31,31,30,31,30,31};`

3. `int N=sizeof(leap)/sizeof(non_leap[0])`

Размерность массивов:

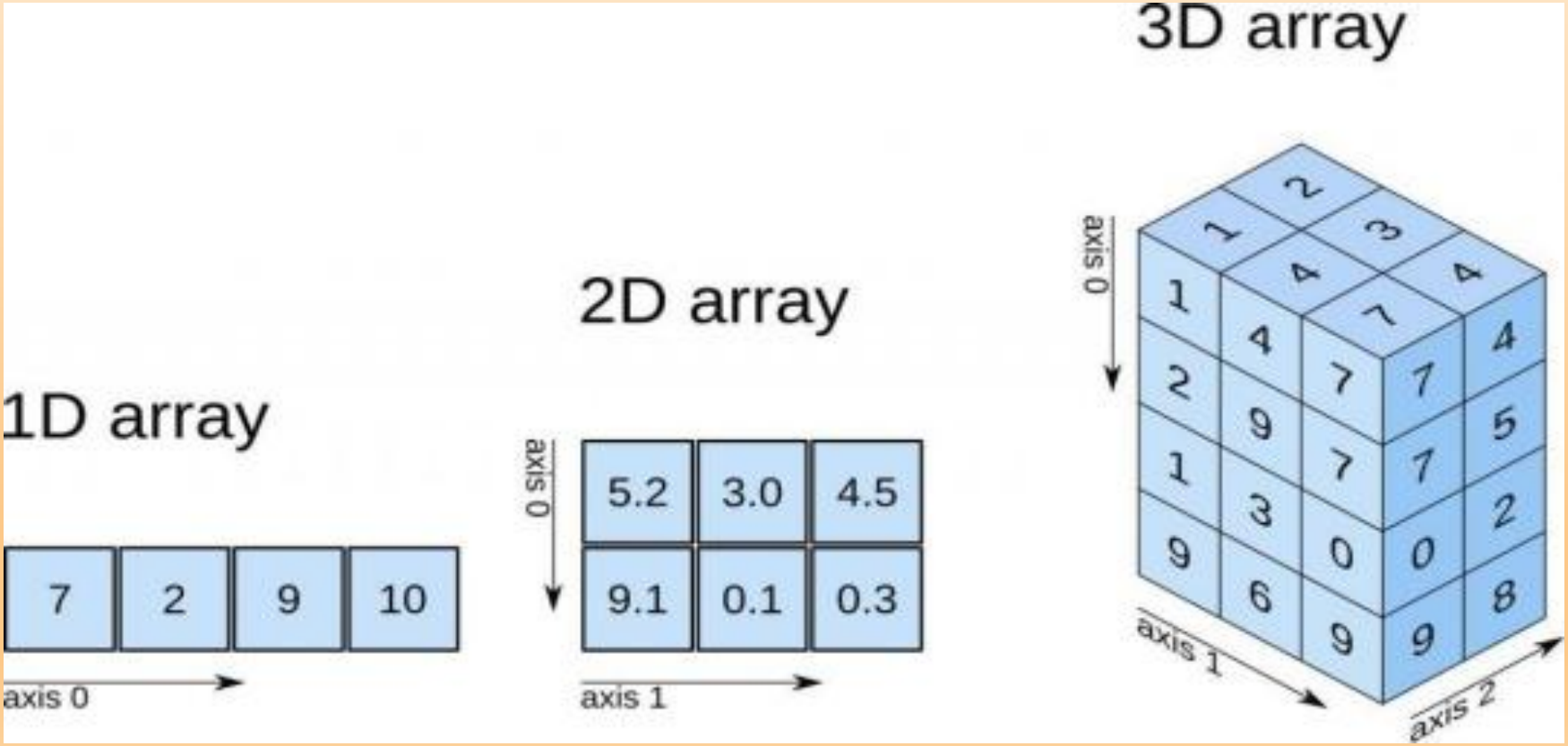
Массивы могут быть одномерными, двумерные (матрицы), трехмерные и т.д.

Размерность массивов может быть ограничена или неограничена в ЯП, например в С++ никак не ограничивается.

Пример 1: объявление 2-мерного массива - вещественный массив (3 строки, 2 столбца): double matrix[3][2];

Обращаются к элементам такого массива, указывая два индекса: matrix[1][2];

Представления 1-, 2-, 3-х мерных массивов



Алгебра матриц

- Матрица – 2-мерный массив данных.
- Действия:

1. Сложение матриц
2. Вычитание матриц
3. Умножение матрицы на число
4. Умножение матрицы на вектор
5. Умножение матриц

Умножение массивов – это НЕ УМНОЖЕНИЕ МАТРИЦ

Линейные операции над матрицами.

Линейные операции – это сложение, вычитание, умножение на число.

- **Сложение и вычитание матриц.**

Складывать и вычитать можно только матрицы одинакового размера.

Чтобы сложить (или вычесть) две матрицы надо сложить (вычесть) попарно их соответствующие элементы.

$$A=\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \cdots & \cdots & \cdots \\ a_{k1} & \cdots & a_{kn} \end{pmatrix} \qquad B=\begin{pmatrix} b_{11} & \cdots & b_{1n} \\ \cdots & \cdots & \cdots \\ b_{k1} & \cdots & b_{kn} \end{pmatrix} \qquad \mathrm{A}\pm\mathrm{B}=\begin{pmatrix} a_{11}\pm b_{11} & \cdots & a_{1n}\pm b_{1n} \\ \cdots & \cdots & \cdots \\ a_{k1}\pm b_{k1} & \cdots & a_{kn}\pm b_{kn} \end{pmatrix}$$

Умножение матрицы на число

Произведением матрицы А на число λ называется матрица B=λA, элементы которой получены умножением:

$$b_{ij} = \lambda \cdot a_{ij}$$

Пример.

$$A = \begin{pmatrix} 2 & 4 \\ 3 & 2 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & 5 & 2 \\ 2 & 1 & 3 \\ 4 & 1 & 7 \end{pmatrix}$$

$$5A = \begin{pmatrix} 10 & 20 \\ 15 & 10 \end{pmatrix} \qquad 2B = \begin{pmatrix} 2 & 10 & 4 \\ 4 & 2 & 6 \\ 8 & 2 & 14 \end{pmatrix}$$

Преобразование матриц

Умножение матрицы на вектор, получаем изменённый вектор:

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} ax + by + cz + dw \\ ex + fy + gz + hw \\ ix + jy + kz + lw \\ mx + ny + oz + pw \end{bmatrix}$$

Умножение матриц

Умножение матрицы на число k

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

$$k \cdot A = \begin{pmatrix} a_{11} \cdot k & a_{12} \cdot k & a_{13} \cdot k \\ a_{21} \cdot k & a_{22} \cdot k & a_{23} \cdot k \\ a_{31} \cdot k & a_{32} \cdot k & a_{33} \cdot k \end{pmatrix}$$

Умножение матриц

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}$$

$$C = A \cdot B = \begin{pmatrix} a_{11} \cdot b_{11} + a_{12} \cdot b_{21} + a_{13} \cdot b_{31} & a_{11} \cdot b_{12} + a_{12} \cdot b_{22} + a_{13} \cdot b_{32} & a_{11} \cdot b_{13} + a_{12} \cdot b_{23} + a_{13} \cdot b_{33} \\ a_{21} \cdot b_{11} + a_{22} \cdot b_{21} + a_{23} \cdot b_{31} & a_{21} \cdot b_{12} + a_{22} \cdot b_{22} + a_{23} \cdot b_{32} & a_{21} \cdot b_{13} + a_{22} \cdot b_{23} + a_{23} \cdot b_{33} \\ a_{31} \cdot b_{11} + a_{32} \cdot b_{21} + a_{33} \cdot b_{31} & a_{31} \cdot b_{12} + a_{32} \cdot b_{22} + a_{33} \cdot b_{32} & a_{31} \cdot b_{13} + a_{32} \cdot b_{23} + a_{33} \cdot b_{33} \end{pmatrix}$$

Работа с массивами:

- Для сложения,
- вычитания,
- умножения,
- деления
- образования формул

массивы должны быть одного размера.

Массивы в Питоне

Python не имеет встроенной поддержки массивов, но вместо этого можно использовать **списки (list) Python**.
Массивы используются для хранения нескольких значений в одной переменной:

```
cars = ["Ford", "Volvo", "BMW"]
```

Двумерные массивы в Питоне.

Многомерные списки в Питон.

2. Операторы и операции (арифметические и матричные)

Операторы:

.* поэлементное умножение массивов одинаковых размеров.
C = A .* B равносильно C(i , j) = A(i , j) * B(i , j)

./ поэлементное деление массивов одинаковых размеров.
C = A ./ B равносильно C(i , j) = A(i , j) / B(i , j)

.\ поэлементное левое деление массивов одинаковых размеров.
C = A .\ B равносильно C(i , j) = B(i , j) / A(i , j)

Задачи ДЗ- алгебра матриц

Задача1. Сложить и – массив B и 2B.

Задача 2. Умножить на число (-3) массив A.

Задача 3. **Поэлементное** умножение массивов B и 2B

Задача 4 – умножить матрицы B и 2B

Проверочная работа.
Задача 4.
Написать псевдокод:

Описать 2-мерный массив A целых чисел 2*8, инициализировать его и сделать расчет суммы S всех элементов.

Задача 5.
Написать псевдокод:

Описать два 2-мерных массива целых чисел A и B размера 2*5; инициализировать их и сделать расчет нового массива C, каждый элемент которого равен c=2a+3b соответствующих элементов массивов A и B.

Задача 6.
Написать псевдокод:

Описать 2-а массива вещественных чисел Аи B размера 3*3.
Инициализировать их.
Рассчитать значения элементов следующих 2-х массивов:
C= произведение матриц A, B;
D= поэлементное произведение матриц A,B.

На дом.

Задача 7. Описать 2-мерный массив **ALA** размера m*n (вводятся). Инициализировать массив.

Найти в этом массиве заданный элемент **x**, распечатать – значение элемента, адрес (индексы) элемента.

Рекурсия в программировании

Опр. 1. В программировании рекурсия — вызов функции (процедуры) из неё же самой, непосредственно (*простая рекурсия*) или через другие функции (*сложная* или *косвенная рекурсия*),

Примеры:
Функция A(x) вызывает функцию A(x-1),
Функция A вызывает функцию B, а функция B — функцию A.

Опр.2. Количество вложенных вызовов функции или процедуры называется глубиной рекурсии.

Преимущества: рекурсивная программа позволяет описать повторяющиеся вычисления без явных повторений частей программы и использования циклов.

Одномерные и двумерные массивы - основные задачи.

1.1.Одномерные массивы:

1. Объявление массива
2. Инициализация, заполнение
3. Сортировка
4. Поиск
5. Замена элементов массива
6. Удаление элементов, сдвиг

- Сортировка массива - преобразование последовательности элементов в неубывающую (или невозрастающую) последовательность.
- Последовательность элементов $\{A_i\}$ называют неубывающей, если для любых i и j, таких что $i < j$ выполняется неравенство $A_i <= A_j$. Для строго возрастающей последовательности неравенство принимает вид $A_i < A_j$. Аналогичным образом определяется невозрастающая и убывающая последовательности.

Элементы массива можно сортировать:

1. по возрастанию — каждый следующий элемент больше предыдущего — A[1] < A[2] <… < A[N];

2. по неубыванию — каждый следующий элемент не меньше предыдущего, то есть больше или равен A[1] ≤ A[2] ≤… ≤A[N];

3. по убыванию — каждый следующий элемент меньше предыдущего A[1] > A[2] > … > A[N];

4. по невозрастанию — каждый следующий элемент не больше предыдущего, то есть меньше или равен A[1] ≥ A[2] ≥ … ≥ A[N].

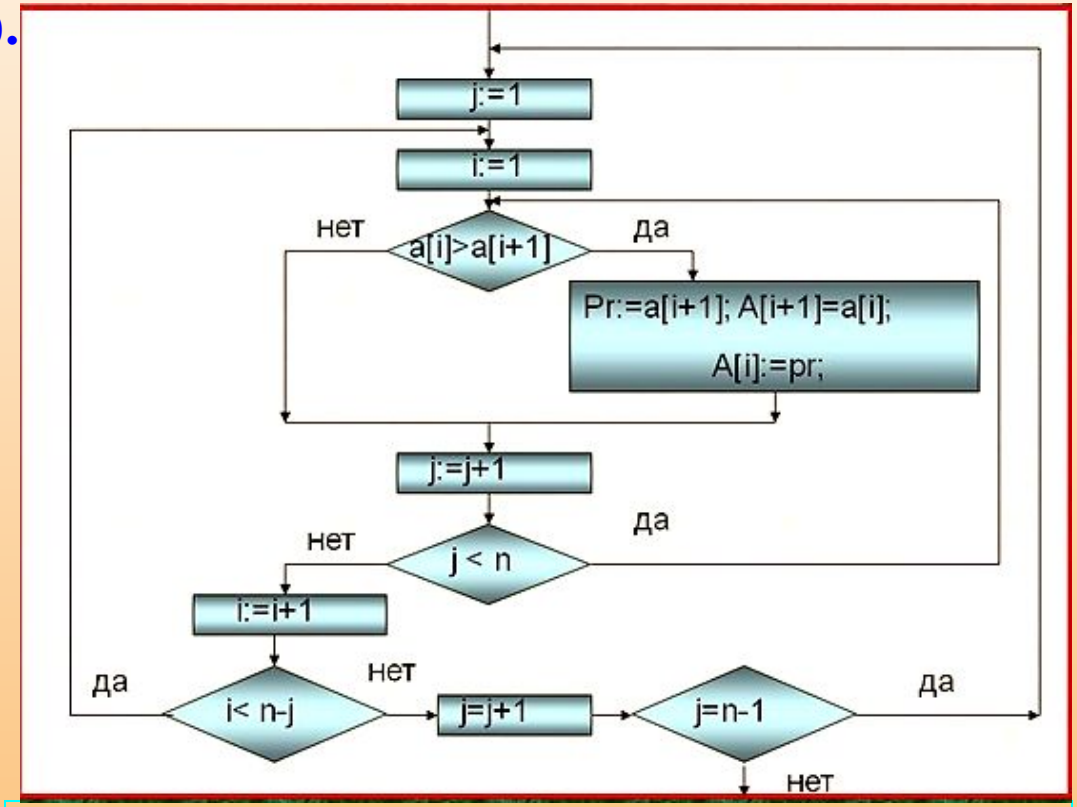
- **Постановка задачи**
- **Пусть требуется упорядочить массив *N* элементов по возрастанию, причем в каждом элементе массива хранятся данные одного типа, каждый элемент массива имеет индекс.**

Методы сортировки массива
Метод сортировки пузырьком.

(Пример - по убыванию).

1. При первой итерации цикла элементы массива попарно сравниваются между собой. Если текущий элемент больше следующего, то производится их обмен местами. Так наименьший элемент становится на последнее место в массиве.
2. При второй итерации попарно сравниваются элементы с первого до предпоследнего….
3. Сортировка заканчивается после сравнения 1 и 2-го элементов.

Временная сложность **O(n²)**.



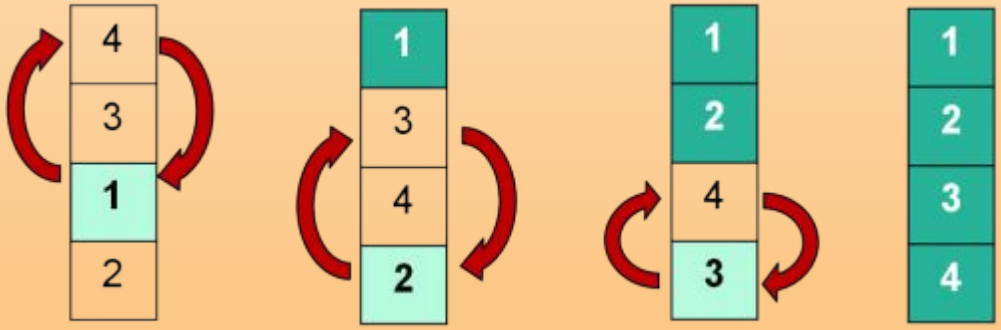
Сортировка методом Пузырька

Сортируем массив Arr:

```
for I := Low(Arr) to High(Arr) - 1 do
for J := I + 1 to High(Arr) do
   if Arr[I] > Arr[J] then
    begin
        T := Arr[I];
        Arr[I] := Arr[J];
        Arr[J] := T;
    end;
```

2.Сортировка методом выбора (по возрастанию)

1. в массиве ищется минимальный элемент и ставится на первое место (меняется местами с A[1]);
2. среди оставшихся элементов также производится поиск минимального, который ставится на второе место (меняется местами с A[2]) и т.д.

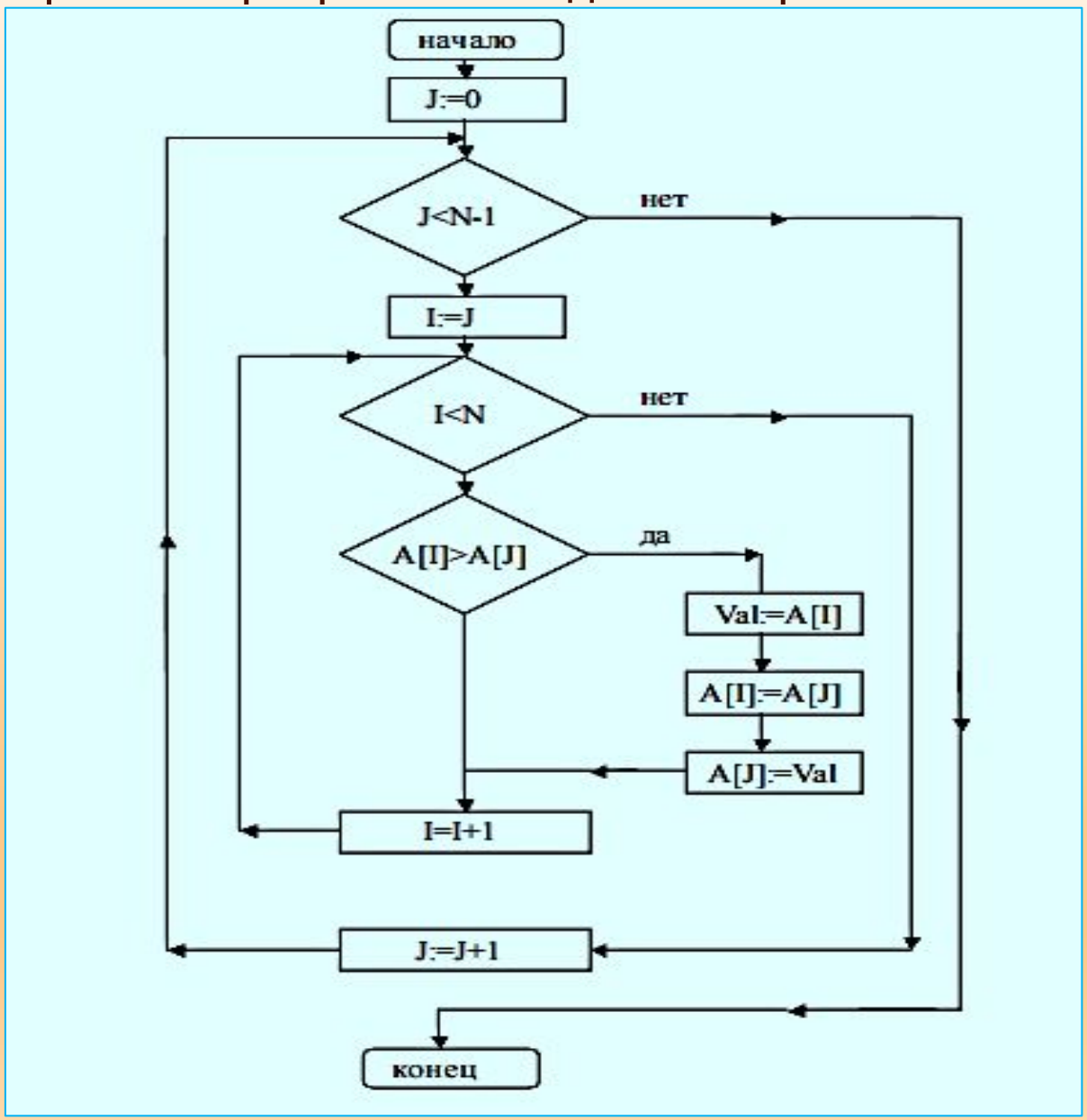


- **Временная сложность.**

Поиск минимума для каждого элемента массива равен $\mathbf{O(n^2)}$.

Начертим блок-схему метода выбора

Блок-схема алгоритма сортировки методом выбора



Задача

1.Написать псевдокод для решения задачи :
Упорядочить по возрастанию массив A из N эл-в методом выбора. N вводится, массив вводится.
Распечатать исходный и упорядоченный массивы.
2.Сделать расчет по псевдокоду
для N=6 входного массива A=[9;8;1;2;4;0]

3. Сортировка вставками

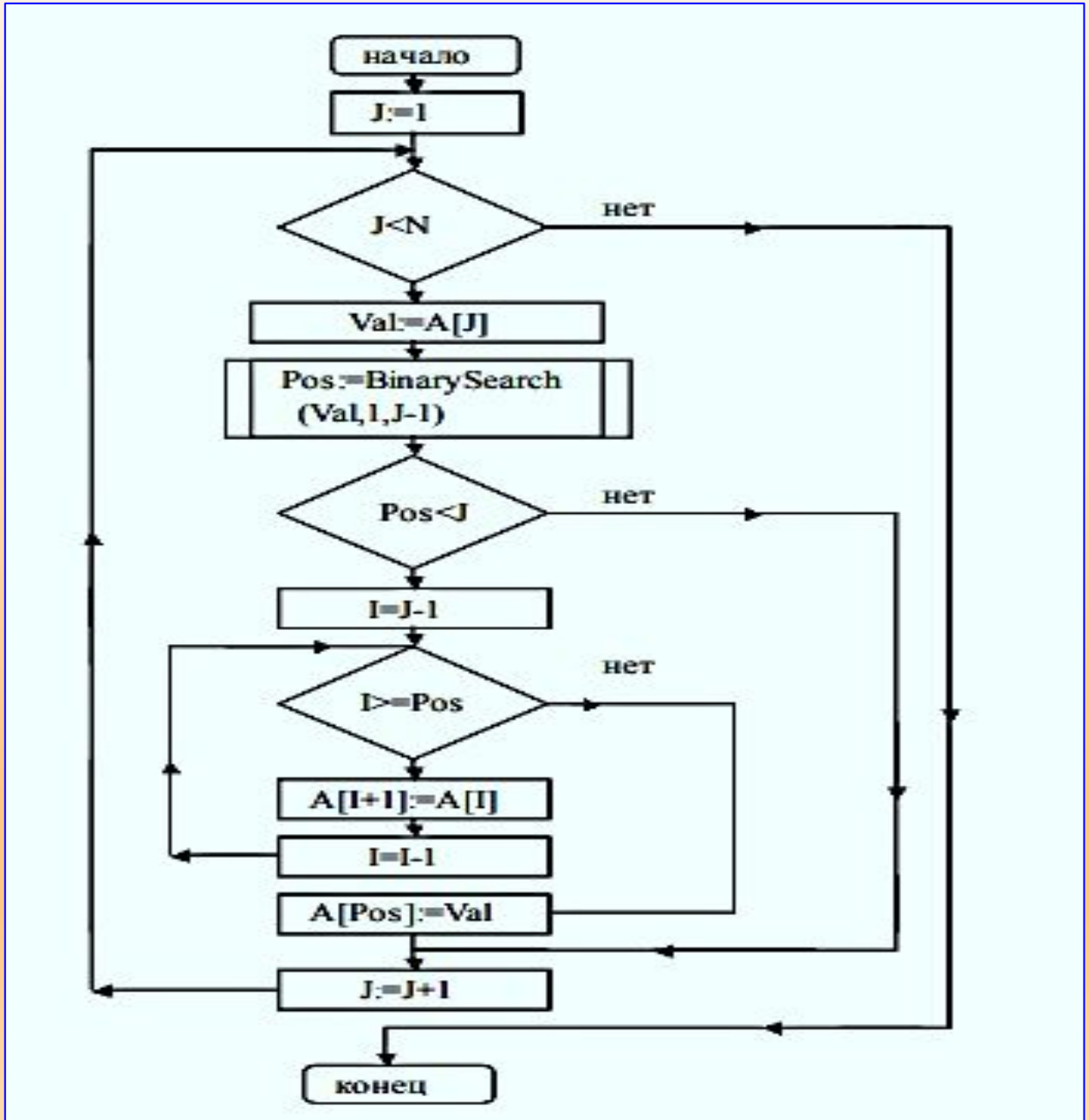
Алгоритм:

- на первом шаге сортировки второй элемент сравнивается с первым, на втором шаге третий элемент сравнивается с двумя первыми и т. д.
- среди уже отсортированных i-1 элементов массива вставляют i-й элемент без нарушения порядка, т. е. при вставке i-го элемента на j-е место (j < i) элементы с индексами >j и <i увеличивают свой номер на единицу.

- **Временная сложность - $O(n^2)$.**

Начертим блок-схему метода сортировки вставками

Блок-схема сортировки вставками



Задача 2.

1.Написать псевдокод для решения задачи :
Упорядочить по возрастанию массив B из M эл-в методом вставок. M вводится, массив вводится.
Распечатать исходный и упорядоченный массивы.
2.Сделать расчет по псевдокоду
для N=4 входного массива B=[6;1;2;3]

- **4.Сортировка Шелла.**
  Усовершенствованный вариант метода сортировки вставками.

Алгоритм:

1. сначала сравниваются и сортируются между собой значения, стоящие один от другого на некотором d.
2. Затем процедура повторяется для некоторых меньших значений d ...
3. завершается сортировка Шелла упорядочиванием элементов при d=1 – сортировкой вставками.

Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места.

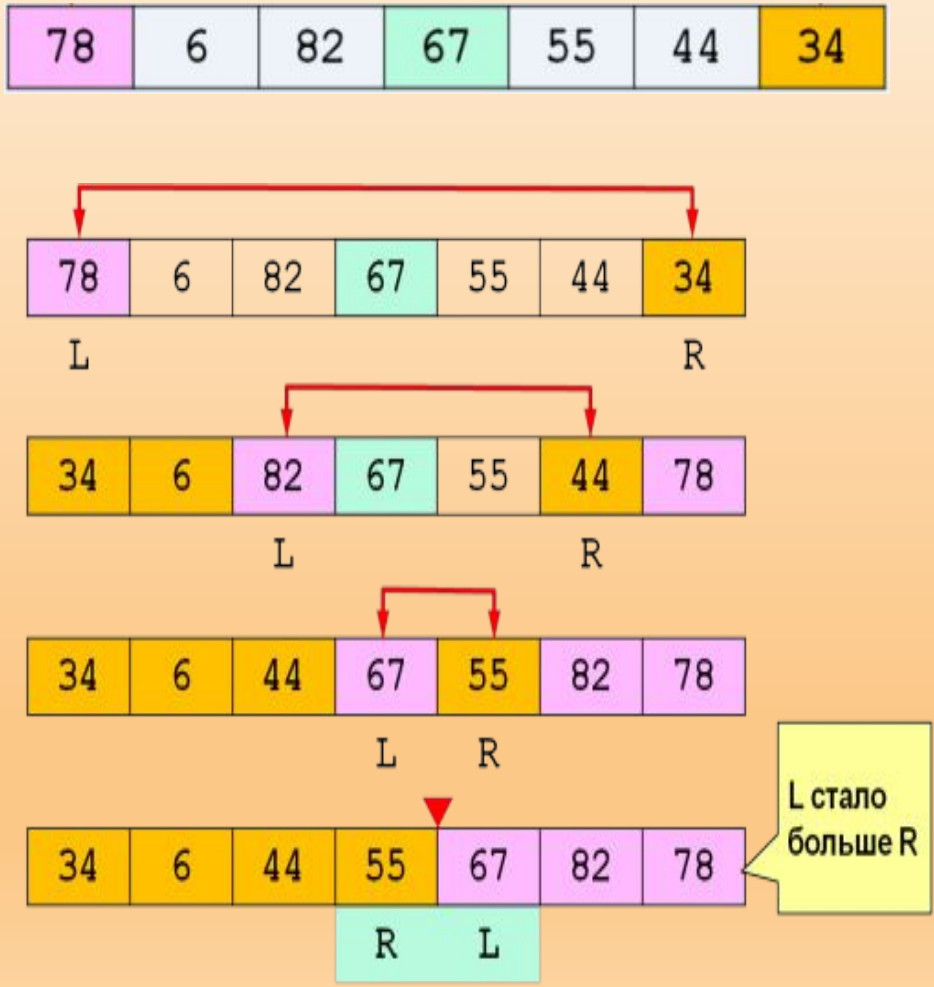
Временная сложность $O(n\log^2 n)$

5.Быстрая сортировка или quick sort:

Алгоритм:

1. Выберем и запомним средний элемент массива (присвоим ***X=67***):
2. Инициализируем две переменные (будущие индексы массива): L:=1, R:=N (N — количество элементов).
3. Увеличиваем L и ищем первый элемент A[L], который больше либо равен X (в итоге он должен находиться справа).
4. Уменьшаем R и ищем элемент A[R], который меньше либо равен X (в итоге он должен находиться слева).
5. Смотрим, если L<=R, то меняем местами A[L] и A[R], возвращаемся к пункту 3.

Временная сложность $O(n\log_2 n)$



6.Сортировка слиянием.

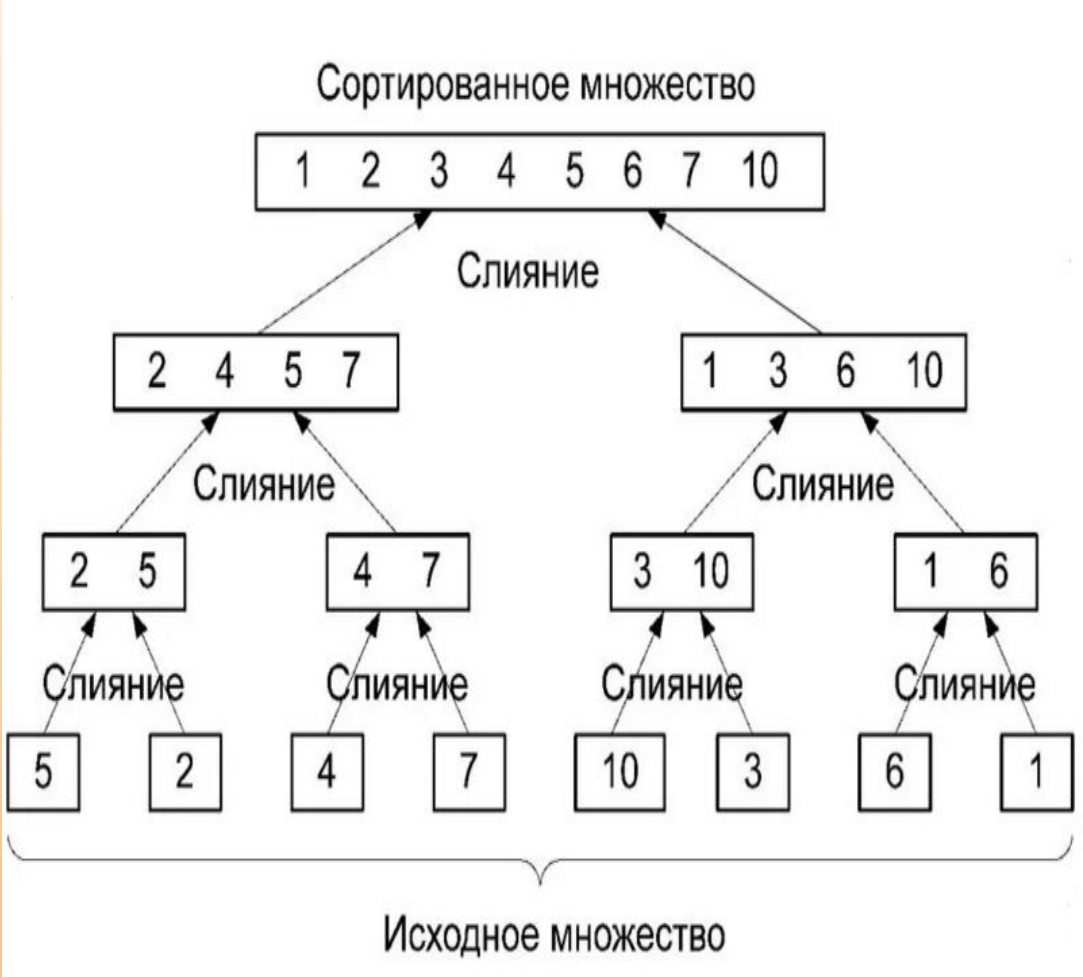
Алгоритм:

1.Массив делится на два подмассива, а затем происходит:

1) Сортировка левой половины массива (рекурсивно)
2) Сортировка правой половины массива (рекурсивно)
3) Слияние, упорядочивание.

Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не достигнет единицы (любой массив длины 1 можно считать упорядоченным).

Временная сложность $O(n\log_2 n)$,



7. Пирамидальная сортировка.

Может рассматриваться как усовершенствованная сортировка пузырьком, в которой элемент всплывает по многим путям.

Временная сложность $O(n\log_2 n)$

ДЗ упорядочить массив по убыванию A=[1,7,9,15,10,0,3,8,18,5]

- – методами Сортировки Пузырьком и Быстрой сортировки. Псевдокод.
- – методами Сортировки Пузырьком и Сортировки вставками. Псевдокод.
- - методами сортировки Пузырьком и методом Выбора. Псевдокод.

Задачи проверочная: составить блок-схемы к алгоритму:

Задача 1. Дан массив из 11 элементов. Упорядочить по возрастанию методом выбора.

Задача 2. Дан массив 20 целых чисел, Упорядочить по невозрастанию методом вставок.

Задача 3. Дан массив 20 целых чисел, часть которых - 0. Упорядочить массив, удалив нули со сдвигом влево, следующими ненулевыми элементами.

Задача 4. Дан массив 15 целых чисел. Упорядочить массив, удалив повторяющиеся элементы

Алгоритмы поиска значения в массиве

1. Линейный поиск – алгоритм:

   поиск значения осуществляется сравнением значения текущего элемента и искомого - если значения совпадают (с той или иной точностью), то поиск считается завершённым. Временная сложность O(n).

2. Бинарный поиск- алгоритм:
   1. Определение значения элемента в середине структуры данных. Полученное значение сравнивается с ключом.
   2. Если ключ меньше значения середины, то поиск осуществляется в первой половине элементов, иначе — во второй.
   3. Поиск сводится к тому, что вновь определяется значение серединного элемента в выбранной половине и сравнивается с ключом.
   4. Процесс продолжается до тех пор, пока не будет найден элемент со значением ключа или не станет пустым интервал для поиска.

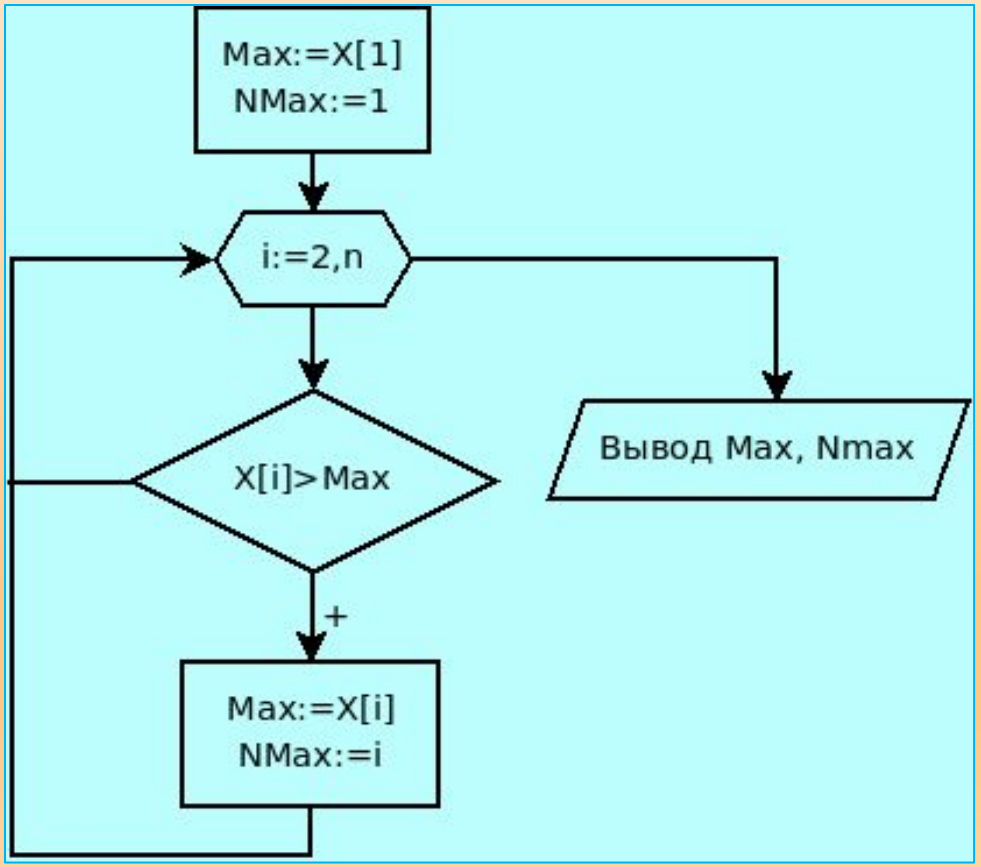
Временная сложность O(log(n))

- 3.Интерполяционный поиск - алгоритм производит предсказание области местонахождения элемента по расстоянию между ключом и текущим значением элемента.
- В сортированном массиве-можно применять линейную интерполяцию
- Средняя временная сложность в худшем случае O(n)

Одномерные и двумерные массивы - основные задачи.

1.1.Одномерные массивы.

1.1.Поиск max, min элемента массива



- Задачи:
- Задача 1. Дан массив целых чисел. Количество чисел N вводится с клавиатуры. Все числа различны.

  Найти:

  •максимальный элемент массива и его номер,
  •минимальный элемент массива и его номер.
- Задача 2. Дан одномерный целочисленный *массив*, заданный случайными числами на промежутке [-30; 30).

  Посчитать количество вхождений элемента со значение 10.

1.1.Двумерные массивы:

1. Объявление массива
2. Инициализация, заполнение
3. Сортировка элементов в столбце или строке
4. Поиск элементов в столбце или строке или массиве.
5. Замена элементов массива

Семестр 2.

Дисциплина
Вычислительная математика.

Алгоримы на Python.

Современная вычислительная математика включает в круг своих проблем изучение особенностей вычисления с применением компьютеров.

На её основе в последнее десятилетие образовались такие новые области естественных наук, как вычислительная экономика, вычислительная физика, вычислительная химия, вычислительная биология.

Вычислительная математика – это раздел математики, включающий в себя теорию численных методов и алгоритмов решения математических задач

В вычислительной математике выделяют следующие направления:

1. анализ математических моделей,
2. разработка методов и алгоритмов решения стандартных математических задач,
3. программирование (автоматизация) алгоритмов решения задач.

Глава 1. Структуры данных: Строки, Списки, Деревья.

П.1. Тип - Строки

Опр. 1. Строка – это одномерный массив данных типа char - значениями которого является произвольная последовательность (строка) символов алфавита.

Каждая переменная такого типа (**строковая переменная**) может быть представлена фиксированным количеством байтов либо иметь произвольную длину.

Под **строками в Python** подразумевается набор символов между кавычками. В **Python** можно использовать пары одинарных либо двойных кавычек.

В языках программирования возможны следующие операции со строками:

1. получение символа по индексу;
2. конкатенация (соединение) строк;
3. получение подстроки по индексам начала и конца;
4. проверка вхождения одной строки в другую (поиск подстроки в строке);
5. проверка на совпадение строк (с учётом или без учёта регистра);
6. получение длины строки;
7. замена подстроки в строке;
8. нахождение минимальной надстроки, содержащей все указанные строки;
9. поиск в двух массивах строк совпадающих последовательностей задача о плагиате и т д.

Функции и процедуры со строковыми переменными в ПИТОНЕ

Функция или метод	Назначение
S = 'str'; S = "str";	Литералы строк
S1 + S2	Конкатенация (сложение строк)
S1 * 3	Повторение строки
S[i]	Обращение по индексу
S.rfind(str, [start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или -1

Продолжение - функции строк в Питоне

Функция или метод	Назначение
S[i:j:step]	Извлечение среза
len(S)	Длина строки
S.find(str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или -1
S.index(str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или вызывает ValueError
S.rindex(str, [start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или вызывает ValueError

П.2. Динамические структуры в ЯП.

- **Динамические структуры данных** – это структуры данных, *память* под которые выделяется и освобождается по мере необходимости.
- *Динамические структуры данных* в процессе существования в памяти могут изменять не только число составляющих их элементов, но и характер связей между элементами.

Динамическая структура данных характеризуется тем, что:

1. она не имеет имени;
2. этой структуре выделяется память в процессе выполнения программы, а не компиляции;
3. количество элементов структуры может не фиксироваться;
4. размерность структуры может меняться в процессе выполнения программы;
5. в процессе выполнения программы может меняться характер взаимосвязи между элементами структуры.

Особенности динамических структур

Каждой динамической структуре данных сопоставляется *статическая переменная* типа *указатель* (ее *значение – адрес* этого объекта), с помощью которой осуществляется *доступ* к динамической структуре.

Сами динамические величины не требуют описания в программе, поскольку во *время компиляции память* под них не выделяется.

- Для установления связи между элементами динамической структуры используются указатели, через которые устанавливаются явные связи между элементами.

  Такое *представление* данных в памяти называется *связным*.

- Достоинства связного представления данных:

1. размер структуры ограничивается только доступным объемом машинной памяти;
2. при изменении логической последовательности элементов структуры требуется не перемещение данных в памяти, а только коррекция указателей;
3. большая гибкость структуры.

Порядок работы с *динамическими структурами данных* :

1. создать (отвести место в динамической памяти);
2. работать при помощи указателя;
3. удалить (освободить занятое структурой место).

Классификация динамических структур данных:

Для представления данных, у которых *конфигурация*, размеры и состав могут меняться в процессе выполнения программы используют следующие динамические структуры:

1. однонаправленные (односвязные) списки;
2. двунаправленные (двусвязные) списки;
3. циклические списки;
4. стек;
5. дек;
6. очередь;
7. бинарные деревья.

Данные структуры отличаются способом связи отдельных элементов и/или допустимыми операциями.

Динамическая структура может занимать несмежные участки оперативной памяти.

- Динамические структуры широко применяют и для более эффективной работы с данными, размер которых известен, особенно для решения задач сортировки.

п.3. Тип данных - 2).Списки

Опр.2. **Списком** называется упорядоченное множество, состоящее из переменного числа элементов, к которым применимы *операции включения*, исключения.

Список, отражающий отношения соседства между элементами, называется *линейным*.

Длина списка равна числу элементов, содержащихся в списке, *список* нулевой длины называется *пустым списком*.

В языках программирования возможны следующие операции со списками:

1.создание списка;
2.печать (просмотр) списка;
3.вставка элемента в список;
4.удаление элемента из списка;
5.поиск элемента в списке
6.проверка пустоты списка;
7.удаление списка.

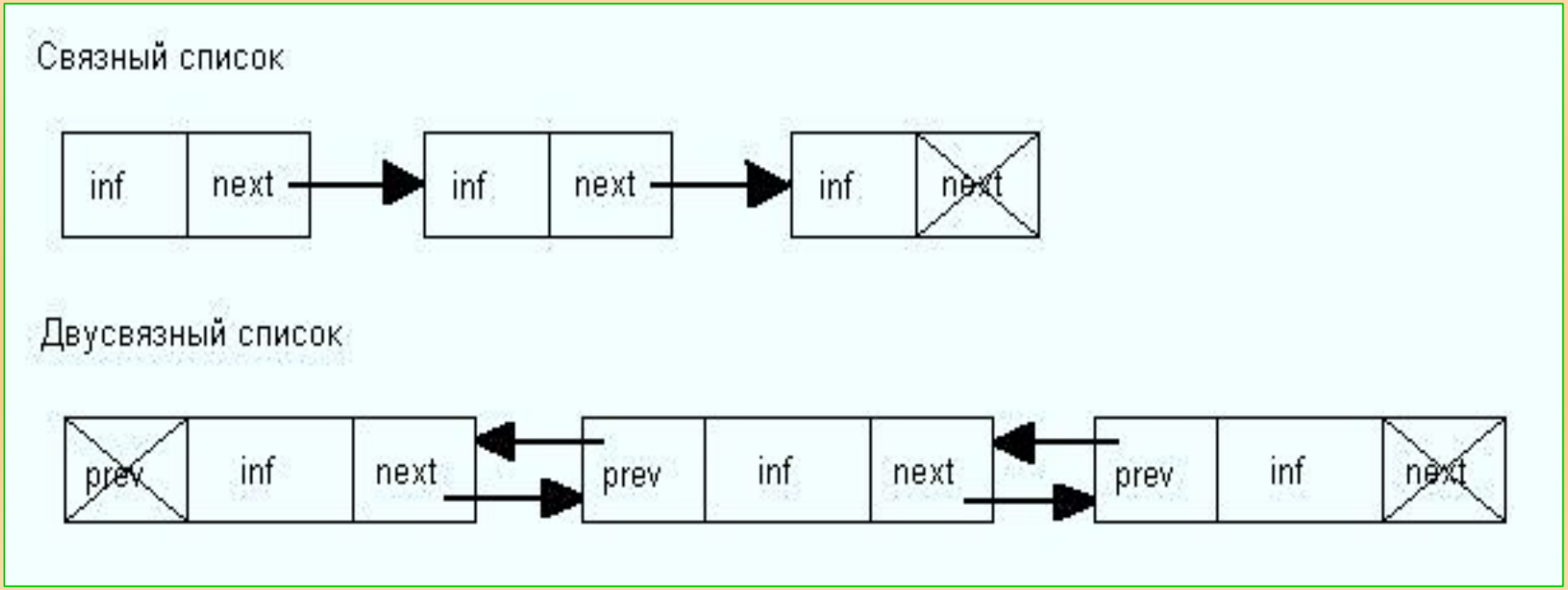
3.1.Односвязные списки.

Опр.3. Односвязный список – это *структура данных*, представляющая собой последовательность элементов, в каждом из которых хранится *значение* и *указатель* на следующий *элемент списка*. В последнем элементе *указатель* на следующий элемент равен NULL.

3.2. Двусвязный список

- **Опр.4. Двунаправленный (двусвязный) список –** это *структура данных*, состоящая из последовательности элементов, каждый из которых содержит информационную часть и два указателя на соседние элементы.

При этом два соседних элемента должны содержать взаимные ссылки друг на друга.



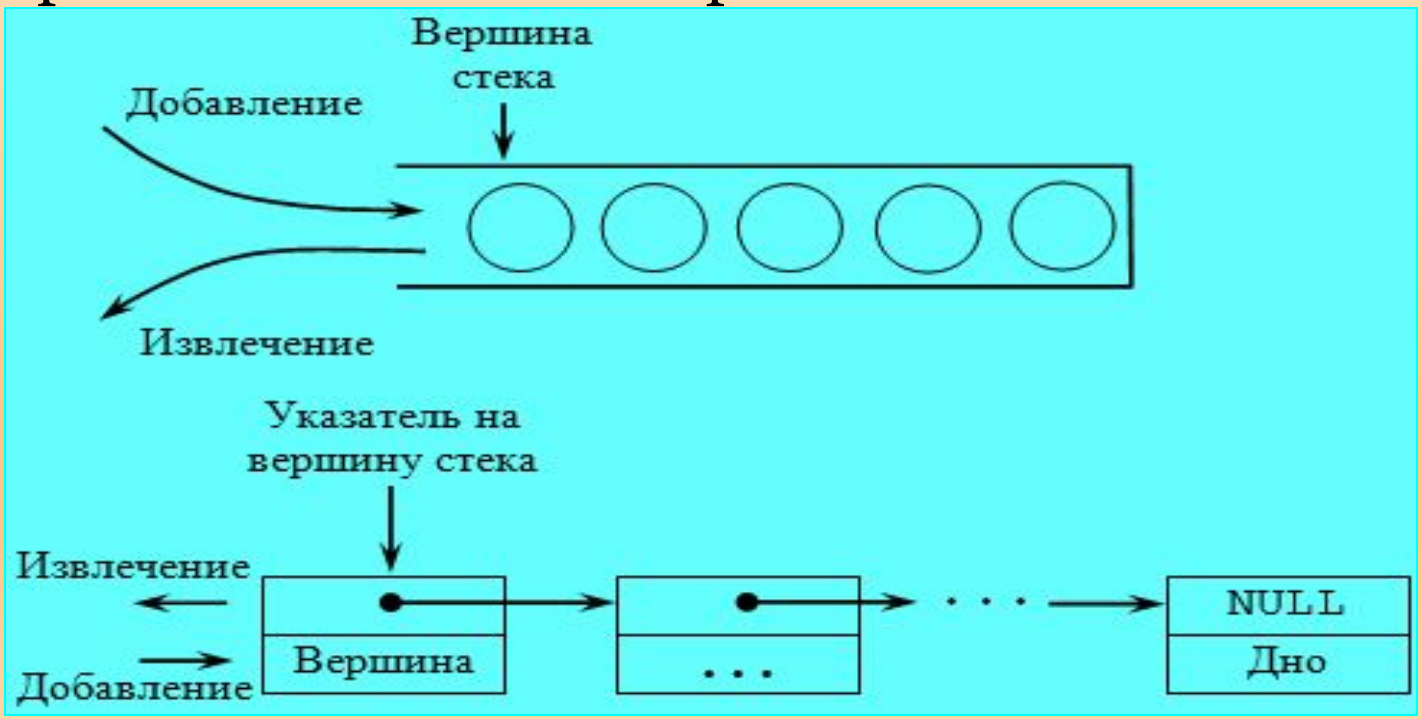
Списки в Питон – это упорядоченные изменяемые коллекции объектов произвольных типов. **В Питоне к списками применимы встроенные функции и методы:**

Метод	Что делает
list.append(x)	Добавляет элемент в конец списка
list.extend(L)	Расширяет список list, добавляя в конец все элементы списка L
list.insert(i, x)	Вставляет на i-ый элемент значение x
list.remove(x)	Удаляет первый элемент в списке, имеющий значение x. ValueError, если такого элемента не существует
list.copy()	Поверхностная копия списка
list.clear()	Очищает список

Продолжение - Методы списков в Питоне

Метод	Что делает
list.pop([i])	Удаляет i-ый элемент и возвращает его. Если индекс не указан, удаляется последний элемент
list.index(x, [start [, end]])	Возвращает положение первого элемента со значением x (при этом поиск ведется от start до end)
list.count(x)	Возвращает количество элементов со значением x
list.sort([key=функция])	Сортирует список на основе функции
list.reverse()	Разворачивает список

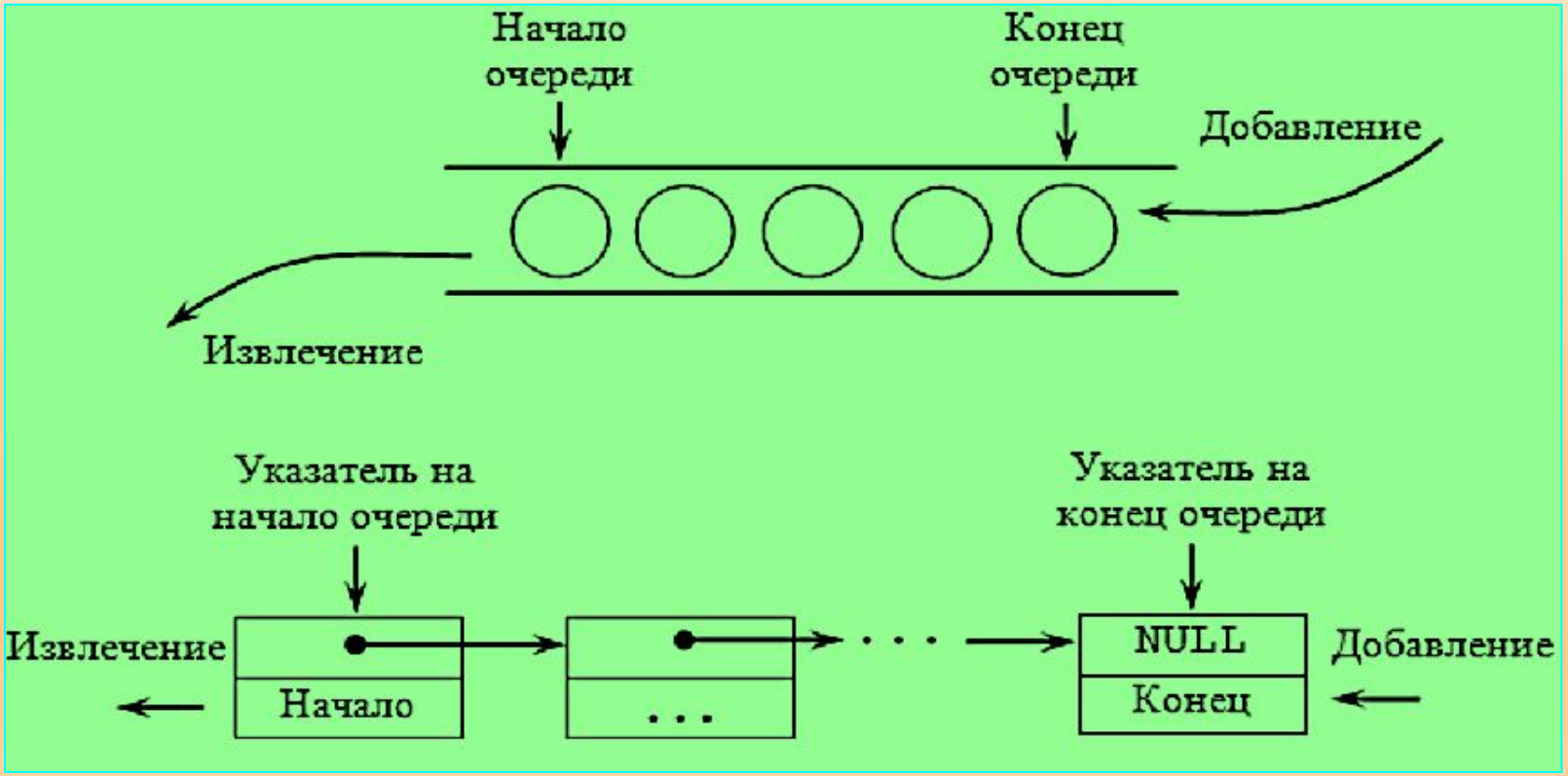
- **Типы 3.Стек и 4.Очередь** -*динамические структуры данных, которые* можно организовать на основе линейного списка.
- 3). **Стек** – это *структура данных*, в которой новый элемент всегда записывается в ее начало (вершину) и очередной читаемый элемент также всегда выбирается из ее начала-вершины.



- Операции со стэком:

1. создание стека;
2. печать (просмотр) стека;
3. добавление элемента в *вершину стека*;
4. извлечение элемента из *вершины стека*;
5. проверка пустоты стека;
6. удаление стека.

- **4) Очередь** – это динамическая *структура данных*, представляющая собой последовательность элементов, образованная в порядке их поступления.
- Каждый новый элемент размещается в конце очереди; элемент, стоящий в начале очереди, выбирается из нее первым.

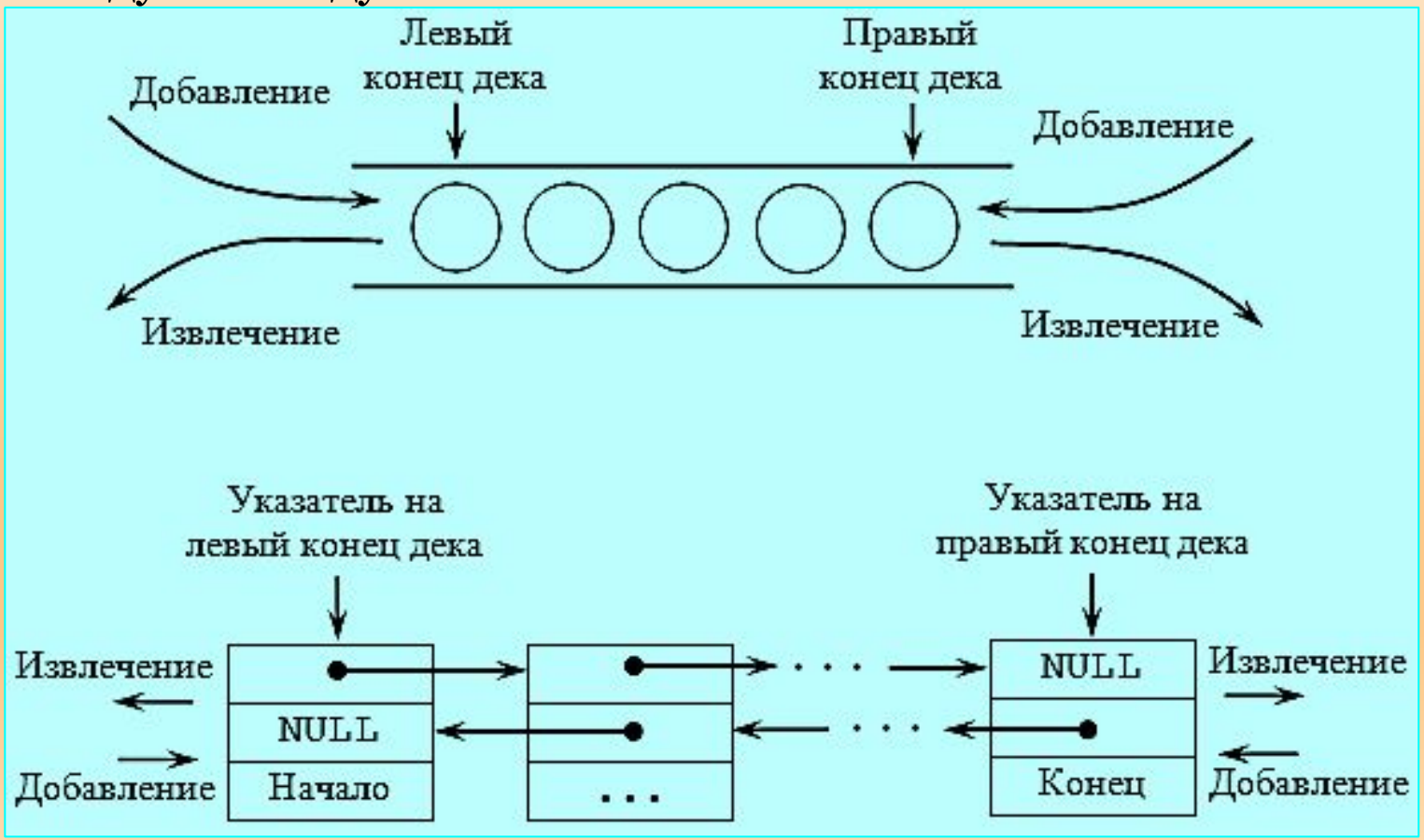


- Основные *операции*, производимые с очередью:

1. создание очереди;
2. печать (просмотр) очереди;
3. добавление элемента в конец очереди;
4. извлечение элемента из начала очереди;
5. проверка пустоты очереди;
6. очистка очереди.

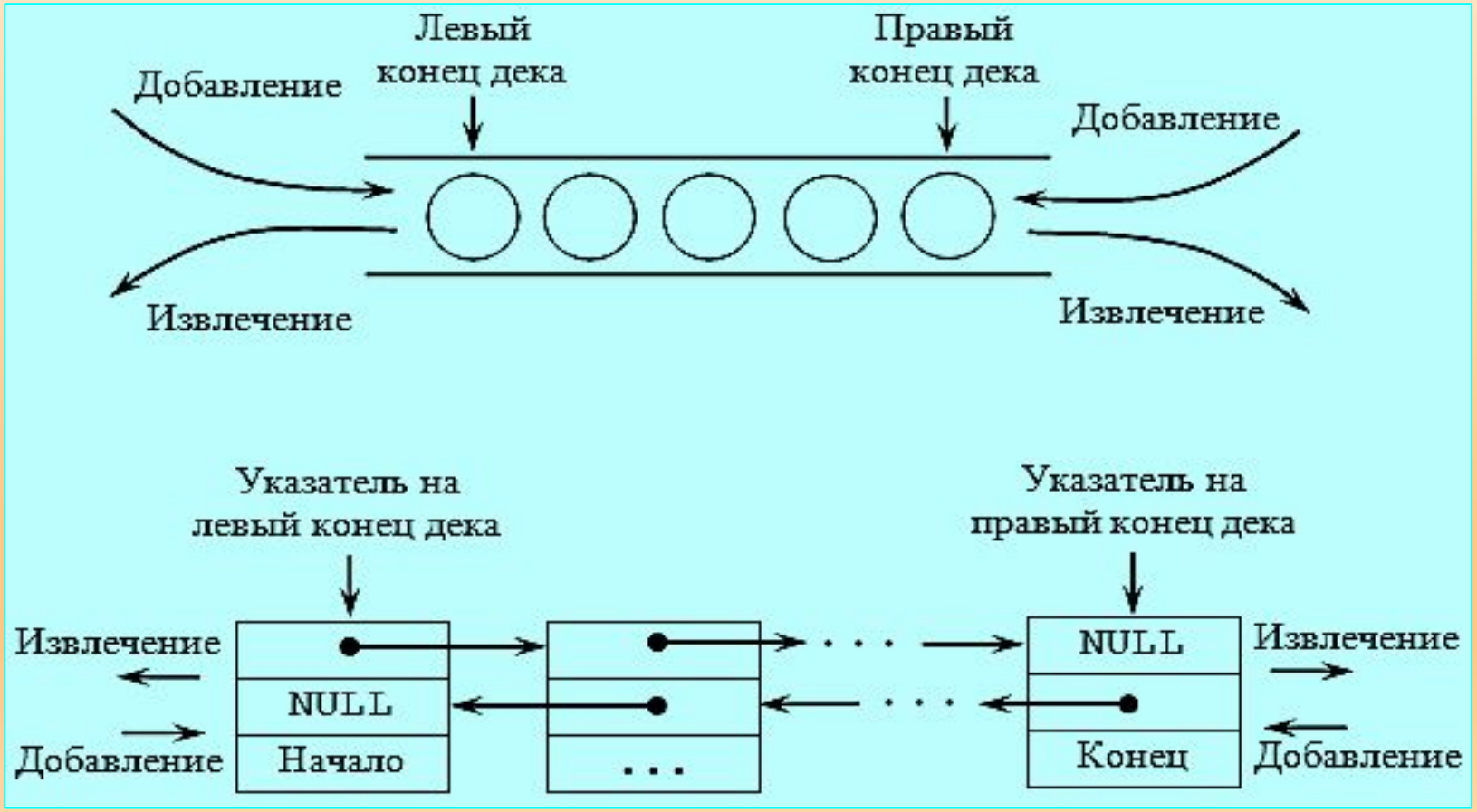
П.5.Тип данных 5.Дэк (дек)

Опр.5. Дек (двухсторонняя *очередь*) – это *структура данных*, представляющая собой последовательность элементов, в которой можно добавлять и удалять в произвольном порядке элементы с двух сторон . Первый и последний элементы *дека* соответствуют входу и выходу *дека*.



Опр.6.*дек* с ограниченным входом – из конца *дека* можно только извлекать элементы;

Опр.7.*дек* с ограниченным выходом – в конец *дека* можно только добавлять элементы.



Основные операции с деком:

1.создание *дека*;
2.печать (просмотр) *дека*;
3.добавление элемента в левый конец *дека*;
4. добавление элемента в правый конец *дека*;
5.извлечение элемента из левого конца *дека*;
6.извлечение элемента из правого конца *дека*;
7.проверка пустоты *дека*;
8.очистка *дека*.

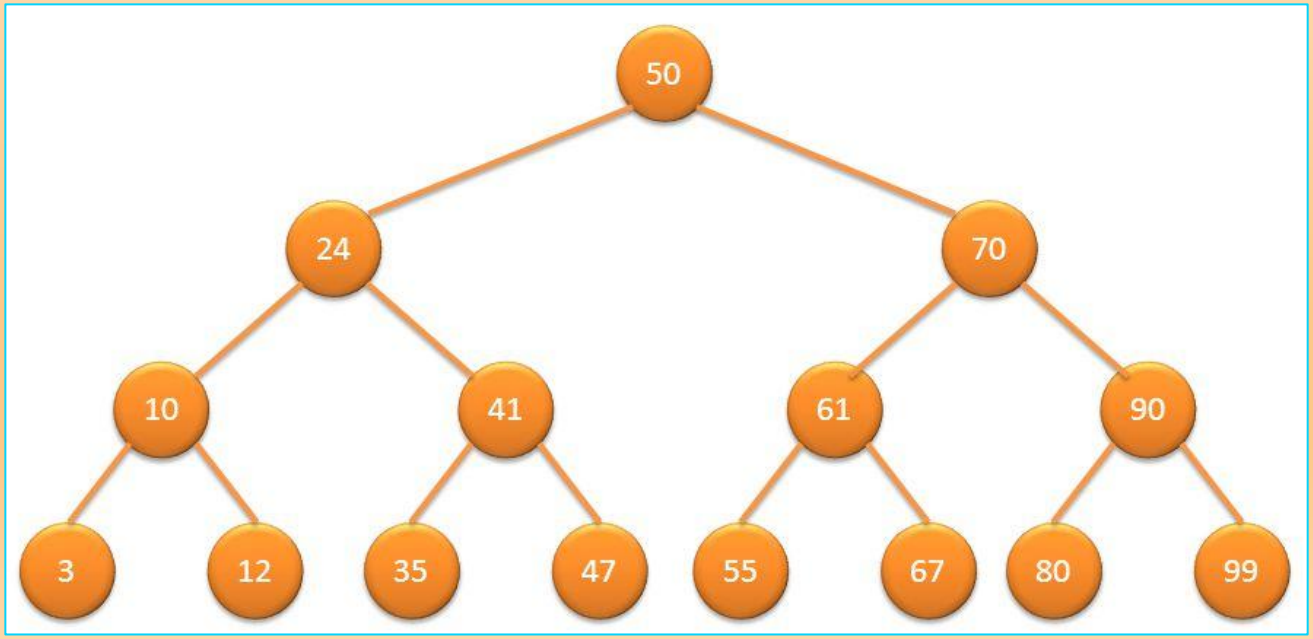
П.6.Тип данных-Деревья

Опр.8. Дерево – это динамическая *структура данных*, представляющая собой совокупность элементов и отношений, образующих иерархическую структуру этих элементов.

Опр.9. Вершина (узел) дерева – это каждый элемент дерева.

Опр.10. Ветви дерева – это направленные дуги, которыми соединены вершины дерева.

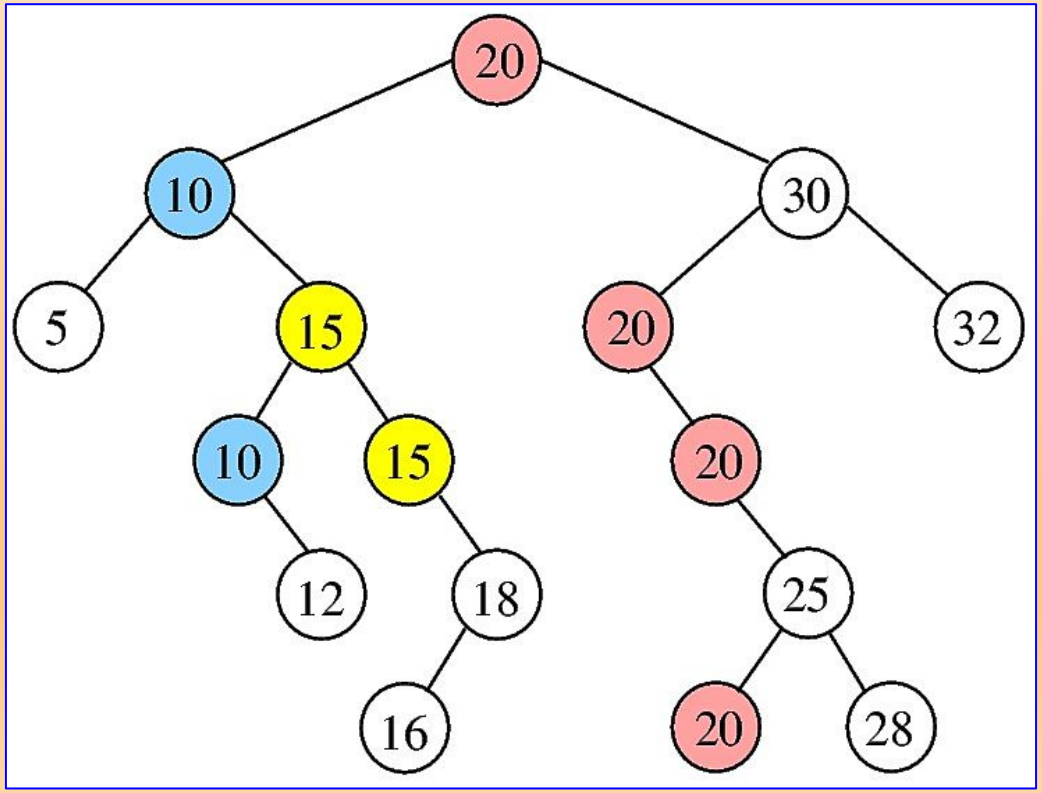
Опр.11. Высота (глубина) дерева – это количество уровней, на которых располагаются его вершины.



Опр. 12. Корень дерева – это начальный узел дерева, ему соответствует нулевой уровень.

Опр.13. Листья дерева – это вершины, в которые входит одна *ветвь* и не выходит ни одной ветви.

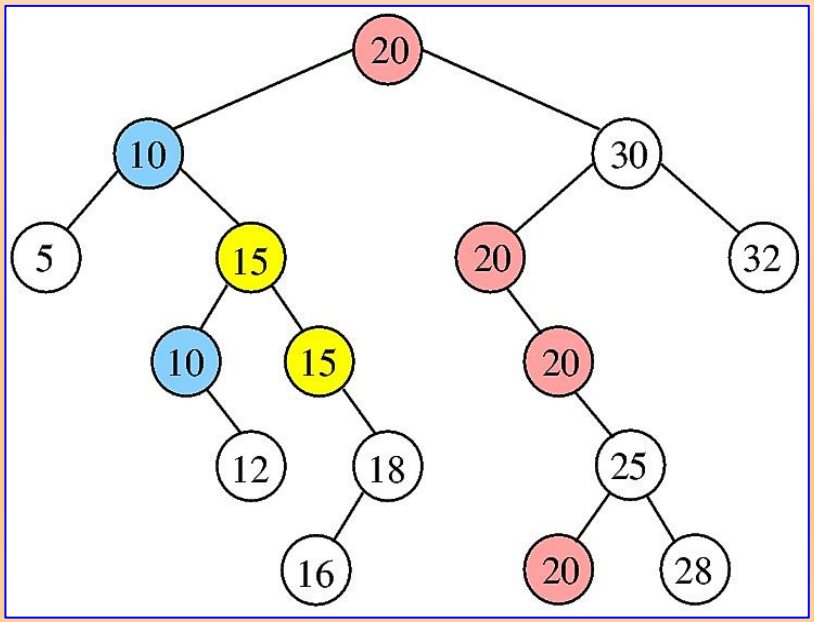
Опр.14. Поддерево – это часть древообразной структуры данных, которая может быть представлена в виде отдельного дерева.

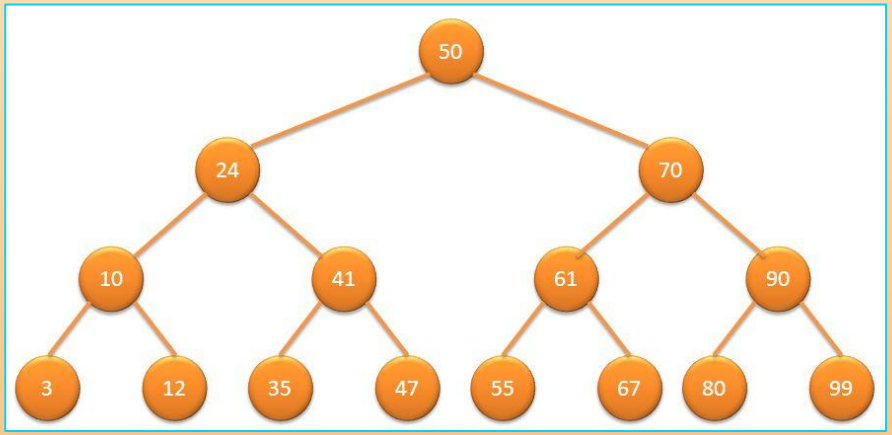


Опр.15. Потомки – это все вершины, в которые входят ветви, исходящие из одной общей вершины.

Опр.16. Предок – это *вершина*, из которой исходят ветви к вершинам следующего уровня.

Опр.17. Сбалансированное дерево – это *дерево*, у которого длины всех путей от корня к внешним вершинам равны между собой.





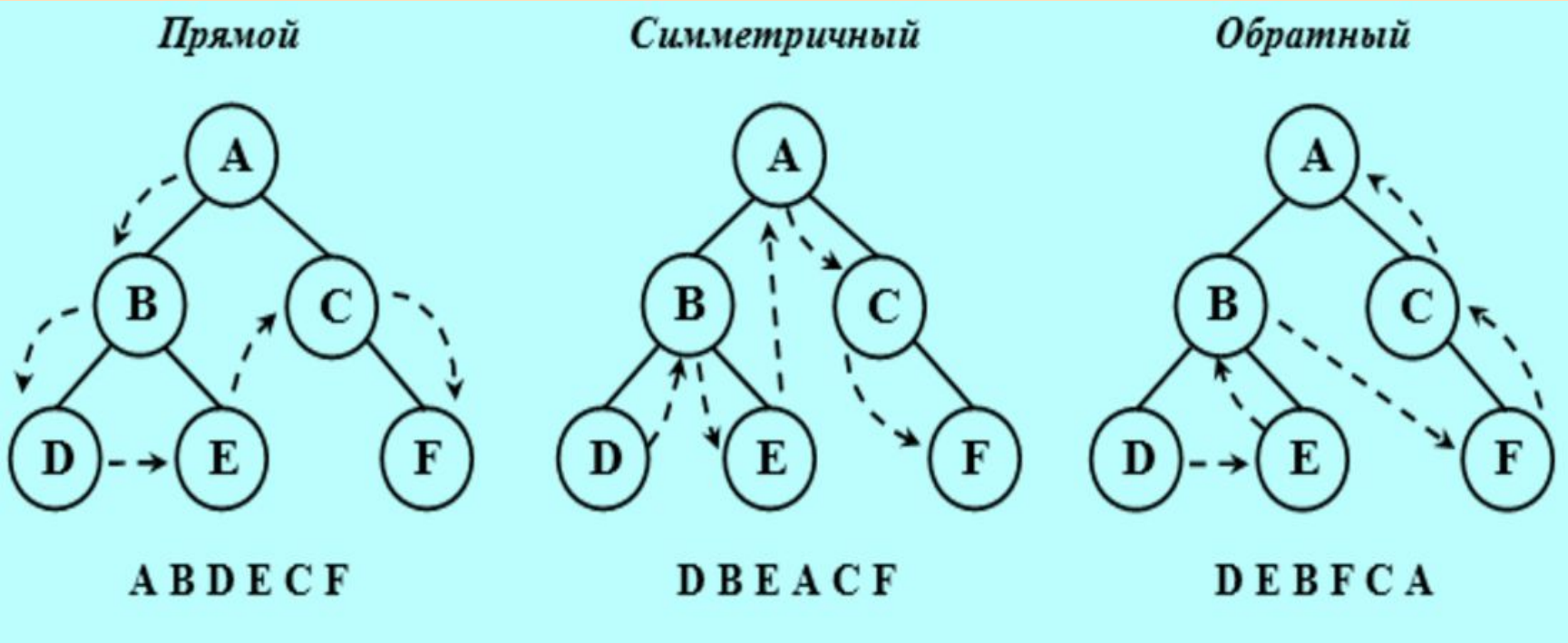
Опр.18.Степень вершины – это количество дуг, которое выходит из этой вершины.

Опр.19.Степень дерева – это *максимальная степень* вершин, входящих в *дерево*.

Опр.20.Упорядоченное дерево – это *дерево*, у которого ветви, исходящие из каждой вершины, упорядочены *по* определенному критерию.

Опр.21.Уровень вершины – это количество дуг от корня дерева до вершины.

- **Опр. 22. Обход дерева** – это упорядоченная последовательность вершин дерева, в которой каждая *вершина* встречается только один раз.

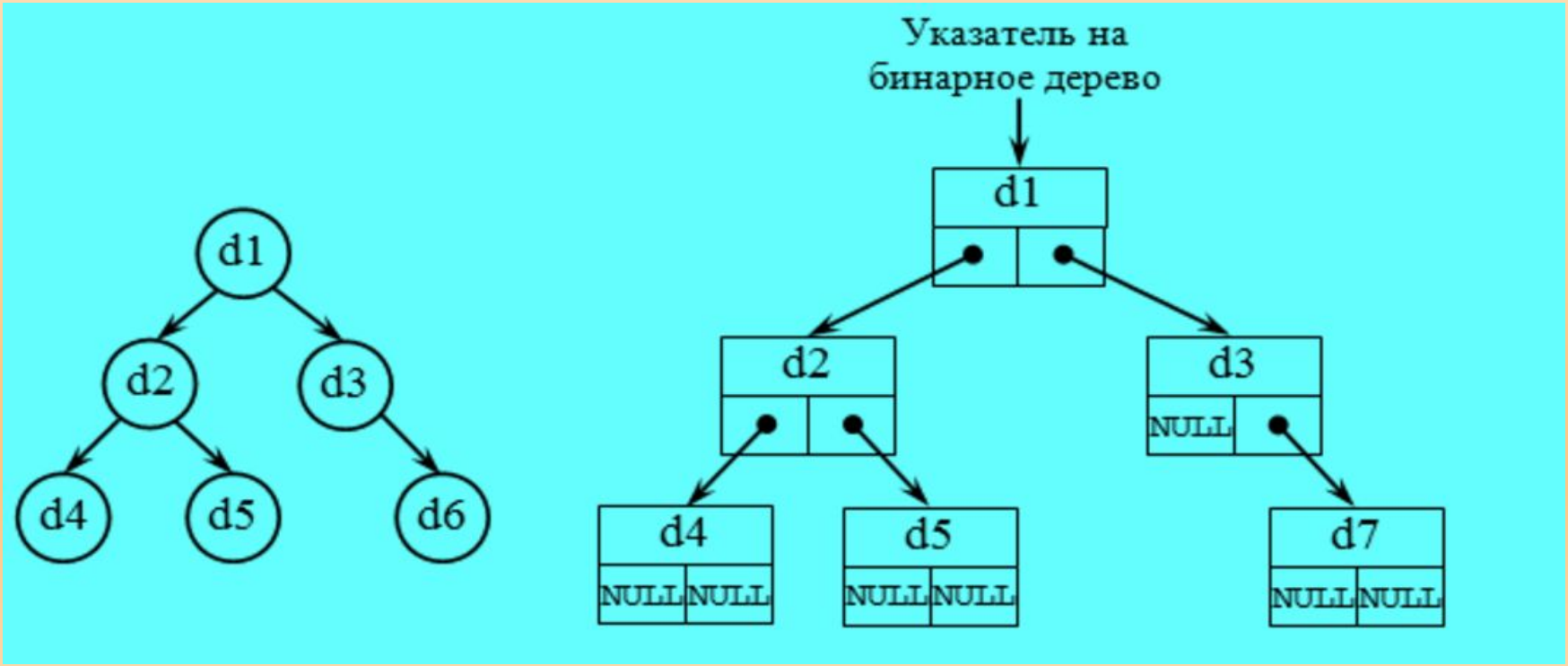


П.7.Тип данных 7. Бинарные деревья

Опр. 23. Бинарное (двоичное) дерево – это *дерево*, в котором каждая *вершина* имеет не более двух потомков.

Опр. 24.Неполное бинарное дерево – это *дерево*, уровни которого заполнены не полностью.

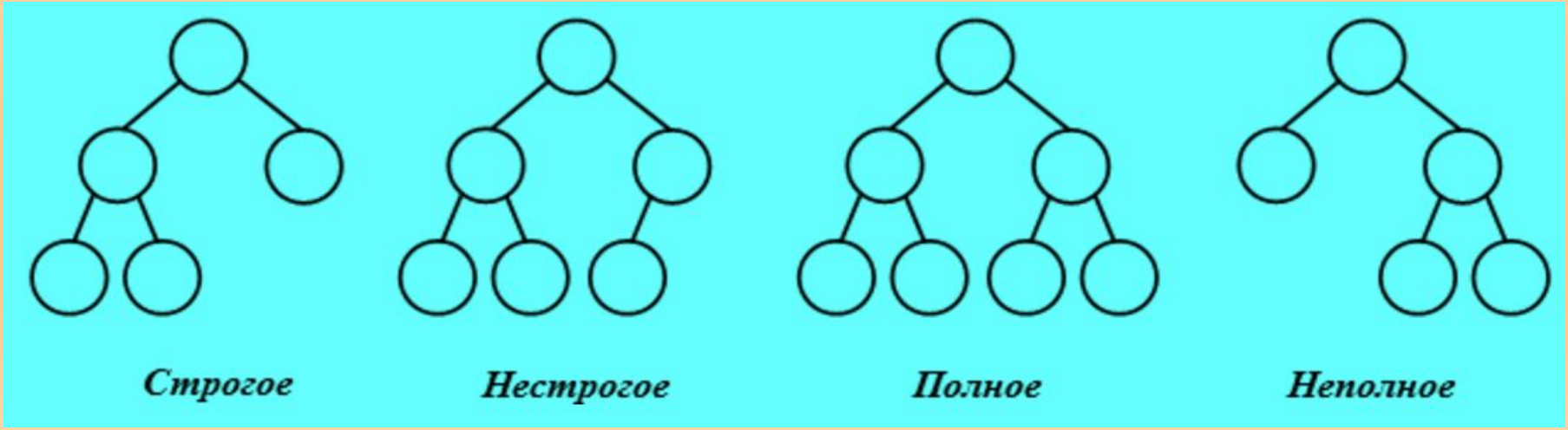
Опр. 25. Полное бинарное дерево – это *дерево*, которое содержит только полностью заполненные уровни.



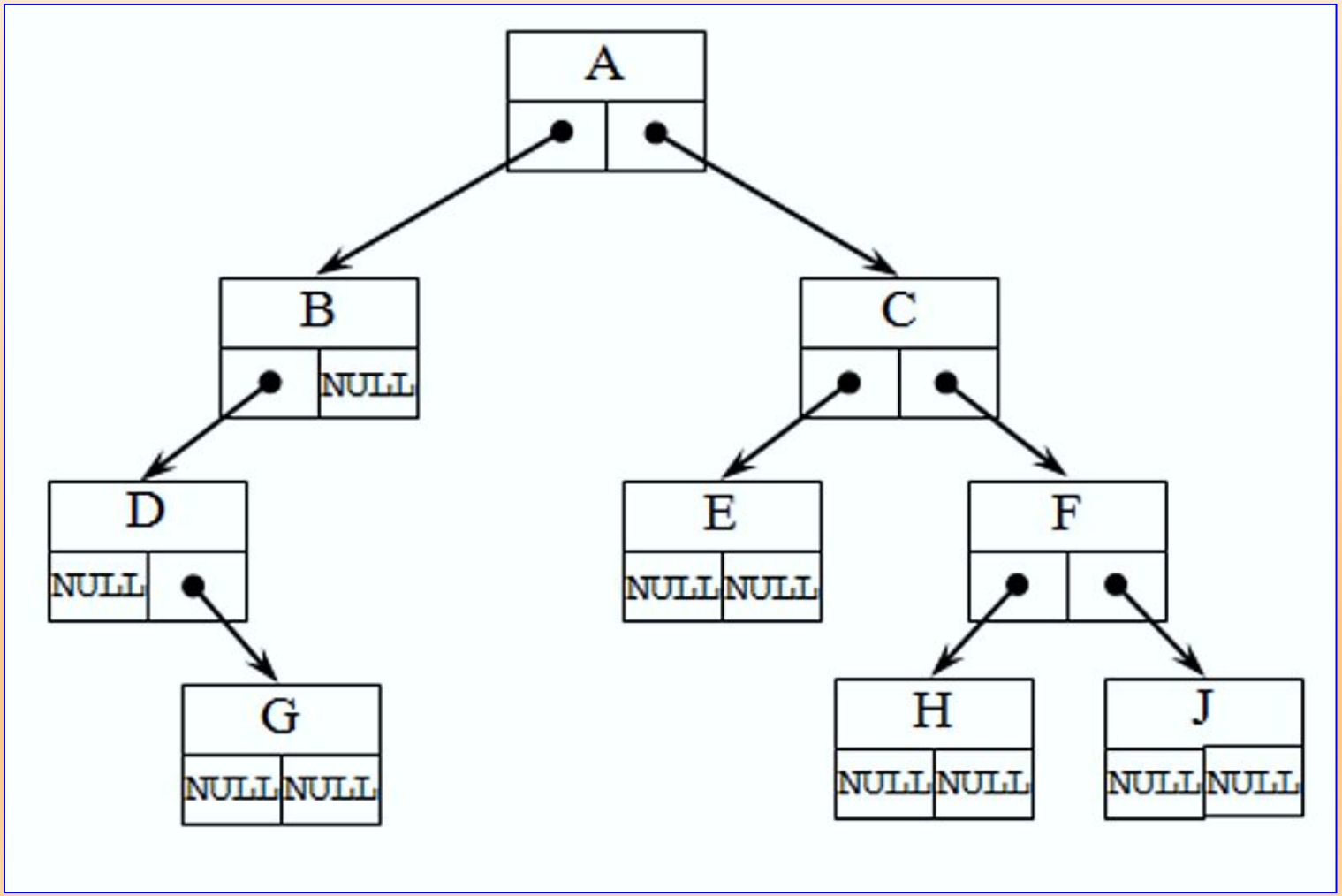
Каждая *вершина бинарного дерева* является структурой, состоящей из четырех видов полей:

1. информационное поле (ключ вершины);
2. служебное поле (их может быть несколько или ни одного);
3. указатель на *левое поддерево*;
4. указатель на правое *поддерево*.

По степени вершин бинарные деревья делятся на:



Адресация в бинарном дереве



Основными операциями с бинарными деревьями, являются:

1. создание *бинарного дерева*;
2. печать *бинарного дерева*;
3. обход *бинарного дерева*;
4. вставка элемента в *бинарное дерево*;
5. удаление элемента из *бинарного дерева*;
6. проверка пустоты *бинарного дерева*;
7. удаление *бинарного дерева*.

Глава 2. Задача поиска подстроки в строке.

П.1. Определения.

- **Алфавит** – конечное множество символов.
- **Строка (слово)** – это последовательность символов из некоторого алфавита.
- **Длина строки** – количество символов в строке.
- Строка, не содержащая ни одного символа, называется **пустой**.

Опр.1. Строка X называется **подстрокой** строки Y, если найдутся такие строки Z_1 и Z_2, что $Y=Z_1XZ_2$. Z_1 называют левым, а Z_2 – правым крылом подстроки.

Опр. 2. Подстрока X называется **префиксом** строки Y, если есть такая подстрока Z, что Y=XZ.

Опр. 3. Подстрока X называется **суффиксом** строки Y, если есть такая подстрока Z, что Y=ZX.

- Строка является подстрокой, префиксом и суффиксом себя самой.

Постановка задачи поиска подстроки в строке:

Пусть задана строка, состоящая из некоторого количества символов.

1) . Проверить - входит ли заданная *подстрока* в данную строку.

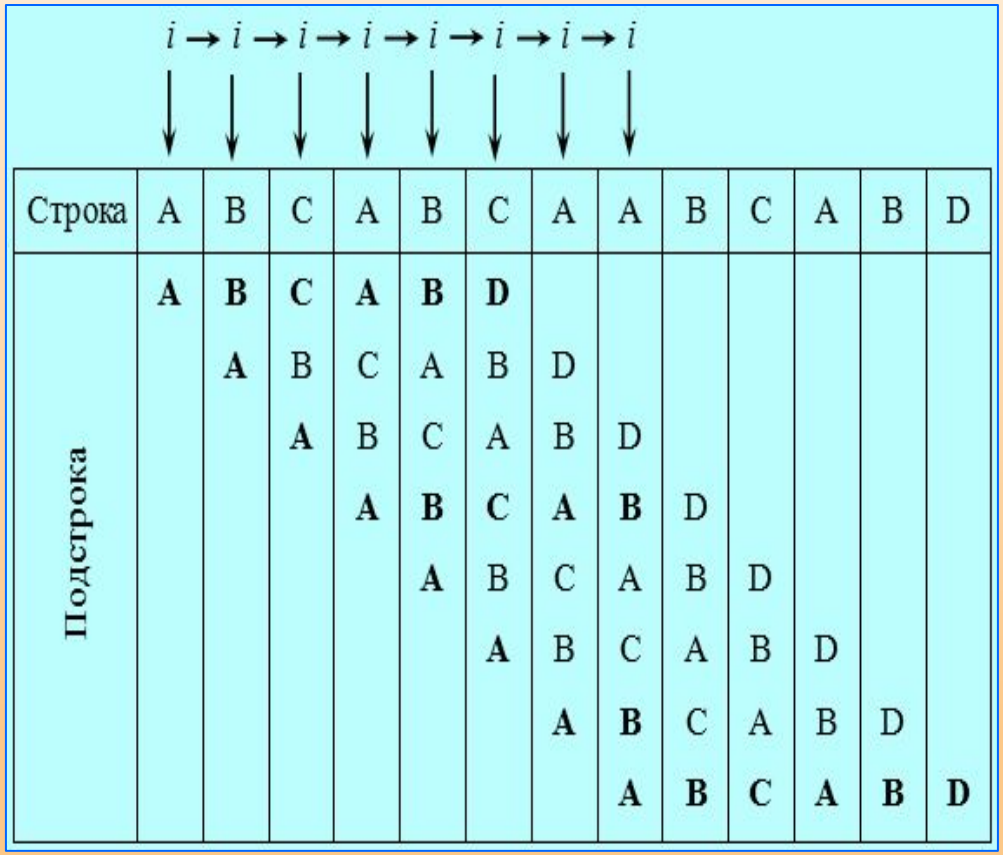
2). Если входит, то найдем номер, начиная с какого символа строки начинается первое вхождение заданной подстроки в исходной строке.

П.2. Наивный (прямой) алгоритм поиска подстроки в строке.

Суть алгоритма:

1). В начальный момент происходит посимвольное сравнение первого символа строки с первым символом подстроки, второго символа строки со вторым символом подстроки и т. д. Если произошло совпадение всех символов, то фиксируется факт нахождения подстроки и индекс 1-го эл.

2). Если не совпадает, то - сделать сдвиг подстроки на одну позицию вправо и повторить посимвольное сравнение и т.д. до совпадения подстроки либо конца строки.



- **Алгоритм прямого поиска**

Идея алгоритма:
1. I=1,
2. сравнить I-й символ массива T с первым символом массива W,
3. совпадение → сравнить вторые символы и так далее,
4. несовпадение → I:=I+1 и переход на пункт 2,

Условие окончания алгоритма:
1. подряд M сравнений удачны,
2. I+M>N, то есть слово не найдено.

Сложность алгоритма:
Пусть массив T→{AAA….AAAB}, длина |T|=N, образец W→{A….AB}, длина |W|=M. Очевидно, что для обнаружения совпадения в конце строки потребуется произвести порядка N*M сравнений, то есть **O(N*M).**

Вычислительная математика.

- **Лекция 2.**
- Алгоритмы поиска Кнутта-Морис –Пратта.
- Хэширование.
- Алгоритм поиска Рабина-Карпа.

П.3.Алгоритм Кнута, Морриса и Пратта. Префикс-функция:

- Пусть дана строка S длины n.
- **Опр.4. Префикс-функция – это массив, каждый элемент которого вычисляется, как наибольшая длина собственного префикса подстроки S[1,..i], совпадающего с его суффиксом для каждого i=1,n, где n –длина строки.**

- Префикс-функция вычисляется для строки S и каждого индекса позиции i:
  - Для постороки S[1,1], - длина 1
  - Для постороки S[1,2] - длина 2
  - ……..
  - Для постороки S[1,n] – длина n.

  - Важно!!! Если найденный на предыдущем шаге префикс не может быть расширен на следующую позицию, то мы пытаемся рассматривать меньшие префиксы до тех пор, пока это возможно.

Пример 1. Для строки **"abcabcd"** префикс-функция равна: [0,0,0,1,2,3,0], так как:

1. у строки "a" нет нетривиального префикса, совпадающего с суффиксом;
2. у строки "ab" нет нетривиального префикса, совпадающего с суффиксом;
3. у строки "abc" нет нетривиального префикса, совпадающего с суффиксом;
4. у строки "abca" префикс длины 1 совпадает с суффиксом;
5. у строки "abcab" префикс длины 2 совпадает с суффиксом;
6. у строки "abcabc" префикс длины 3 совпадает с суффиксом;
7. у строки "abcabcd" нет нетривиального префикса, совпадающего с суффиксом.

Пример 2. ДЗ Вычислить префикс-функцию для строки **"aabaaab"**

Ответ на пример 2

- (она равна: [0,1,0,1,2,2,3])

Алгоритм Кнута-Мориса-Пратта

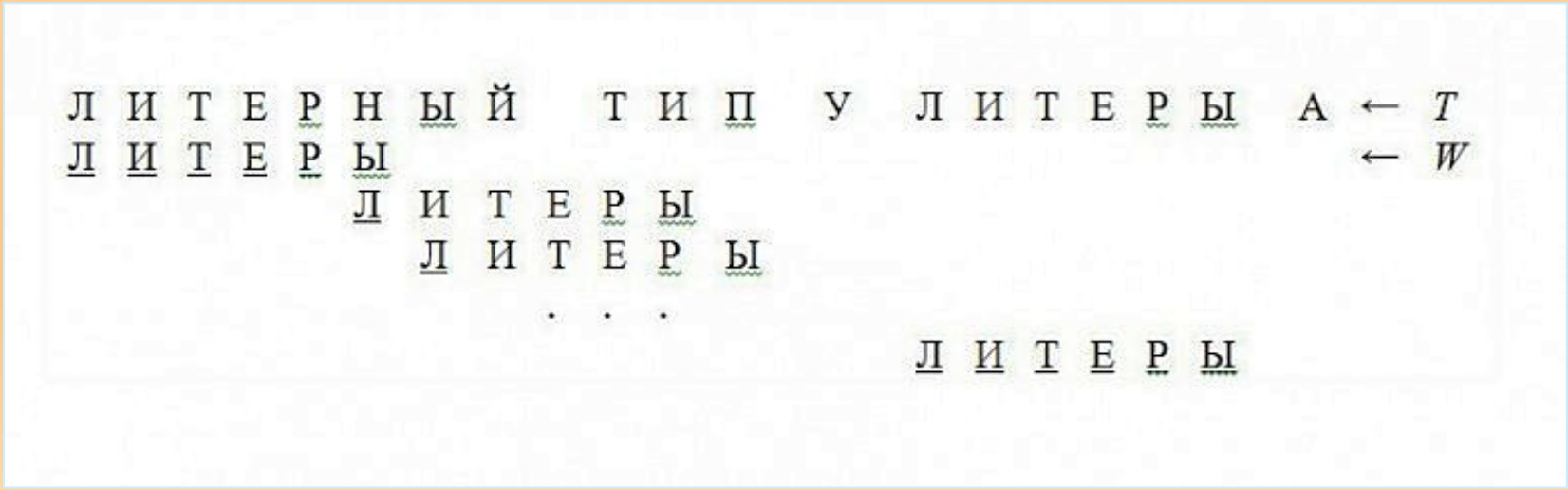
Суть алгоритма: сдвиг подстроки выполняется не на один символ на каждом шаге алгоритма, а на некоторое переменное количество символов.

То есть надо каждый раз определять max большую величину сдвига.

Пример 1.

После частичного совпадения начальной части образа W
с соответствующими символами строки T мы фактически знаем
пройденную часть строки и может «вычислить» некоторые сведения
(на основе самого образа W),
с помощью которых потом быстро продвинемся по тексту.

Идея КМП-поиска – при каждом несовпадении двух символов текста
и образа образ сдвигается на все пройденное расстояние,
так как меньшие сдвиги не могут привести к полному совпадению.

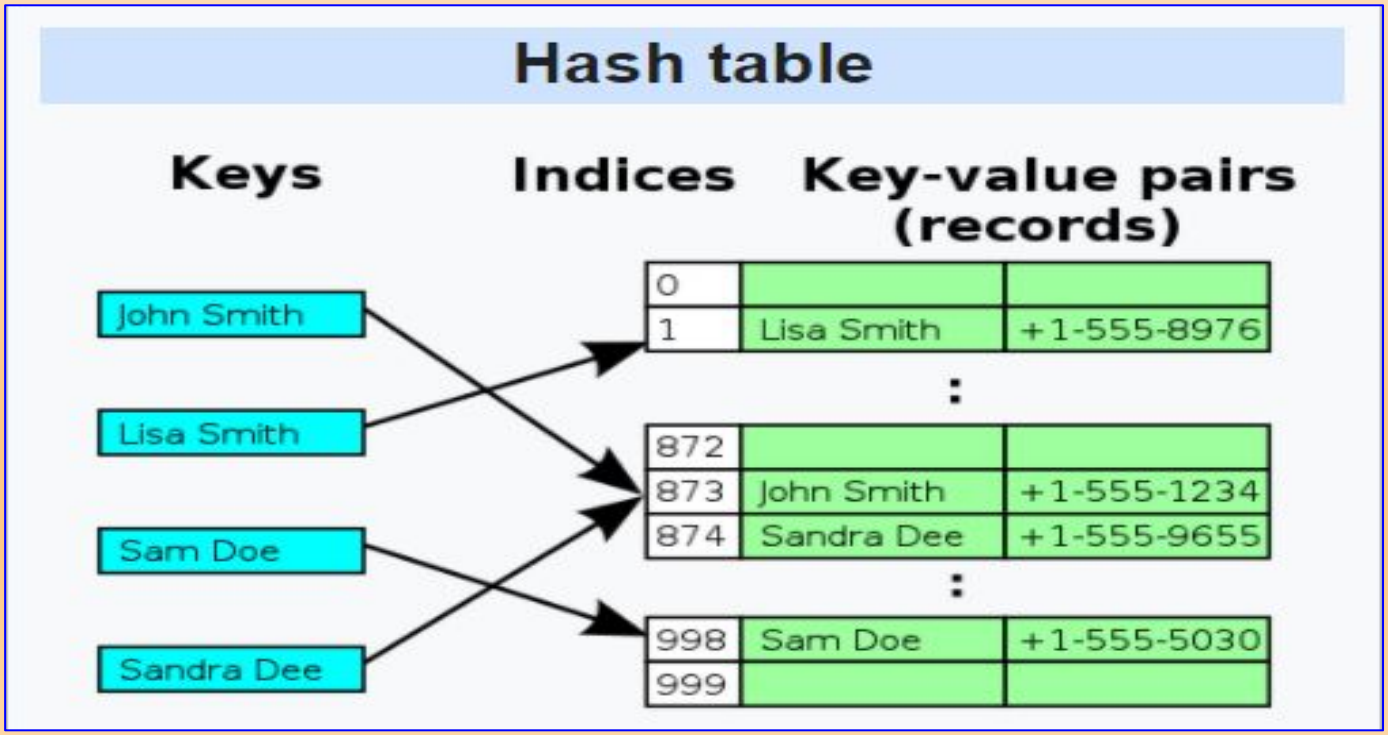


П.4. Хеширование

- **Опр. 5. Хеш-функция** — функция, которая по определенному алгоритму преобразовывает массив входных данных произвольной длины в выходную битовую строку установленной длины.
- Хеш-функция формирует *хеш-таблицу*.
- **Опр. 6.Хеширование** – это преобразование входного массива данных определенного типа и произвольной длины **в выходную битовую строку фиксированной длины.**

Опр. 7.Хеш-таблица – это *структура данных*, реализующая *интерфейс* ассоциативного массива, она позволяет хранить пары вида "*ключ-значение*" и выполнять три *операции*:

1. операцию добавления новой пары,
2. операцию поиска
3. операцию удаления пары по ключу.



- **Опр. 7.Коллизия** –ситуация, когда разным ключам соответствует одно значение *хеш-функции* (ключи в этом случае называются *синонимами*).

- *Хеш-функция должна удовлетворять* следующим условиям:

1. функция должна быть простой с вычислительной точки зрения;
2. функция должна распределять ключи в хеш-таблице наиболее равномерно;
3. функция не должна отображать какую-либо связь между значениями ключей в связь между значениями адресов;
4. функция должна минимизировать число *коллизий* – то есть ситуаций, когда разным ключам соответствует одно значение *хеш-функции*(ключи в этом случае называются *синонимами*).

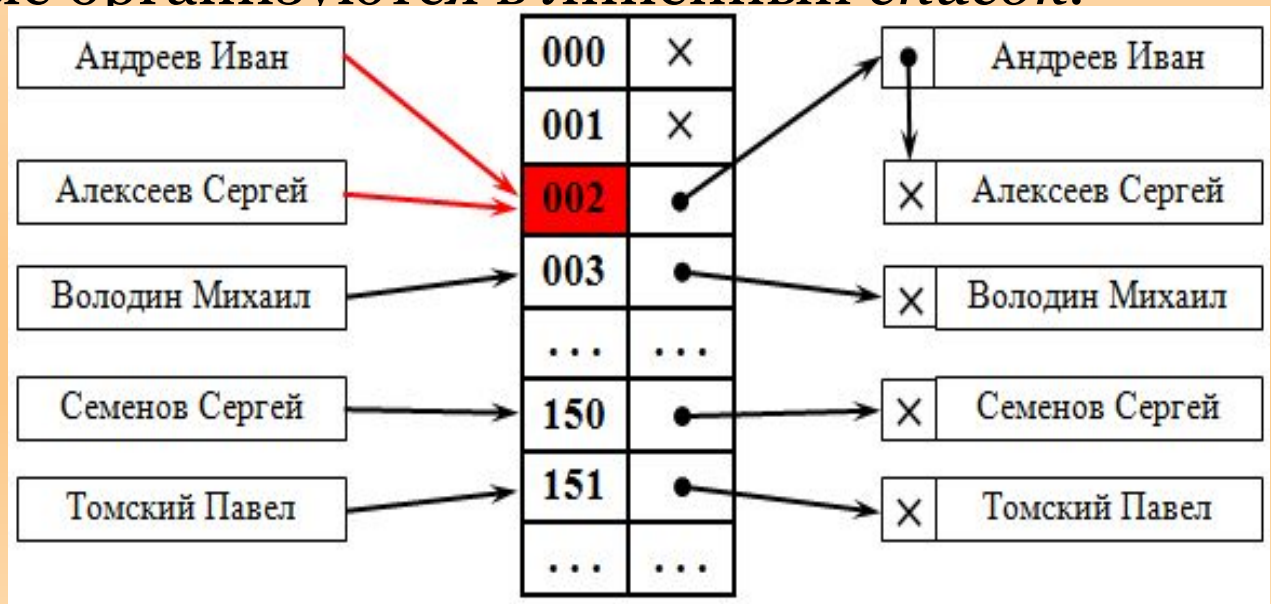
- *Коллизии* нарушают однозначность соответствия между хеш-кодами и данными.

- Существуют способы преодоления коллизий:

1. метод цепочек (внешнее или открытое *хеширование*);
2. метод открытой адресации (закрытое *хеширование*).

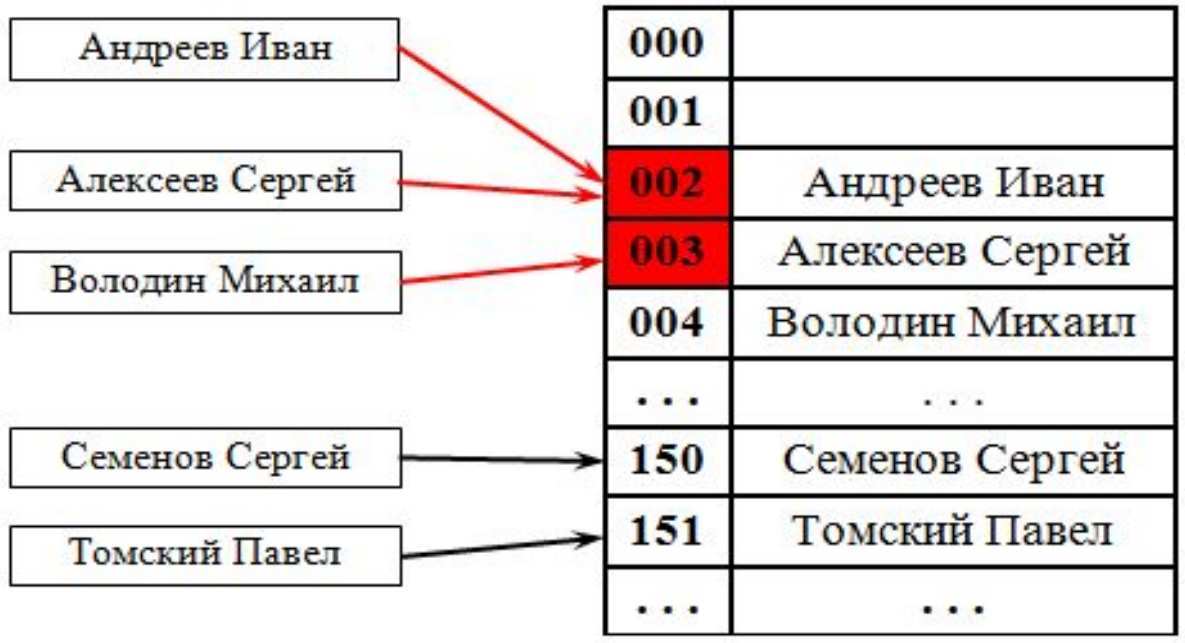
П.5.Разрешение коллизий.

- 1.Разрешение коллизий при помощи цепочек
- *элементы множества*, которым соответствует одно и то же хеш-*значение*, связываются в цепочку-*список*.
- В позиции номер i хранится *указатель* на *голову списка* тех элементов, у которых хеш-*значение* ключа равно i ; если таких элементов в множестве нет, в позиции i записан NULL.
- На рисунке - на *ключ* 002 претендуют два значения, которые организуются в линейный *список*.



2.Разрешение коллизий при помощи открытой адресации

- При открытой адресации все записи хранятся в самой хеш-таблице. Каждая *ячейка* таблицы содержит либо элемент динамического *множества*, либо NULL.



Задача 1. Первое и последнее вхождение

Дана строка. Если в этой строке буква f встречается только один раз, выведите её индекс. Если она встречается два и более раз, выведите индекс её первого и последнего появления. Если буква f в данной строке не встречается, ничего не выводите.

При решении задачи не использовать метод count
Входные данные
Вводится строка.
Выходные данные
Выведите ответ на задачу.
Примеры
входные данные
office
выходные данные
1 2

входные данные
comfort
выходные данные
3

Задача 2. Дублирование фрагмента

Дана строка, в которой буква h встречается как минимум два раза. Повторите последовательность символов, заключенную между первым и последнием появлением буквы h два раза, сами буквы h повторять не надо.

Задача 3. Замена подстроки

Дана строка. Замените в этой строке все цифры 5 на слово Five.

Банковские проценты

- Вклад в банке составляет X рублей. Ежегодно он увеличивается

  на q процентов, после чего дробная часть копеек отбрасывается. Определите, через сколько лет вклад составит не менее S рублей.

Задача ДЗ

- Написать программу простого поиска подстроки в строке.
- Можно использовать циклы, if

П.6.Алгоритм Рабина — Карпа

- это алгоритм поиска подстроки в строке, который ищет шаблон, то есть подстроку, в тексте, используя хэширование
- Алгоритм разработан в 1981-1987 гг. Рабином М. и Карпом Р.

Алгоритм Рабина-Карпа

Нужно определить, входит ли хотя бы одна из N строк S_i (каждая из них одинаковой длины L) в текст T (длины M).

1) Вычисляем хэш-функции от каждой строки S_i (например, контрольную сумму).

2) Перебираем в цикле все подстроки T длины L.

3) Для каждой такой подстроки вычисляем хэш-функцию.

4) Сравниваем значение хэш-функции с значениями хэш-функций всех строк S_i.

5) Только если есть совпадение хэш-функций, то тогда сравниваем эту подстроку T с той строкой S_i, для которой было совпадение.

Таким образом, экономится время (сложность **O(L*M/N)**

Алгоритм Рабина-Карпа

Пусть алфавит D={0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, то есть каждый символ в алфавите есть d–ичная цифра, где d=|D|.

Пример 2. Пусть образец имеет вид W = 3 1 4 1 5
Вычисляем значения чисел из окна длины |W|=5 по mod q,
q — простое число.

T	=	2	3	5	9	0	2	3	1	4	1	5	2	6	7	3	9	9	2	1
Значения по mod 13 =						8	9	3	11	0	1	7	8	4	5	10	11	7	9	11

T	=	2	3	5	9	0	2	3	1	4	1	5	2	6	7	3	9	9	2	1
Значения по mod 13 =						8	9	3	11	0	1	7	8	4	5	10	11	7	9	11

23590(mod 13)=8, 35902(mod 13)=9, 59023(mod 13)=9, …
k1=31415≡7(mod 13) – вхождение образца,
k2=67399≡7(mod 13) – холостое срабатывание.

Из равенства ki= kj (mod q) не следует, что ki= kj (например, 31415=67399(mod 13), но это не значит, что 31415=67399).
Если ki= kj (mod q), то надо проверять, совпадают ли строки W[1…m] и T[s+1…s+m] на самом деле.

Задача на циклы 1

Последовательность Фибоначчи определяется так:

$$\varphi_0 = 0, \varphi_1 = 1, \ldots, \varphi_n = \varphi_{n-1} + \varphi_{n-2}.$$

По данному числу n определите n-е число Фибоначчи φ_n.

Входные данные

Вводится натуральное число n.

Выходные данные

Выведите ответ на задачу.

Задача 2. Номер числа Фибоначчи

$$\varphi_0 = 0, \varphi_1 = 1, \ldots, \varphi_n = \varphi_{n-1} + \varphi_{n-2}.$$

Дано натуральное число A. Определите, каким по счету числом Фибоначчи оно является, то есть выведите такое число n, что
$\varphi_n = A$.
Если A не является числом Фибоначчи, выведите число -1.

Входные данные

Вводится натуральное число A.

Выходные данные

Выведите ответ на задачу.

Задача 3. Фрагменты.

- Дана последовательность натуральных чисел, завершающаяся числом 0.

Определите, какое наибольшее число подряд идущих элементов этой последовательности равны друг другу.

- **Входные данные**

Вводится последовательность целых чисел, оканчивающаяся числом 0 (само число 0 в последовательность не входит, а служит как признак ее окончания).

- **Выходные данные**

Выведите ответ на задачу.

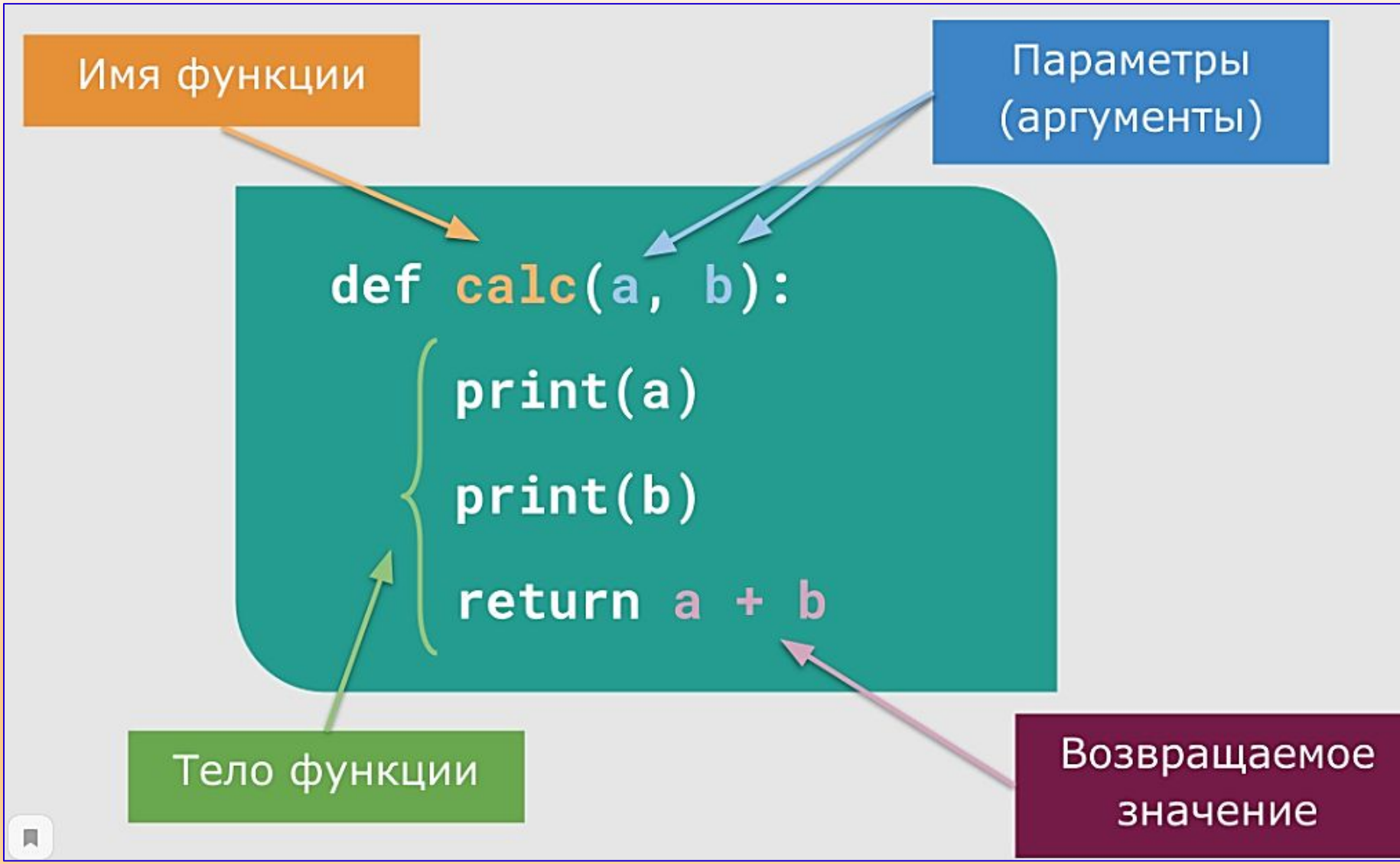
Задача 4. Монотонный фрагмент

- Максимальная длина монотонного фрагмента
- Дана последовательность натуральных чисел, завершающаяся число 0. Определите наибольшую длину монотонного фрагмента последовательности (то есть такого фрагмента, где все элементы либо больше предыдущего, либо меньше).
- **Входные данные**
- Вводится последовательность целых чисел, оканчивающаяся числом 0 (само число 0 в последовательность не входит, а служит как признак ее окончания).
- **Выходные данные**
- Выведите ответ на задачу.

Задача 5. Банковские проценты

Вклад в банке составляет x рублей. Ежегодно он увеличивается на p процентов, после чего дробная часть копеек отбрасывается.
Определите, через сколько лет вклад составит не менее y рублей.

Функция в Python


Имя функции
Параметры (аргументы)
def calc(a, b):
print(a)
print(b)
return a + b
Тело функции
Возвращаемое значение

ЗАДАЧА 1. Найти число сочетаний

По данным целым неотрицательным n и k вычислите значение числа сочетаний из n элементов по k, то есть $\frac{n!}{k!(n-k)!}$.

Входные данные

Вводятся числа *n* и *k*.

Выходные данные

Выведите ответ на задачу.

Вместо повторения в алгоритме выше
лучше сделать одну *функцию*,
вычисляющую факториал любого данного числа n
и 3-и раза использовать эту функцию в своей программе.

Соответствующая функция может выглядеть так:

```
def factorial(n):
    f = 1
    for i in range(2, n + 1):
        f *= i
    return f
```

Синтаксис функции в Python

```
def fib(n):
    if n <= 2:
        return 1
    return fib(n - 1) + fib(n - 2)


n1 = fib(1)  # = 1
n10 = fib(10)  # = 55
```

- Объявление функций начинается с ключевого слова **def**.
- Т.к. объявления типов в Python нет, то и аргументы функций объявляются просто именами.
- Значение из функции возвращается с помощью **return**.
- Функция может вызывать сама себя (рекурсия).

- Вызвать функцию можно либо просто передав аргументы позиционно, либо по их именам

Теперь мы можем использовать нашу функцию несколько раз.

В этом примере мы трижды вызываем функцию factorial для вычисления трех факториалов: factorial(n), factorial(k), factorial(n-k).

```
n = int(input())
k = int(input())
print factorial(n) // (factorial(k) * factorial(n - k))
```

Функция нахождения максимума из двух чисел

Итак – задача нахождения наибольшего из двух или трех чисел.

Функцию нахождения максимума из двух чисел можно написать так:

```
def max(a, b):
    if a > b:
        return a
    else:
        return b
```

!!! Внутри функции можно использовать переменные, объявленные вне этой функции

```
def f():
    print a
a = 1
f()
```

Здесь переменной a присваивается значение 1,
и функция f печатает это значение, несмотря на то,
что выше функции f эта переменная не инциализируется.
В момент вызова функции f переменной a
уже присвоено значение, поэтому функция f может
вывести его на экран.

Опр. переменные, объявленные вне функции,
но доступные внутри функции, называются *глобальными*.

Опр. Переменные, инициализированные внутри
функции, называются *локальными* и использовать эту
переменную вне функции нельзя.

Задача 1. Напишите функцию `min4(a, b, c, d)`, вычисляющую минимум четырех чисел, которая не содержит инструкции `if`, а использует **стандартную** функцию `min`.
Считайте четыре целых числа и выведите их минимум.

Входные данные

Вводятся четыре целых числа.

Выходные данные

Выведите ответ на задачу.

Задача 2. Дано действительное положительное число a и **целое** число n.
Вычислите a в степени n.
Решение оформите в виде функции `power(a, n)`.
Стандартной функцией или операцией возведения в степень пользоваться нельзя.

Входные данные

Вводится действительное положительное число a и **целое** число n.

Выходные данные

Выведите ответ на задачу.

- Задача 3.

Даны два натуральных числа n и m. Сократите дробь $\frac{n}{m}$, то есть выведите два других числа p и q таких, что $\frac{n}{m} = \frac{p}{q}$ и дробь $\frac{p}{q}$ — несократимая.

Решение оформите в виде функции `ReduceFraction(n, m)`, получающая значения `n` и `m` и возвращающей кортеж из двух чисел.

Входные данные

Вводятся два натуральных числа.

Выходные данные

Выведите ответ на задачу.

Задача 4. Минимальный делитель числа

Дано натуральное число $n > 1$. Выведите его наименьший делитель, отличный от 1.

Решение оформите в виде функции `MinDivisor(n)`. Алгоритм должен иметь сложность $O(\sqrt{n})$.

Указание. Если у числа n нет делителя не превосходящего $\sqrt{n}$, то число n — простое и ответом будет само число n.

Входные данные

Вводится натуральное число.

Выходные данные

Выведите ответ на задачу.

Лекция 3. Суффиксные деревья

П.9. Суффиксные деревья.

- Суффиксное дерево — это способ представления текста.
- **Опр.8. Бором называется префиксное дерево –** абстрактный тип данных – структура, позволяющая хранить ассоциативный массив, ключами которого являются строки.
- Значение ключа можно получить просмотром всех родительских узлов, каждый из которых хранит один из символов алфавита.
- Корень дерева связан с пустой строкой. То есть потомки узла имеют общий префикс.

- **Для сокращения количества вершин оптимизируем бор:**

1. рассмотрим цепочку вершин бора, такую что из каждой вершины исходит единственное ребро в следующую.
2. Сожмем такую цепочку в одно ребро, а вместо буквы напишем на нем всю последовательность букв с ребер, которые мы заменили.
3. Эта последовательность букв обязательно является подстрокой некоторой строки si из набора, поэтому запишем на ребре только номер строки, а также начало и конец соответствующей подстроки.

- **Так получим сжатый бор.**

- **Опр. 9. Сжатый бор** — это корневое дерево, на каждом

  ребре которого написана непустая строка, обладающее следующими свойствами:

1. Ни из какой вершины не выходит два ребра, строки для которых начинаются на одну букву.
2. Если вершина не является корнем или листом дерева, из нее выходит не менее двух ребер.

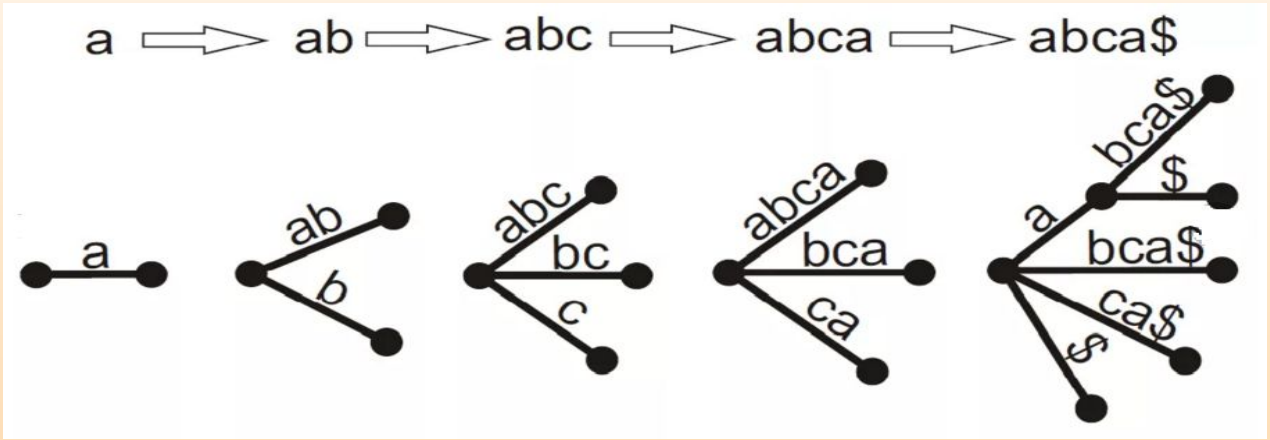
Утверждение. Количество вершин в сжатом боре составляет O(k), где k — количество строк в наборе.

Опр.10. Суффиксным деревом строки S называется сжатый бор, построенный на всех суффиксах S.

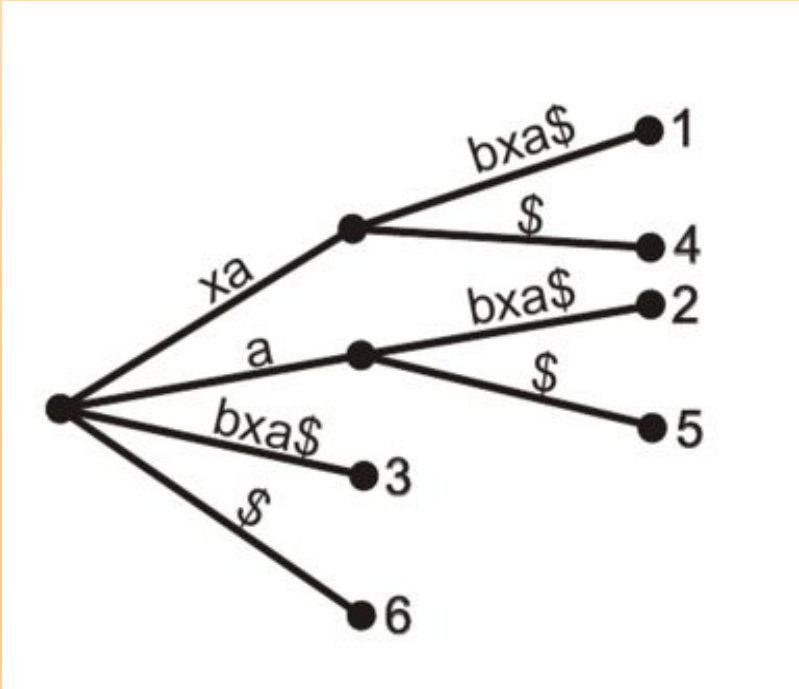
Опр. 11. Суффиксное дерево для m-символьной строки S — это ориентированное дерево с корнем, имеющее ровно m листьев, занумерованных от 1 до m. Каждая внутренняя вершина, отличная от корня, имеет не меньше двух детей, а каждая дуга помечена непустой подстрокой строки S (дуговой меткой). Никакие две дуги, выходящие из одной и той же вершины, не могут иметь пометок, начинающихся с одного и того же символа.

- Пример 1. Рассмотрим строку "алгоритм", у нее есть следующие суффиксы:
- Номер Суффикс
- 0 "алгоритм"
- 1 "лгоритм"
- 2 "горитм"
- 3 "оритм"
- 4 "ритм"
- 5 "итм"
- 6 "тм"
- 7 "м"
- Первым суффиксом строки называется эта строка без первого символа.

Пример 2.
Суффиксное дерево строки **abca**



Пример 3.
Суффиксное дерево строки **xabxa**



Основные задачи, решаемые с использованием суффиксных деревьев и суффиксных массивов:

1. Определение количества различных подстрок строки s.
2. Нахождение максимальной по длине строки, имеющей два непересекающихся вхождения
3. Нахождение наибольшей общей подстроки нескольких строк

Продолжение. Задачи, решаемые с помощью суффиксных деревьев:

4.Определение длины наибольшего общего префикса для двух подстрок строки

5.Лексикографическое упорядочивание суффиксов строки s.

6.Определение рефренов:

Опр. 12. Рефреном строки s называется подстрока q, для которой произведение |q| на количество её вхождений максимально. Вхождения при этом могут пересекаться.

7. Нахождение подпалиндромов.

П. 10. Алгоритм Лемпеля-Зива – алгоритм сжатия данных.

В 1977 году Абрахам Лемпель и Якоб Зив предложили *алгоритм* сжатия данных, названный позднее **LZ77.**

Этот *алгоритм* используется в программах архивирования текстов **pkzip** и **arj и др.**

Суть алгоритма - когда ранее прочитанная подстрока встречается повторно, алгоритм пишет вместо неё ссылку на её предыдущее вхождение.

Алгорим Лемпеля-Зива

На каждом шаге выводится тройка:
(d-смещение относительно текущей позиции,
L-длина подстроки, символ a).

В конкретных реализациях может быть: (d; L; a), либо (d, l) и a.

Пример 1. Строка w = abaababa,

её код — (0, 0, a)(0, 0, b)(2, 1, a)(3, 2, b)(0, 0, a).

Или так: ab(2, 1)(3, 3)(2, 2).

Рассмотрим реализацию 2:

Если в тексте встретится повторение строк символов, причем запись о длине данной строки и смещении от текущей позиции короче, чем сама последовательность, то в выходной файл записывается ссылка (Смещение. Длина), а не сама строка.

Пример 2: Строка "КОЛОКОЛ_ОКОЛО_КОЛОКОЛЬНИ" с помощью алгоритма LZ77 будет закодирована строкой "КОЛО(-4,3)_(-5,4)О_(-14,7)ЬНИ".

Пример 3: Строка "АААААААА" с помощью алгоритма LZ77 будет закодирована "А(-1,6)".

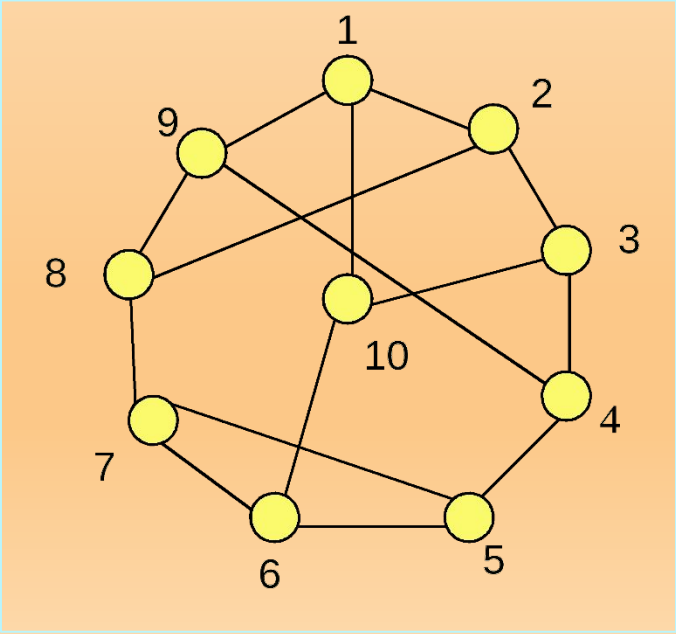
Глава 3. Теория графов и использование графов в алгоритмах.

П.1. основные термины теории графов.

Опр. 1 . Графом называется математический объект, представляющий собой множество *вершин* графа и набор *рёбер*, то есть соединений между парами вершин.

Опр.2.Если некоторое ребро u соединяет две вершины A и B графа, то говорят, что ребро u **инцидентно** вершинам A и B, а вершины в свою очередь **инцидентны** ребру u.

Опр. 3. Вершины, соединенные ребром, также называют **смежными**.

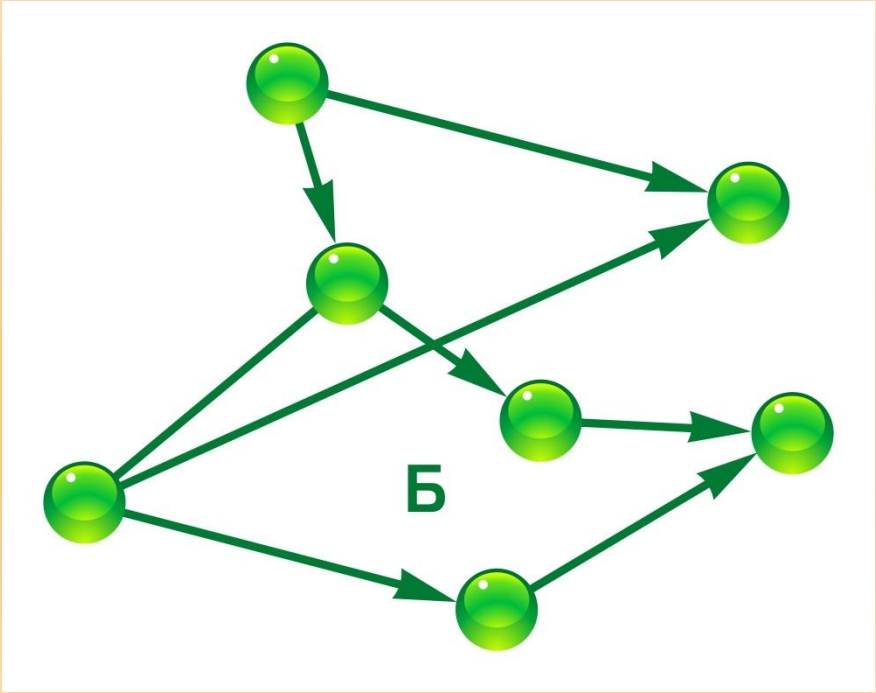


Опр. 4. Два ребра называются **кратными**, если они соединяют одну и ту же пару вершин.

Опр.5.Ребро называется **петлей**, если его концы совпадают.

Опр.6.Степенью вершины называют количество ребер, для которых она является концевой (при этом петли считают дважды).

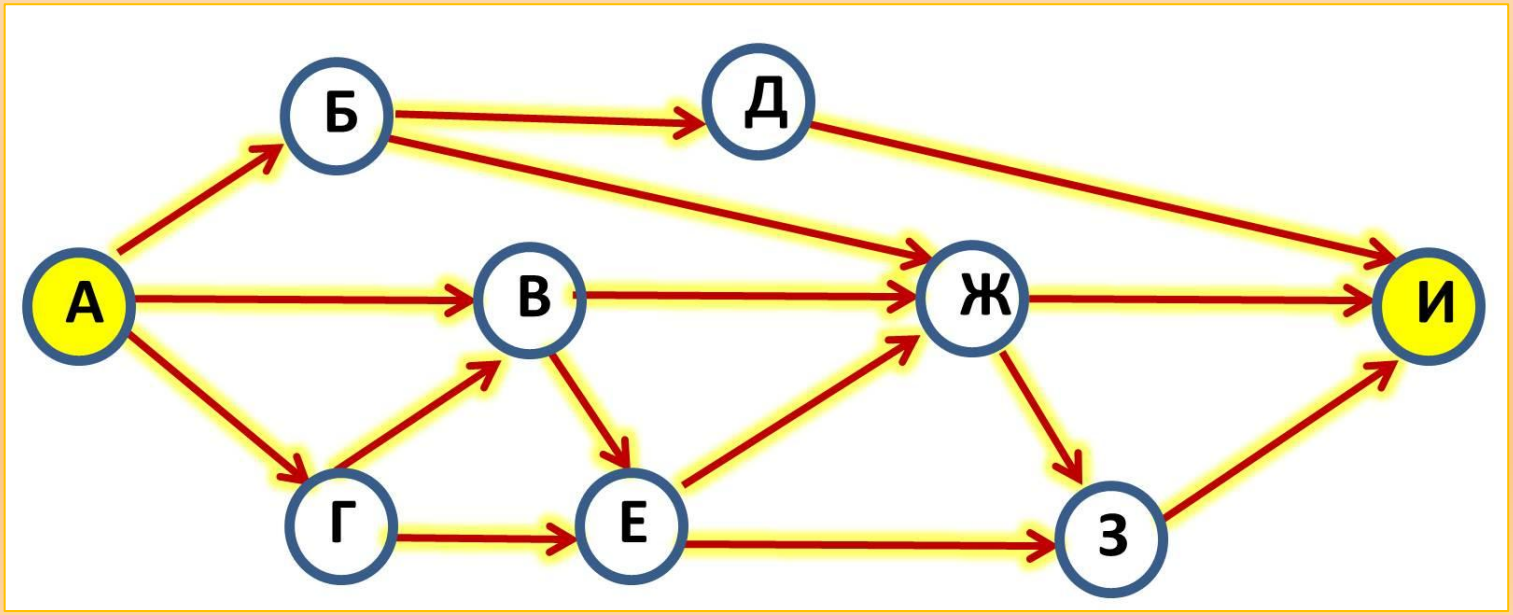
Опр.7.Ориентированный граф – граф, у ребер которого одна вершина считается начальной, а другая – конечной.



Опр. 8. Маршрутом в графе называют конечную последовательность вершин, в которой каждая вершина (кроме последней) соединена со следующей в последовательности вершиной ребром.

Опр.9.Цепью называется маршрут без повторяющихся рёбер.

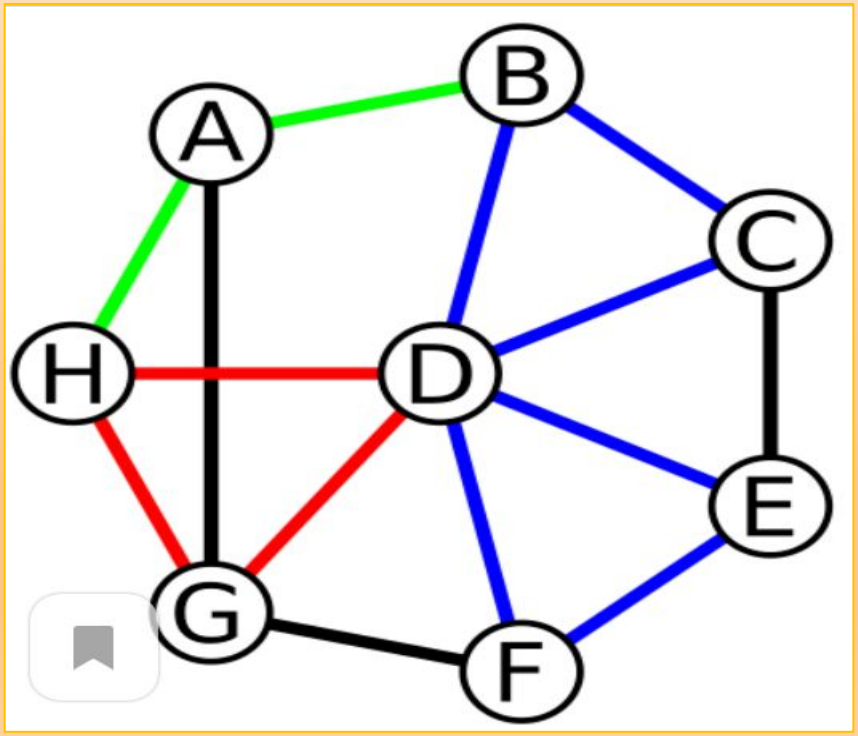
Опр.10.Простой цепью называется маршрут без повторяющихся вершин (откуда следует, что в простой цепи нет повторяющихся рёбер).



Опр.11.Ориентированным маршрутом (или **путём**) в орграфе называют конечную последовательность вершин и дуг, в которой каждый элемент инцидентен предыдущему и последующему.

Опр.12.Циклом называют цепь, в которой первая и последняя вершины совпадают.

Опр.13.При этом **длиной** пути (или цикла) называют число составляющих его *рёбер*.



- В теории графов два типа объектов называются **циклами**:
- 1). **замкнутый обход**, состоит из последовательности вершин, начинающейся и заканчивающейся в той же самой вершине, и каждые две последовательные вершины в последовательности смежны.
- 2 **простой цикл** — это замкнутый обход без повторного прохода по ребру или посещения вершины дважды, за исключением начальной и конечной вершин.

!!! Простые циклы можно описать набором рёбер, в отличие от замкнутых обходов, в которых наборы рёбер (с возможным повторением) не определяют однозначно порядок вершин.

Опр.14. Степенью вершины в неориентированном графе называется число инцидентных данной
вершине ребер (при этом петля считается два раза, то есть степень - это количество «концов» ребер, входящих в вершину).

Опр.15. Граф, в котором любые две вершины соединены одним ребром, называется полным **графом**

Свойства графов

А. В любом графе число вершин нечетной степени – четно. Этот факт называется **«леммой о рукопожатиях»** – в любой компании число людей, сделавших нечетное число рукопожатий всегда четно.

Б. Сумма степеней всех вершин равна удвоенному числу ребер в графе.

С. Максимальное число ребер в простом графе с n- вершинами есть n(n–1)/2.

П.2. Способы представления графов в памяти.

Представление графов в памяти – это способ хранения информации о ребрах графа, позволяющий решать следующие задачи:

1. Для двух данных вершин u и b проверить, соединены ли вершины u и v ребром.
2. Перебрать все ребра, исходящие из данной вершины u .

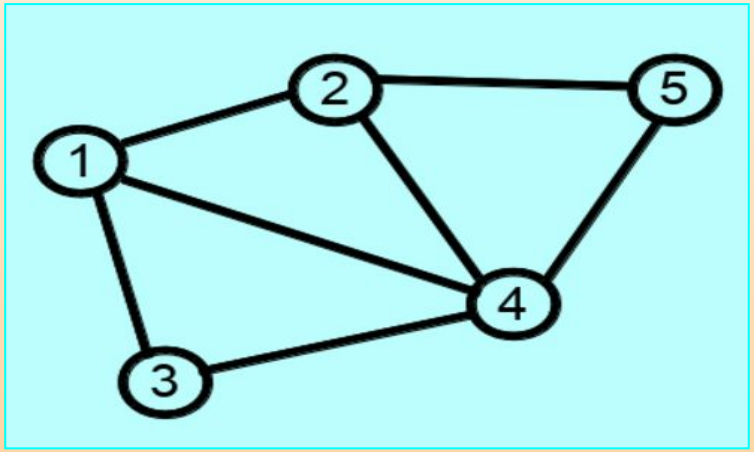
- Способы представления графов в памяти:

1. Матрица смежности
2. Списки смежных вершин
3. Список ребер

1.Матрица смежности

При представлении графа **матрицей смежности** информация о ребрах графа хранится в квадратной матрице, где:

- элемент A[i][j] равен1, если ребра i и j соединены ребром
- равен 0 в противном случае.

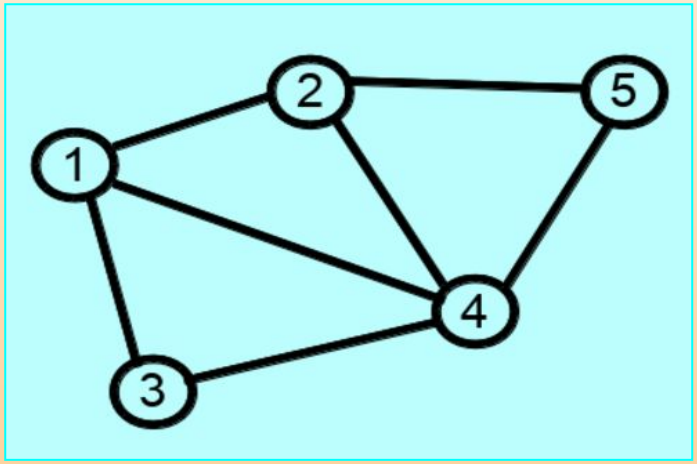


	1	2	3	4	5
1	0	1	1	1	0
2	1	0	0	1	1
3	1	0	0	1	0
4	1	1	1	0	1
5	0	1	0	1	0

Если граф неориентированный, то матрица смежности всегда симметрична относительно главной диагонали.

2. Хранение графа в виде списка ребер.

- При представлении графа **списком ребер** для каждого ребра графа хранится пара номеров вершин, которые оно соединяет.



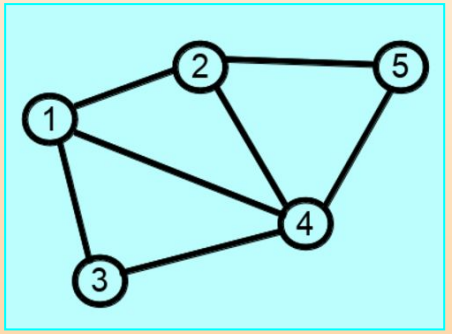
- **Для ориентированных графов** первое число в паре соответствует начальной вершине ребра, а второе — конечной.

[1,2]
[1,3]
[1,4]
[2,5]
[4,5]
[2,4]
[3,4]

3. Представление графа списками смежности

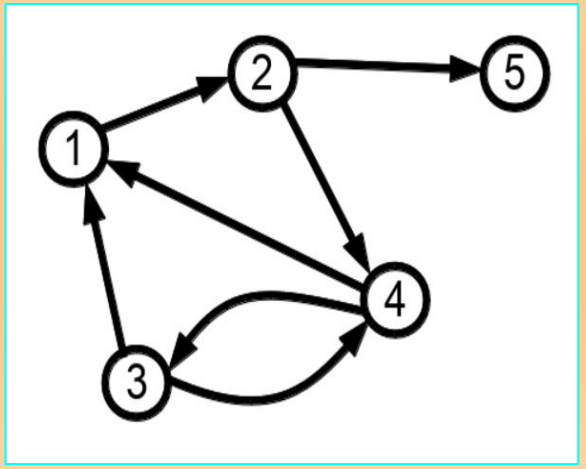
1. Для каждой вершины i хранится список W[i] смежных с ней вершин.

W[1] = [2, 3, 4]
W[2] = [1, 4, 5]
W[3] = [1, 4]
W[4] = [1, 2, 3, 5]
W[5] = [2, 4]



2.При помощи списков смежности можно представлять и ориентированные графы.

списки смежности будут следующими:
W[1] = [2]
W[2] = [4, 5]
W[3] = [1, 4]
W[4] = [1, 3]
W[5] = []



П.3.Алгоритмы поиска в графе.

Опр.16. Вершина, которая еще не посещена, называется новой.

- После посещения, вершина становится открытой и остается открытой пока не посещены все ее инциндентные ребра.
- После того, как будут исследованы все инциндентные ребра открытой вершины, вершина становится закрытой.

- **Опр.17. Взвешенным графом** называется граф, у которого любая пара вершин (ребро) снабжено числом, характеризующим отношение этих вершин (расстояние пунктами теснота связи объектов)

- **Опр. 18. Кратчайшим путем** называется путь в графе, содержащий наименьшее количество ребер.

- **Опр. 19. Связный граф** - это граф, у которого для любых двух различных вершин существует путь (последовательность смежных вершин), соединяющий их.
- Критерии связности Теорема Менгера

1.Алгоритм поиска в глубину.

Алгоритм поиска (или обхода) в глубину реализует обход ориентированного или неориентированного графа, при котором посещаются все вершины, доступные из начальной вершины.

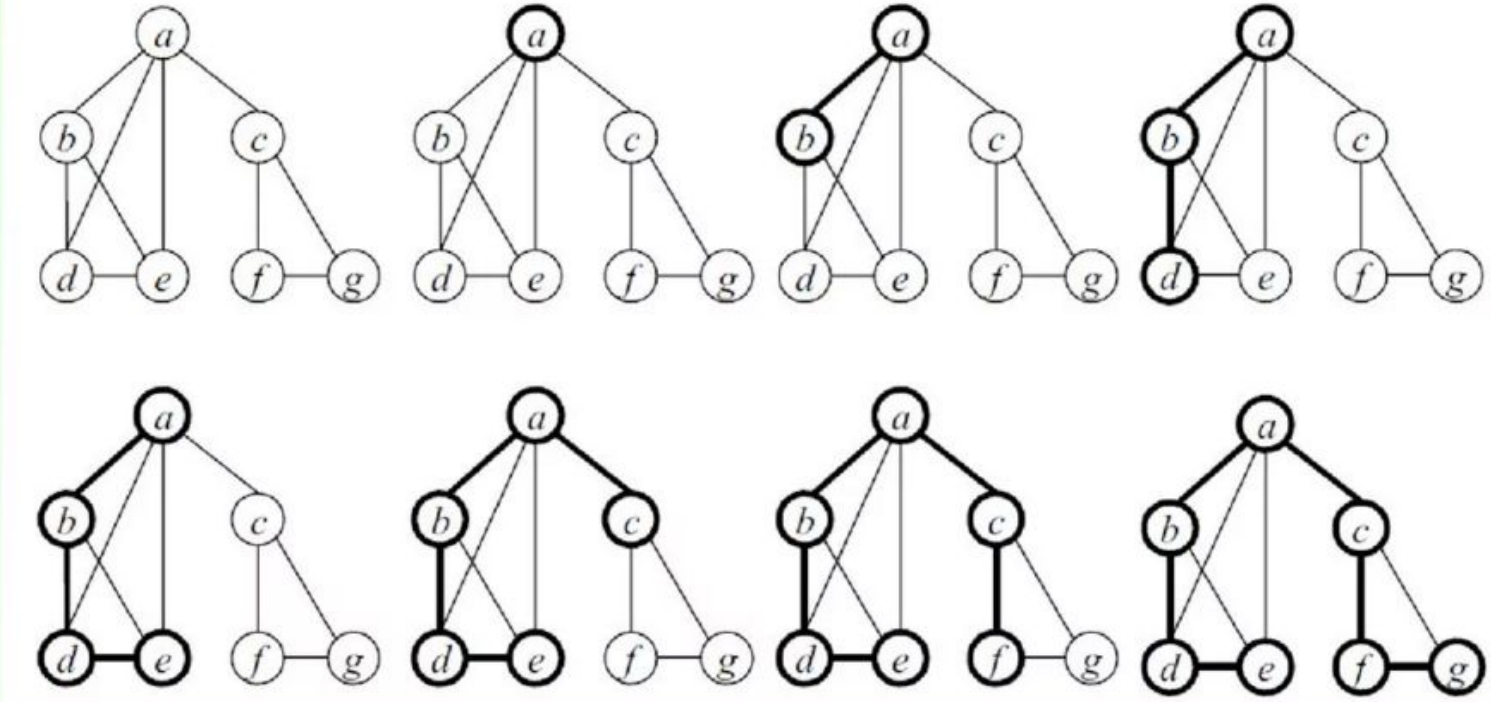
1. Из активной вершины пойти в какую-нибудь смежную вершину, не посещенную ранее.
2. Запустить из этой вершины алгоритм обхода в глубину
3. Вернуться в начальную вершину.
4. Повторить пункты 1-3 для всех не посещенных ранее смежных вершин.

Алгоритм является рекурсивным — для обхода всего графа нужно переместиться в соседнюю вершину, после чего повторить для этой вершины алгоритм обхода.

-Недостаток поиска в глубину - не находит кратчайших путей.

+Преимущество - применим в ситуациях, когда граф неизвестен целиком.

Поиск в глубину (порядок обхода)



Поиск в глубину для полного обхода графа с *n* вершинами и *m* дугами требует общего времени порядка *O(max(n, m))*. Поскольку обычно *m≥n*, то получается *O(m)*. ***Используется стек.***

Задачи, решаемые поиском в глубину:

1. **Поиск элементов**
2. **Поиск компонент связности**
3. **Поиск циклов и др.**

Оценка времени работы алгоритма:

- Если граф задан списками смежности , то время работы алгоритма можно оценить как O(n+m), где m- число ребер, а n-число вершин.
- Если граф задан матрицей смежности – то $O(n^2)$

Лекция и семинар 7.

3.2. Поиск в ширину в графе.

Идея *поиска в ширину* состоит в том, чтобы посещать вершины в порядке их удаленности от некоторой заранее выбранной или указанной стартовой вершины **a**.

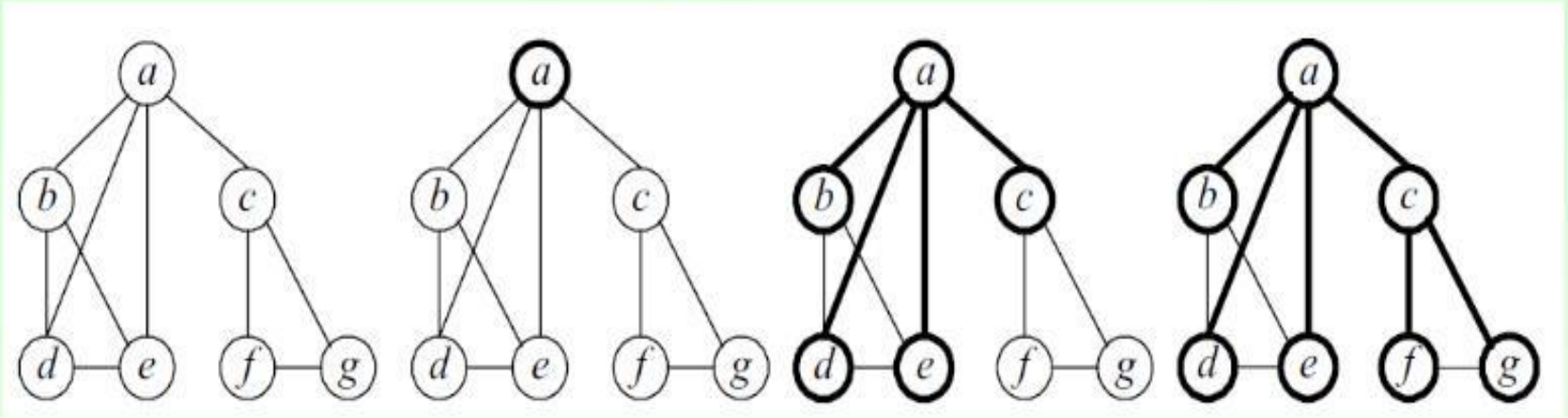
То есть сначала посещается сама *вершина* **a**, затем все вершины, смежные с **a**, то есть находящиеся от нее на расстоянии 1 , затем вершины, находящиеся от **a** на расстоянии 2, и т.д.

Главное свойство *поиска в ширину*:

чем ближе *вершина* к старту, тем раньше она будет посещена.

Для хранения *множества* открытых вершин *очередь* - когда новая *вершина* становится открытой, она добавляется в конец очереди, а активная выбирается в ее начале.

Поиск в ширину (волновой алгоритм)



Оценка времени работы алгоритма поиска в ширину:

Если граф задан списками смежности , то время работы алгоритма можно оценить как O(n+m), где m- число ребер, а n-число вершин.

Если граф задан матрицей смежности – то $O(n^2)$

Алгоритм поиска в ширину:

1. Для каждой вершины в массиве d будем хранить кратчайшее расстояние до этой вершины, если же расстояние неизвестно — будем хранить значение -1 или None (в языке Python). В самом начале расстояние до всех вершин равно -1 (None), кроме начальной вершины, до которой расстояние равно 0.

2. Затем перебираем все вершины, до которых расстояние равно 0, перебираем смежные с ними вершины и для них записываем расстояние равное 1.

3. Затем перебираем все вершины, до которых расстояние равно 1, перебираем их соседей, записываем для них расстояние, равное 2 (если оно до этого было равно -1 (None)).

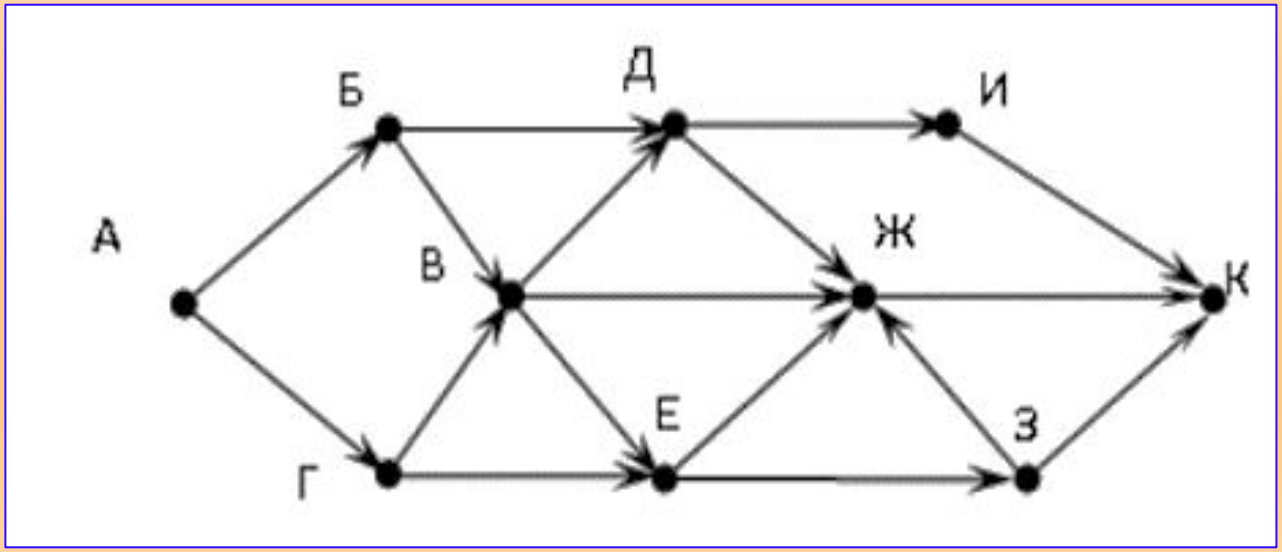
4. Затем перебираем вершины, до которых расстояние было равно 2 и тем самым определяем вершины, до которых расстояние равно 3 и т. д.

5. Этот цикл можно повторять либо пока обнаруживаются новые вершины на очередном шаге, либо n−1 раз (где n – число вершин в графе), так как длина кратчайшего пути в графе не может превосходить n−1.

П.4. Задача 1. Вычисление Количества путей в графе

Пусть есть схема дорог, связывающих города А, Б, В, Г, Д, Е, Ж, И, К. По каждой дороге можно двигаться только в одном направлении, указанном стрелкой.

Сколько существует различных путей из города А в город К?



Рекуррентная формула:

Если в город Z можно приехать только из городов B, C, D, то число различных путей из города A в город Z

равно сумме числа различных путей проезда:

из A в B, из A в C , из A в D, то есть

NZ=NB+NC+ND,

где NQ обозначает число путей из вершины A в некоторую вершину Q.

Алгоритм обхода графа в ширину.

Решение Задачи 1

Начнем считать количество путей с конца маршрута – с города К. N_X — количество различных путей из города А в город N — общее число путей.

В "К" можно приехать из И, Ж, или З, поэтому $N = N_К = N_И + N_Ж + N_З$ (1)

Аналогично:

$N_И = N_Д$;
$N_Ж = N_Д + N_В + N_Е + N_З$;
$N_З = N_Е$.

Добавим еще вершины:

$N_Д = N_Б + N_В = 1 + 2 = 3$;
$N_В = N_Б + N_Г = 1 + 1 = 2$;
$N_Е = N_Г + N_В = 1 + 2 = 3$;

$N_Г = N_А = 1$;
$N_Б = N_А = 1$.

Преобразуем первые вершины с учетом значений вторых:

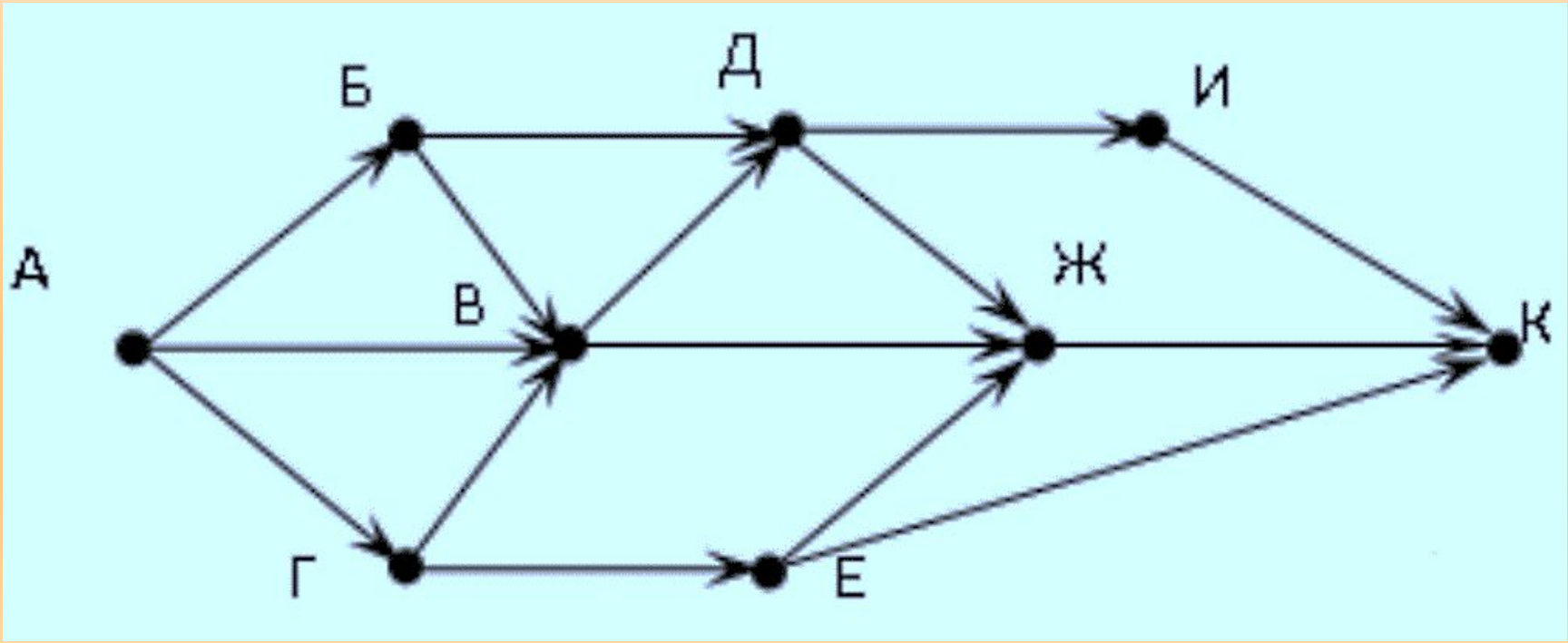
$N_И = N_Д = 3$;
$N_Ж = N_Д + N_В + N_Е + N_З = 3 + 2 + 3 + 3 = 11$;
$N_З = N_Е = 3$.

Подставим в формулу (1):

$N = N_К = 3 + 11 + 3 = 17$.

Задача 2.

- На рисунке — схема дорог, связывающих города А, Б, В, Г, Д, Е, Ж, И, К. По каждой дороге можно двигаться только в одном направлении, указанном стрелкой. Сколько существует различных путей из города А в город К?



- **Решение:** Начнем считать количество путей с конца маршрута – с города К. N_X — количество различных путей из города А в город X, N — общее число путей.
- В "К" можно приехать из И, Ж, или Е, поэтому $N = N_К = N_И + N_Ж + N_Е$ (1)

$N_И = N_Д$;
$N_Ж = N_Д + N_В + N_Е$;
$N_Е = N_Г$.

Добавим еще вершины:

$N_Д = N_Б + N_В = 1 + N_В = 1 + 3 = 4$;
$N_В = N_Б + N_А + N_Г = 1 + 1 + 1 = 3$;
$N_Г = N_А = 1$;

$N_Б = N_А = 1$.

Преобразуем первые вершины с учетом значений вторых:

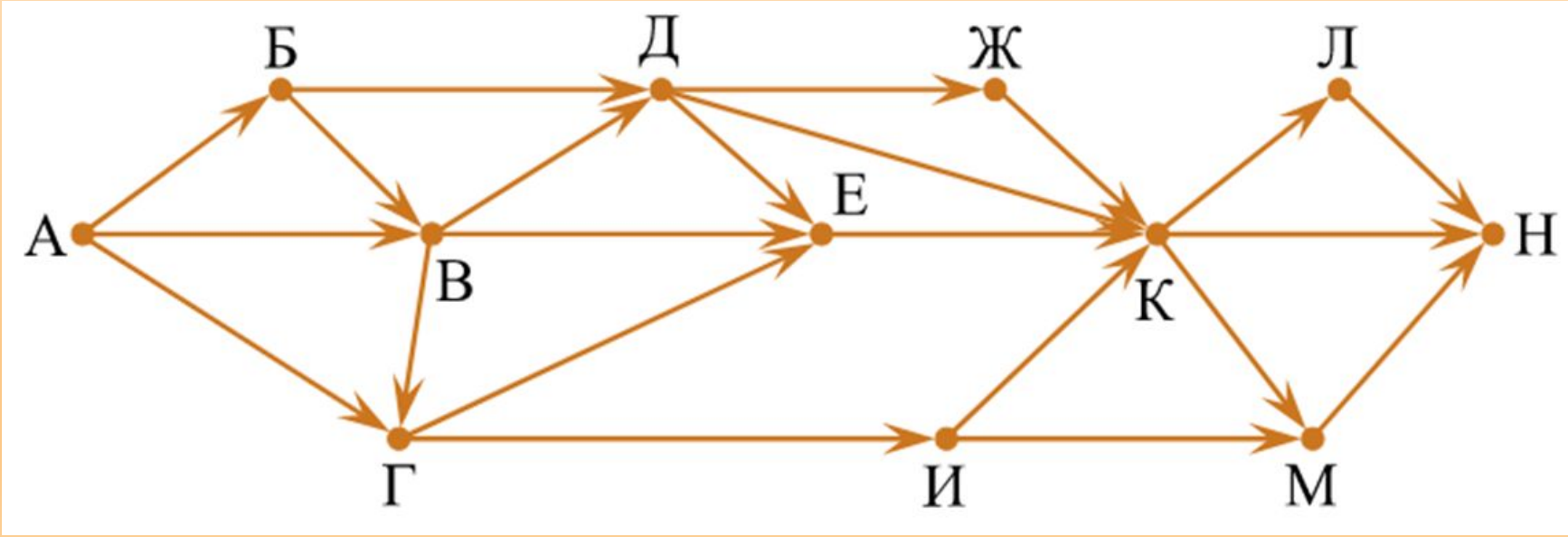
$N_И = N_Д = 4$;
$N_Ж = N_Д + N_В + N_Е = 4 + 3 + 1 = 8$;
$N_Е = N_Г = 1$.

Подставим в формулу (1):

$N = N_К = 4 + 8 + 1 = 13$.

Задачи на вычисление путей в графе с избегаемыми вершинами

Задача 3. На рисунке — схема дорог, связывающих пункты А, Б, В, Г, Д, Е, Ж, И, К, Л, М, Н. Сколько существует различных путей из пункта А в пункт Н, не проходящих через пункт В?



Решение задачи 3:количество путей до города
X = количество путей в любой из тех городов, из которых есть путь в X.

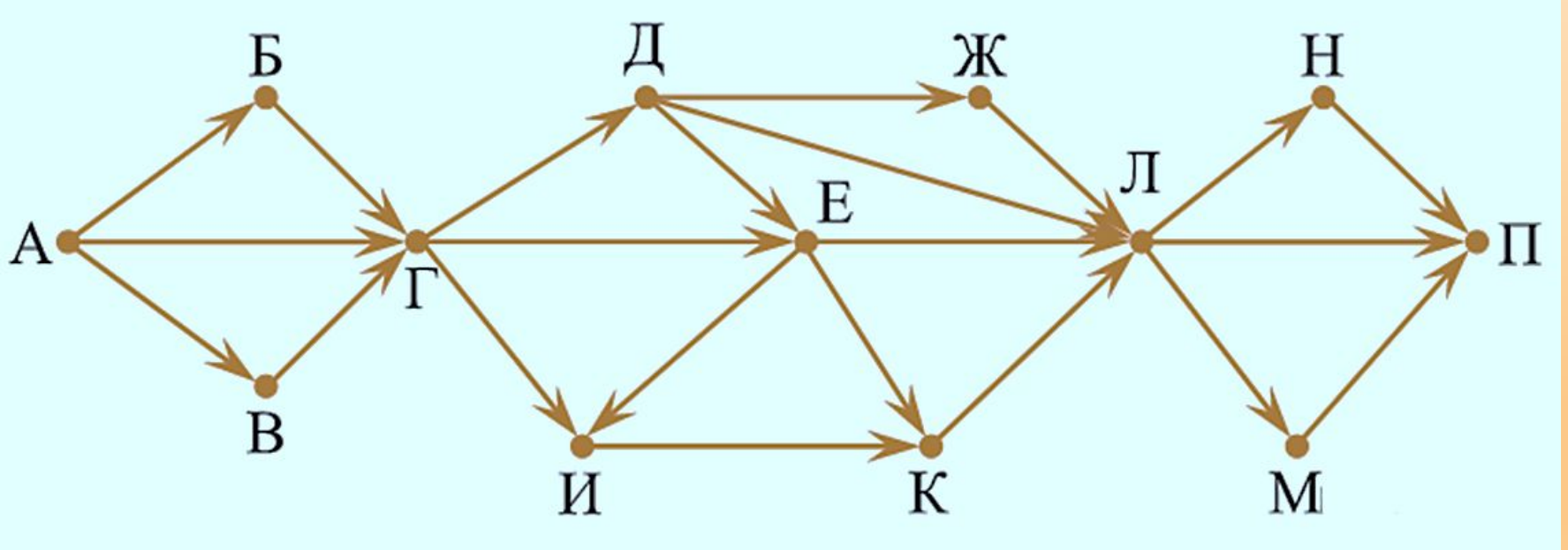
Если путь не должен проходить через вершину, нужно не учитывать эту вершину (город) при подсчёте сумм.

Если вершина обязательна для посещения - тогда для городов, в которые из нужного города идут дороги, в суммах нужно брать только этот город.

Рассчитаем последовательно количество путей до каждого из городов:

- А = 1.
- Б = А = 1.
- В = А + Б = 2.
- Г = А = 1 (В не учитываем, поскольку путь не должен проходить через город В).
- Д = Б = 1 (В не учитываем, поскольку путь не должен проходить через город В).
- Е = Г + Д = 2 (В не учитываем, поскольку путь не должен проходить через город В).
- Ж = Д = 1.
- И = Г = 1.
- К = Д + Ж + И + Е = 5.
- Л = К = 5.
- М = К + И = 6.
- Н = К + Л + М = 16.

- Задача 4. На рисунке – схема дорог, связывающих пункты А, Б, В, Г, Д, Е, Ж, И, К, Л, М, Н, П. Сколько существует различных путей из пункта А в пункт П, не проходящих через пункт Е?



- Решение:

- А = 1.
- Б = А = 1.
- В = А = 1.
- Г = А + Б + В = 3.
- Д = Г = 3.
- И = Г = 3 (Е не учитываем, поскольку путь не должен проходить через город Е).
- Ж = Д = 3.
- К = И = 3 (Е не учитываем, поскольку путь не должен проходить через город Е).
- Л = Д + Ж + К = 3 + 3 + 3 = 9 (Е не учитываем, поскольку путь не должен проходить через город Е).
- Н = Л = 9.
- М = Л = 9.
- П = Н + Л + М = 9 + 9 + 9 = 27.
-
- Ответ: 27.

П.5. Эйлеров цикл.

Опр. 19. Эйлеровым циклом в графе называется цикл, содержащий все ребра графа.

Опр. 20. Связный граф называется эйлеровым, если существует замкнутая цепь, проходящая через каждое его ребро.

Такая цепь называется **эйлеровой цепью**.

Требуется, чтобы каждое ребро проходилось только один раз.

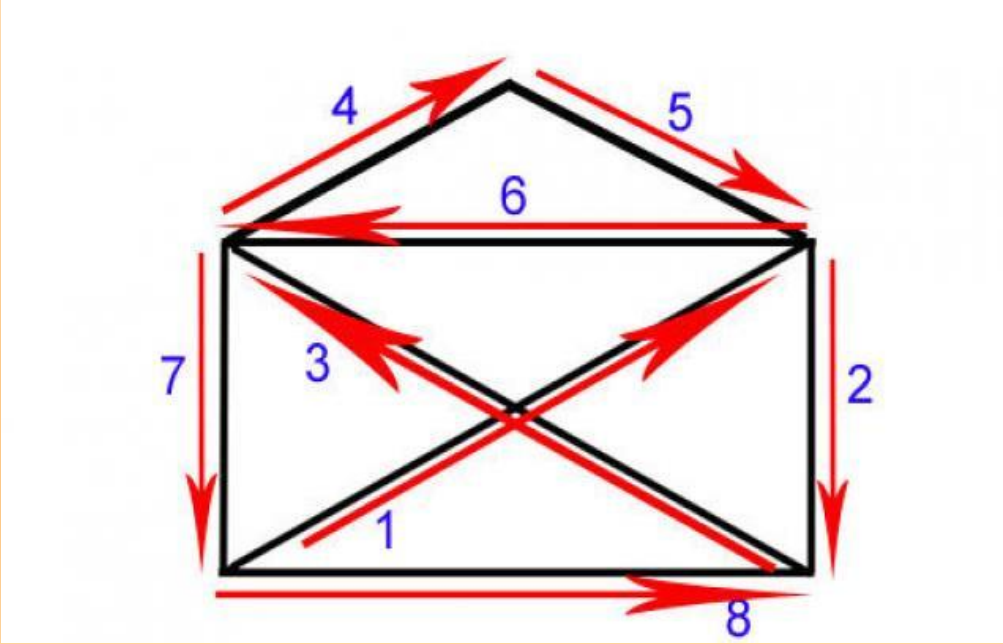
Опр.21. Эйлеров путь в графе — это путь, проходящий по всем рёбрам графа и притом только по одному разу.

Опр.21. Полуэйлеров граф — граф, содержащий эйлеров путь,

Утв.1. каждый эйлеров граф будет полуэйлеровым.

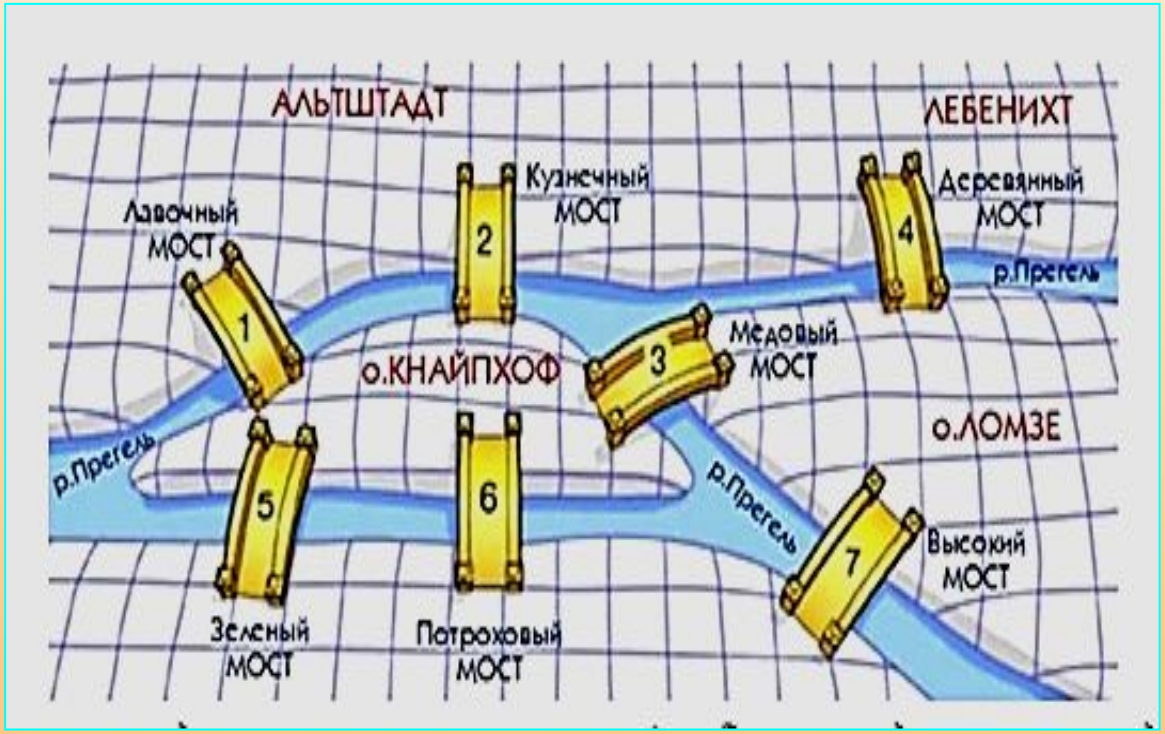
- **Опр.22.** Вершина называется четной, если ее степень четная.
- **Опр.23.** Вершина называется нечетной, если ее степень нечетная.

- Эйлеров цикл в графе означает, что следуя вдоль цикла, можно нарисовать граф на бумаге, не отрывая карандаша.



П.6.Задача 2. Задача о Кенигсбергских мостах:

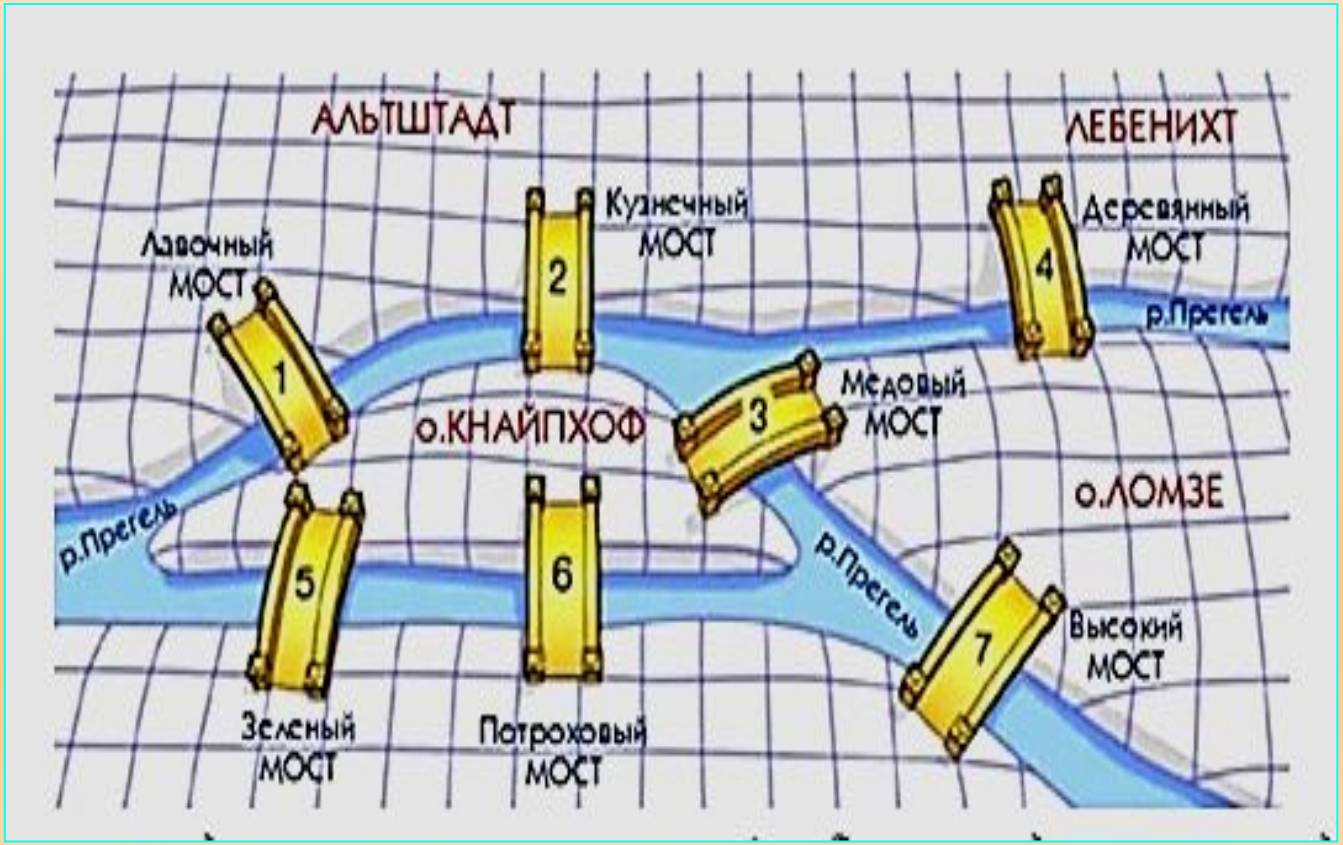
У жителей Кенигсберга была загадка: **как пройти по всем городским мостам через реку Преголя, не проходя ни по одному из них дважды.**



В 1736 году фон Эйлер решил Задачу о семи мостах.

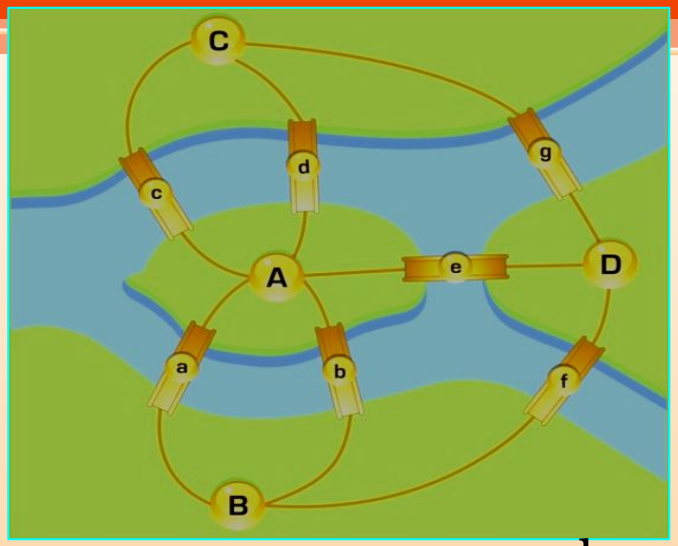
Решение Эйлера:

Нельзя пройти по всем городским мостам через реку Преголя, не проходя ни по одному из них дважды.



Решение задачи Эйлера:

- На схеме города - графе мостам соответствуют линии - ребра графа, а частям города — вершины графа.
- **В ходе рассуждений Эйлер пришёл к следующим выводам:**



1. Число нечётных вершин графа должно быть чётно.

2. Если все вершины графа чётные, то можно начертить этот граф без отрыва карандаша от бумаги, при этом можно начинать с любой вершины графа и завершить его в той же вершине.

3. Если ровно две вершины графа нечётные, то можно начертить этот граф без отрыва карандаша от бумаги, при этом нужно начинать с одной из нечётных вершин и завершить его в другой нечётной вершине.

4. Граф с более чем двумя нечётными вершинами невозможно начертить одним росчерком.

Граф кёнигсбергских мостов имел четыре нечётные вершины — следовательно, невозможно пройти по всем мостам, не проходя ни по одному из них дважды.

Теорема 1. Если *граф* обладает эйлеровым циклом, то он является связным, а все его вершины — четными.

Теорема 2. Если *граф* связный и все его вершины четные, то он обладает эйлеровым циклом.

Если *граф* не обладает эйлеровым циклом, то можно поставить задачу об отыскании одного *эйлерова пути* или нескольких *эйлеровых путей*, содержащих все ребра графа.

Теорема 3. Если *граф* G обладает *эйлеровым путем* с концами V_A и V_B и V_A не совпадает с V_B, то *граф* G является связным, и V_A и V_B — единственные нечетные его вершины.

Теорема 4. Если *граф* G связный и V_A и V_B единственные нечетные вершины его, то *граф* обладает *эйлеровым путем* с концами V_A и V_B

Теорема 5. Если *связный граф* G имеет 2k нечетных вершин, то найдется семейство из k путей, которые в совокупности содержат все ребра графа в точности *по* одному разу.

Контрольная

Глава 3. Математическое программирование –

– это раздел прикладной математики, посвященный теории и методам решения конечномерных задач поиска экстремумов функций на множествах, определяемых линейными и нелинейными ограничениями.

Оптимизационные модели применяются для решения прикладных задач в случаях, когда необходимо найти НАИЛУЧШЕЕ, в каком-то смысле, решение.

Постановка задачи оптимизации в общей форме:

(1)

$$F(X,Q) \to \max \ (\min)$$

Где X- вектор внутренних параметров объекта;
Q- вектор параметров воздействия факторов внешней среды.
F- целевая вектор-функция оптимизации.

Определение 1. Оптимальным решением общей задачи оптимизации (1) называется допустимый вектор X*, обеспечивающий максимальное (минимальное) значение целевой функции F(X,Q).

пl. Динамическое программирование

Сущность метода Динамического программирования - рассмотрим операцию **О**, состоящую из ряда последовательных m-этапов (шагов);

пусть итоговый выигрыш операции **О** (или итоговая эффективность операции **О**) - функция **Z** складывается из выигрышей на отдельных шагах, **то есть Z-** аддитивный критерий.

Операция **О** – является управляемым процесс, то есть мы можем выбирать параметры управления.

От решения, выбираемого на каждом шаге зависит и выигрыш и на данном шаге и итоговый выигрыш операции в целом.

Решения на каждом шаге называются шаговыми.

Совокупность всех шаговых управлений является управлением операцией в целом. Обозначим управление - X: **X(x_1, x_2, … , x_n),** **где** компонентами вектора X являются шаговые управления.

Тогда, постановка задачи динамического программирования будет следующей:

требуется найти такое оптимальное управление $X^*=X^*(x_1^*, x_2^*, \dots, x_m^*)$, при котором выигрыш Z обеспечивается максимальное значение критерия эффективности (выигрыша) Z:

$$Z = \sum_{i=1}^{m} Z_i \rightarrow \max$$

Таким образом, управление x_i должно обеспечить максимальный выигрыш не на данном конкретном шаге, а на всей совокупности шагов, входящих в операцию.

Для задачи можно использовать методы динамического программирования, если она удовлетворяет следующим требованиям:

1. Задача оптимизации интерпретируется как n-шаговый процесс управления.
2. Целевая функция равна сумме целевых функций каждого шага (аддитивна).

В основе решения всех задач динамического программирования лежат принципы оптимальности и вложенности:

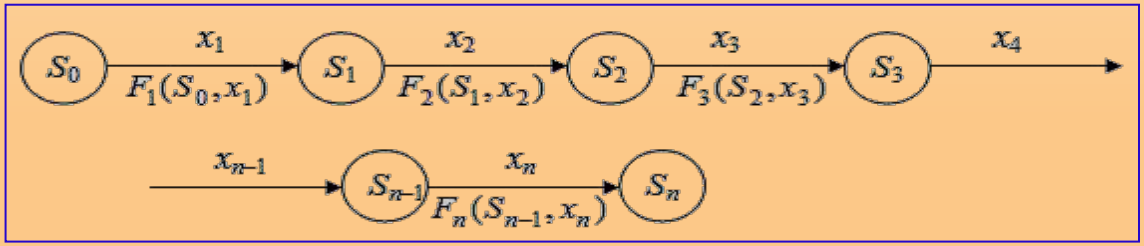
Принцип 1. “Принцип оптимальности” Беллмана:
Каково бы ни было состояние системы S в результате какого-либо числа шагов, на ближайшем шаге нужно выбирать управление так, чтобы оно в совокупности с оптимальным управлением на всех последующих шагах приводило к максимальному выигрышу на всех оставшихся шагах, включая данный.

Целевая функция $f_i(S_{i-1}, x_i)$ на i-том шаге называется функцией Беллмана.

Принцип 2. Принцип Вложенности. Природа задачи, допускающей использование метода динамического программирования, не меняется при изменении количества шагов, т. е. форма такой задачи инвариантна относительно N.

Рассмотрим особенности математической модели динамического программирования

1. задача оптимизации формулируется как конечный многошаговый процесс управления;
2. целевая функция (выигрыш) является аддитивной и равна сумме целевых функций каждого шага: выбор управления x_k на каждом шаге зависит только от состояния системы k этому шагу S_{k-1}, и не влияет на предшествующие шаги (нет обратной связи);
3. состояние системы S_k после каждого шага управления зависит только от предшествующего состояния системы S_{k-1} и этого управляющего воздействия x_h (отсутствие последействия) и может быть записано в виде уравнения состояния: $S_k = f_k(S_{k-1}, x_k)$, $k = 1, n$;
4. на каждом шаге управление x_k зависит от конечного числа управляющих переменных, а состояние системы S_k зависит от конечного числа параметров;
5. оптимальное управление представляет собой вектор , определяемый последовательностью оптимальных пошаговых управлений: $X = (x^*_1, x^*_2, \ldots, x^*_k, \ldots, x^*_n)$, число которых и определяет количество шагов задачи.



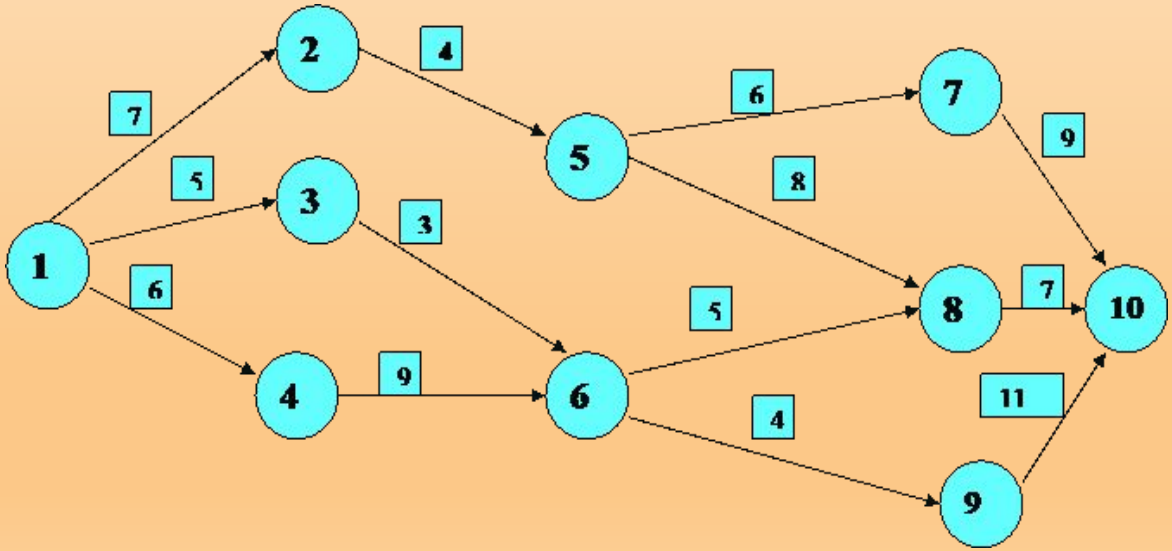
Виды задач, решаемых методами динамического программирования

1. Задача о нахождении кратчайшего либо наилучшего пути
2. Задача вычисления полного количества путей в графе
3. Задачи маршрутизации
4. Разработка правил управления запасами
5. Разработка принципов календарного планирования производства
6. Определение необходимого объема запасных частей, гарантирующего эффективное использование оборудования.
7. Распределение дефицитных капитальных вложений между возможными направлениями их использования.
8. Выбор методов проведения рекламных кампаний

Задача о нахождении кратчайшего пути в графе.

Пусть транспортная сеть состоит из **10** узлов, часть из которых соединены магистралями. На рис.1 –модель сети дорог и стоимости перевозки единицы груза между отдельными пунктами сети.

Необходимо определить маршрут доставки груза из пункта **1** в пункт **10**, обеспечивающий наименьшие транспортные расходы.



- **Процедура решения задач методом динамического программирования состоит из 2-х стадий:**
- 1. С конца в начало –предварительной
- 2.С начала в конец - окончательной.

- **Преимущества:**

1. Метод динамического программирования дает возможность решать задачи, которые раньше не исследовались из-за отсутствия соответствующего математического аппарата.
2. Метод динамического программирования в ряде случаев сокращает объем вычислений при поиске оптимальных решений.

- **Недостатки:**

1. Отсутствие универсального алгоритма, который был бы пригоден для решения всех задач.
2. Трудоемкость при решении задач большой размерности.

Понятие о NP-полных задачах. Примеры.

Опр. 2. P – класс задач, решаемых за полиномиальное (от размера входа) время.

Примеры:

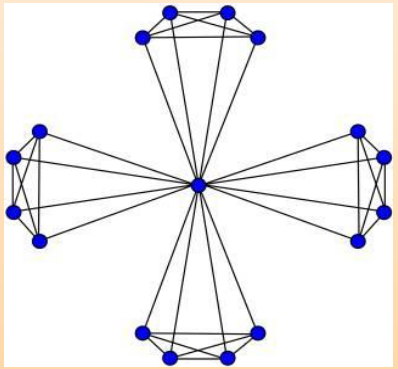
1. **задача о существовании пути в графе,**
2. **задача о взаимно простых числах и т.д.**

Опр. 3. NP – класс задач, верифицируемых (проверяемых) за полиномиальное время.

NP означает - Нелинейный Полиномиальный - то есть, оценка объема вычислений либо времени выполнения алгоритма выражается нелинейным многочленом от объема входной информации (квадратным, кубическим и т. д.).

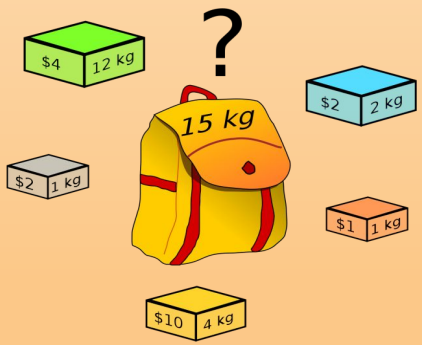
- Есть 2 варианта оценок:
  1) по объему - количеству вариантов, из которых надо выбрать решения (этой оценке соотв. NP-полный класс задач);
  2) по времени решения - количеству итераций алгоритма, необходимых для получения решения (этой оценке соотв. NP-трудный класс задач)

- **Список Карпа** — список, состоящий из формулировки и доказательства NP- полноты 21 задачи, опубликованный Ричардом Карпом в 1972 году, приведем ниже две NP-полных задачи:



1.Задача Штейнера о минимальном дереве состоит в поиске кратчайшей сети, соединяющей заданный набор точек плоскости.

2.Задача о рюкзаке (или задача о ранце) — NP-полная задачи оптимизации. Цель: уложить как можно большее число ценных вещей в рюкзак при условии, что общий вес не превысит заданный предел.



Алгоритм имитации отжига — алгоритм решения различных оптимизационных задач.

Основан на моделировании реального физического процесса, который происходит при кристаллизации вещества из жидкого состояния в твёрдое, в том числе при отжиге металлов.

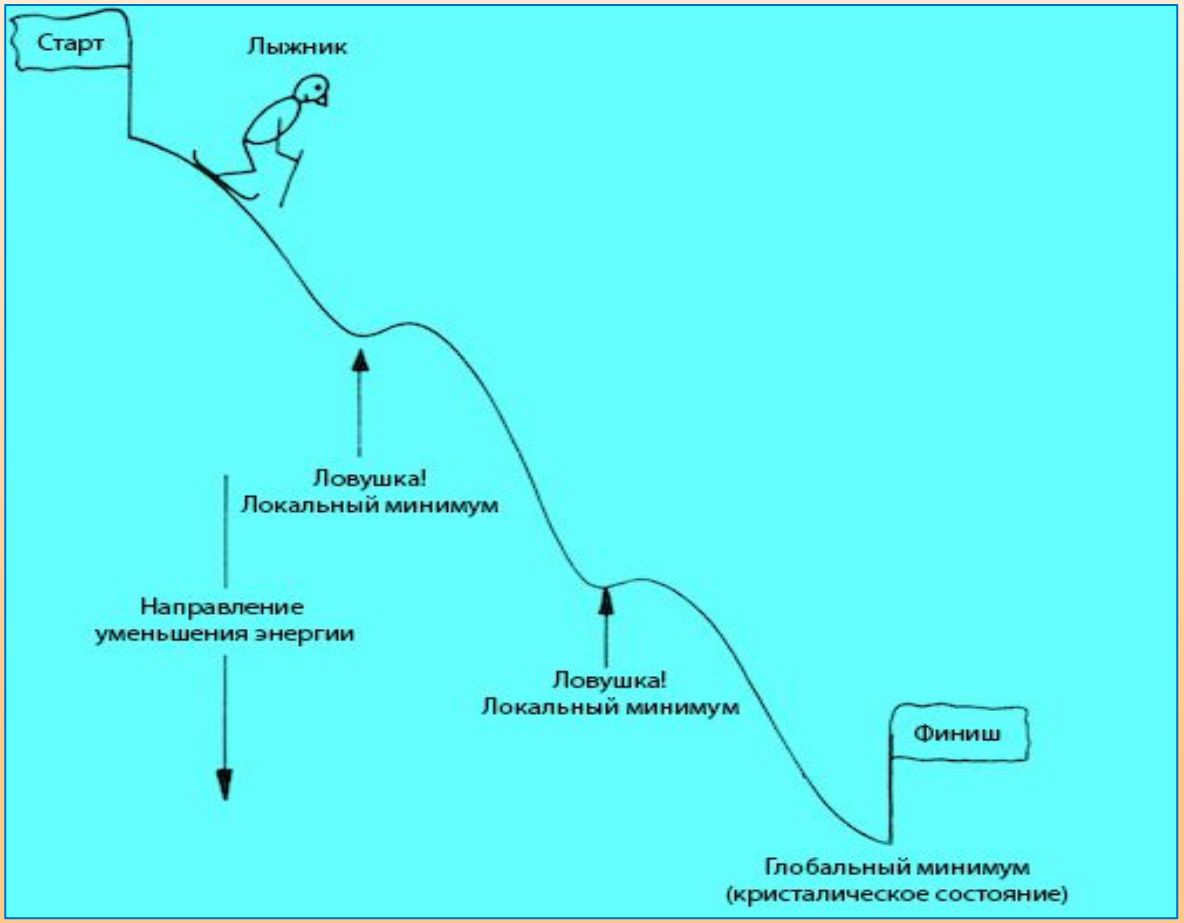
- Целью алгоритма является минимизация некоторого функционала, имеющего, в общем случае много локальных максимумов.

Алгоритм имитации отжига

1. Выбор начального решения и начальной температуры
2. Оценка начального решения
3. Основной шаг алгоритма
1. Случайное изменение текущего решения
2. Оценка измененного решения
3. Критерий допуска
4. Уменьшение температуры и, если температура больше некоторого порога, то переход к основному шагу

Области применения алгоритма имитации отжига

1. Создание пути
2. Реконструкция изображения
3. Назначение задач и планирование
4. Размещение сети
5. Глобальная маршрутизация
6. Обнаружение и распознавание визуальных объектов
7. Разработка специальных цифровых фильтров


Старт
Лыжник
Ловушка!
Локальный минимум
Направление
уменьшения энергии
Ловушка!
Локальный минимум
Финиш
Глобальный минимум
(кристалическое состояние)

Лев Семенович Понтрягин – выдающийся российский ученый

Понтрягин
Лев Семёнович
1908-1988

Вклад **Понтрягина** в математическую науку поистине огромен, его по праву можно назвать классиком математики XX века.
Выдающиеся достижения: создание математической теории оптимального управления; доказательство принципа максимума, носящего сегодня его фамилию – принцип максимума Понтрягина.

Родился 21 августа (3 сентября) 1908 г., Москва.

- В 14 лет Лев потерял зрение в результате несчастного случая (взорвавшийся примус вызвал сильнейший ожог лица).
- Помощь одноклассников обеспечила окончание школы.
- Помощь матери -1925г – поступление в МГУ
- ***«идёт лекция профессора Бухгольца, все слушают не очень внимательно, вдруг голос Понтрягина: „Профессор, вы ошиблись на чертеже!“***

Этапы пути

1. Л. С. Понтрягин окончил физико-математический факультет МГУ по специальности «чистая математика» в 1929 г.
2. В 1930–1932 гг. Л. С. Понтрягин — доцент кафедры алгебры и сотрудник НИИ математики и механики при МГУ.
3. В 1932–1933 гг. — сотрудник лаборатории колебаний Института физики при МГУ.
4. С 1934 г. до конца жизни Л. С. Понтрягин преподавал в МГУ —профессор кафедры высшей геометрии и топологии механико-математического факультета в 1935–1958 гг.,
5. заведующий кафедрой оптимального управления факультета вычислительной математики и кибернетики МГУ

в 1970–1988 гг.

- **Ученая степень доктор физико-математических наук присуждена без защиты диссертации в 1935 г.**
- **Звание профессора по Специальности «математика (топология)» присвоено в 1935 г.**

Области научных интересов Л. С. Понтрягина

1. топология и топологическая алгебра,
2. теория дифференциальных уравнений,
3. **теория управления и математическая теория оптимальных процессов.**

ТРУДЫ – более 300.

1. Математическая теория оптимальных процессов. — 2-е изд. — М.: Наука, 1969. — 384 с. — Совместно с В. Г. Болтянским, Р. В. Гамкрелидзе и Е.Ф. Мищенко
2. Основы комбинаторной топологии. М: Гостехиздат, 1947.
3. Обыкновенные дифференциальные уравнения: Учеб. для гос. ун-тов.— М.: Наука, 1970.
4. *Понтрягин Л. С.* Линейная дифференциальная игра убегания // Тр. МИАН СССР. — 1971.
5. Избранные научные труды. В 3 т. — М.: Наука, 1988.
6. *Понтрягин Л. С.* Обобщения чисел. — М.: Наука, 1986. —
7. Знакомство с высшей математикой. Анализ бесконечно малых. — М.: Наука, 1980.
8. Знакомство с высшей математикой. Алгебра. — М.: Наука, 1987.
9. Математический анализ для школьников. — 3-изд., — М.: Наука, 1988.

Почетные звания и награды

1. Почётный член Лондонского математического общества (1953)
2. Почётный член Международной академии «Астронавтика» (1966)
3. Вице-президент Международного математического союза (1970—1974)
4. Почётный член Венгерской академии наук (1972)
5. Сталинская премия второй степени (1941) — за научную работу «Непрерывные группы» (1938)
6. Ленинская премия (1962)
7. Государственная премия СССР (1975) — за учебник «Обыкновенные дифференциальные уравнения», опубликованный (1974, 4-е издание)
8. Герой Социалистического Труда (1969)
9. четыре ордена Ленина (1953, 1967, 1969, 1978), орден Октябрьской Революции (1975), орден Трудового Красного Знамени (1945), орден «Знак Почёта» (1940)
10. Премия имени Н. И. Лобачевского (1966)

Хобби

- 1.Музыка

- 2.Лыжи
- 3. Коньки
- 4.Литература

- **Из книги “Жизнеописание моей жизни, составленное мной самим”**

На основе многолетнего опыта я пришёл к уверенности, что серьёзный успех в любой области человеческой деятельности требует предельного напряжения сил.

В то же время неизбежными являются многочисленные неудачи. С последними приходится мириться.

Поэтому следует терпимо относиться к неудачам других.