# Алгоритмы и анализ сложности

Структуры данных. Деревья.

# Абстрактные структуры данных. Деревья.

- ***Дерево*** – связный граф, не содержащий циклов.
- *Деревья*: корневые и некорневые.

- **Свойства некорневых деревьев.**

Пусть Т – неориентированный граф, тогда следующие свойства эквивалентны:

1. Т – дерево
2. Для любых двух вершин Т существует единственный путь, соединяющий их
3. Т – связен, но распадается на 2 связных подграфа при удалении любого ребра
4. Т – связен, количество_вершин=количество_ребер+1
5. Т – ацикличен, количество_вершин=количество_ребер+1
6. Т – ацикличен, но добавление любого ребра порождает цикл

# Абстрактные структуры данных. Деревья.

- ***Мат. модель корневого дерева*** – множество записей со следующими свойствами:

  1. существует выделенный узел (корень дерева);

  2. остальные узлы распределены по непересекающимся подмножествам, которые снова образуют деревья:

  - корни этих поддеревьев называются *потомками*

  - количество этих поддеревьев называется *степенью вершины*

  - корень поддерева с нулевой степенью называется *листом*

  - *уровень узла* – длина пути от корня до этого узла

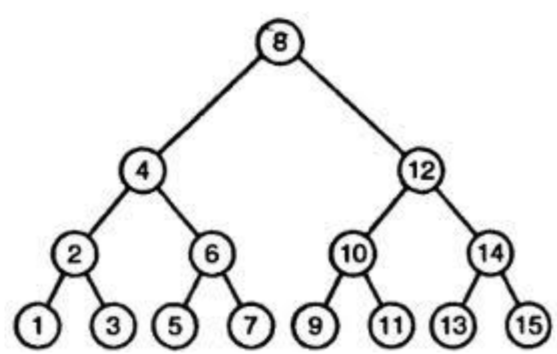
  - все вершины на пути от корня к узлу называются *предками* этого узла

- Если порядок поддеревьев имеет значение, то дерево называется упорядоченным.

# Абстрактные структуры данных. Деревья. Позиционные деревья.

- ***Позиционное дерево*** – это либо пустое множество, либо дерево, которое можно разбить на k+1 непересекающихся подмножеств, где k – это количество поддеревьев у каждого узла.

- ***Двоичное дерево*** – частный случай позиционное при k=2.



# Абстрактные структуры данных. Деревья. Способы представления деревьев.

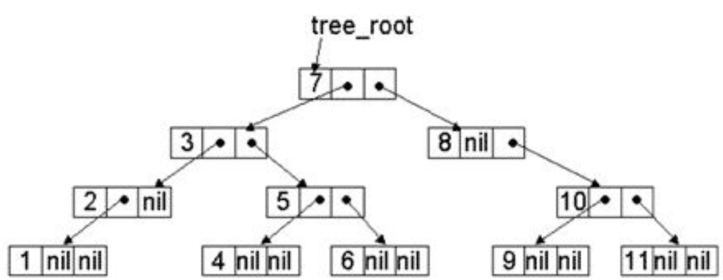
**Корневые деревья**

1) Общий случай: реализация с помощью списков.

Вершина = информационное поле + список указателей на потомков

2) Двоичное дерево:

Вершина = информационное поле + левый указатель + правый указатель



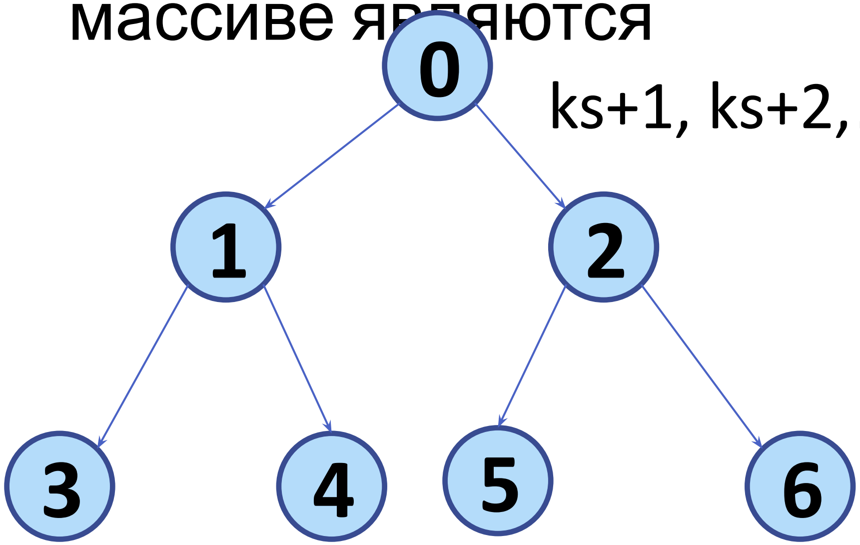

3) Позиционное дерево:

Вершина = информационное поле + массив указателей

# Абстрактные структуры данных. Деревья. Способы представления деревьев.

4) Специальный способ организации позиционного дерева – с помощью массива

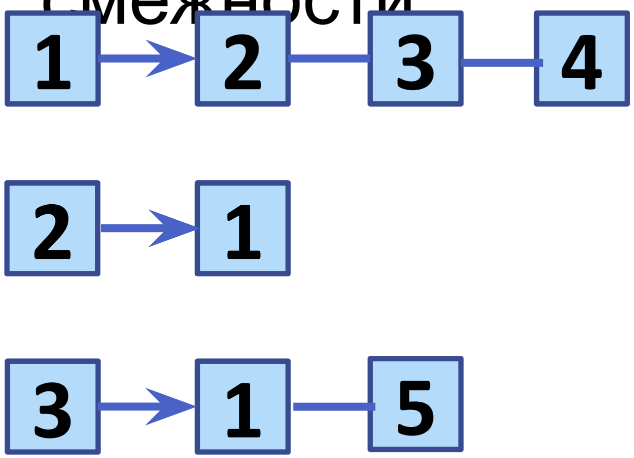
Потомком $s$-ого узла в массиве являются вершины ks+1, ks+2,…, ks+k.




Какие плюсы и минусы данной реализации?

**Некорневые деревья**

1) Общий случай: с помощью списков смежности




Есть массив всех вершин дерева. Для каждой вершины есть список вершин, с которыми она связана.

*Какой очевидный минус можно отметить?*

# Абстрактные структуры данных. Деревья. Способы представления деревьев.

## <u>Некорневые деревья</u>

2) ***Код Прюффера***. Пусть вершины дерева пронумерованы числами от 1 до N. Тогда *кодом Прюффера* называется последовательность из N-2 чисел, построенная по следующему алгоритму:

1. находим висячую вершину с минимальным номером
2. заносим смежную с ней вершину в выходную последовательность
3. повторяем пункты 1-2 N-3 раза

Выходная последовательность и будет кодом Прюффера.

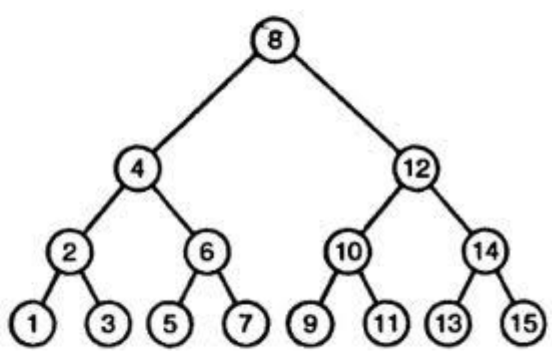
## Некорневые деревья

2) ***Восстановление дерева из кода Прюффера.***

1. Заводим список неиспользованных вершин. Изначально в него помещаются все вершины дерева.
2. Выбираем из этого списка минимальное число, которого нет в коде Прюффера.
3. Строим ребро, соединяющее найденное число с первым числом из ряда Прюффера. Вычеркиваем числа из списка и из кода.
4. Повторяем пункт 2-3, пока не закончатся все числа в коде Прюффера.
5. Строим ребро, соединяющее оставшиеся 2 числа из списка неиспользованных вершин.

# Абстрактные структуры данных. Деревья. Двоичные деревья поиска.

***Двоичное дерево поиска (ДДП)*** – это бинарное дерево такое, что каждому узлу предписан ключ, причем в левом поддереве ключи всегда меньше, чем в узле, а в правом – не меньше.



## Операции в двоичном дереве поиска

1. Поиск ключа FindKey(key)
2. Найти предыдущий ключ FindPrev(key)

   Найти следующий ключ FindNext(key)
3. Добавить вершину Add(key)
4. Удалить вершину Delete(key)
5. Найти минимальный и максимальный ключ Min(), Max()

# Абстрактные структуры данных. Операции в ДДП.

*Высотой дерева* называется максимальная длина пути от корня дерева к листу. Часто обозначается *h*.

**FindKey(key)**

Пошаговое сравнение искомого ключа с ключами в узлах ДДП.

Сложность алгоритма – $O(h)$.

**Add(key)**

Прим. Предполагается, что все ключи уникальны.

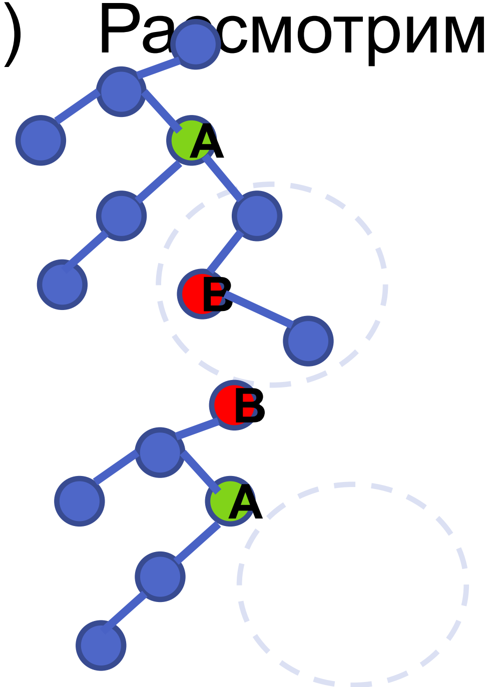
Вставляем ключ *key* туда, где есть пустое место, которое удовлетворяет всем условиям дерева двоичного поиска.

Сложность алгоритма – $O(h)$.

# Абстрактные структуры данных. Операции в ДДП.

**<u>FindNext(key)/ FindPrev(key)</u>**

1) Выполняется операция FindKey(key). Пусть вершина A – результат выполнения этой операции.
2) Рассмотрим 2 случая:




а. A имеет правое поддерево. Искомое значение – минимальный элемент в правом поддереве.

б. A не имеет правого поддерева. Искомое значение – ближайший предок A, для которого A находится в левом поддереве.

# Абстрактные структуры данных. Операции в ДДП.

**Min()/Max()**

Ищется самый левый/правый лист в дереве.

Модификация операции:

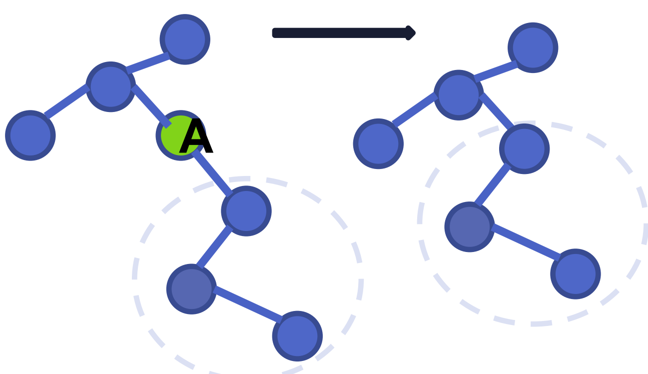
FindMin(key)/FindMax(key) – поиск минимального/максимального ключа в левом/правом поддереве для заданного ключа *key*.

Сложность алгоритма – *O(h)*.

# Абстрактные структуры данных. Операции в ДДП.

**Delete(key)**

1) Выполняется операция FindKey(key). Пусть вершина А – результат выполнения этой операции.
2) Рассмотрим 3 случая:

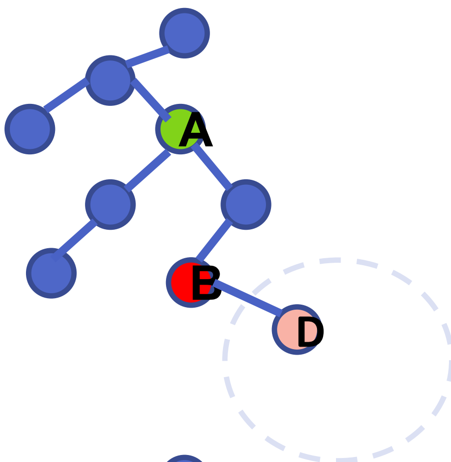


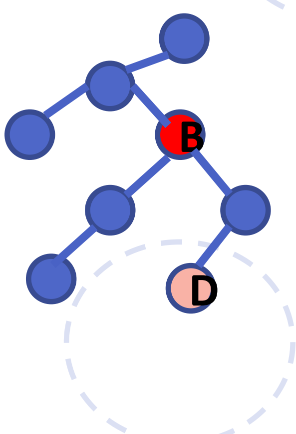

а. А не имеет потомков. Удаление вершины А – просто уничтожение вершины без изменений остального дерева.

б. А имеет ровно 1 потомка. Удаляем А и «подцепляем» её единственное поддерево к ближайшему предку вершины А.

# Абстрактные структуры данных.
# Операции в ДДП.

**<u>Delete(key)</u>**




в. A имеет 2 поддерева.

- осуществляем поиск FindNext(A); пусть это вершина B;
- вершина B не имеет левого поддерева;
- удаляем вершину A; записываем ключ B вместо A; удаляем вершину B из старого места в соответствии с п.а или п.б.

# Абстрактные структуры данных. Операции в ДДП.

**Выводы:**

1. Все интерфейсные операции имеют сложность *O(h)*.
2. Операции вставки и удаления не заботятся о сбалансированности дерева.

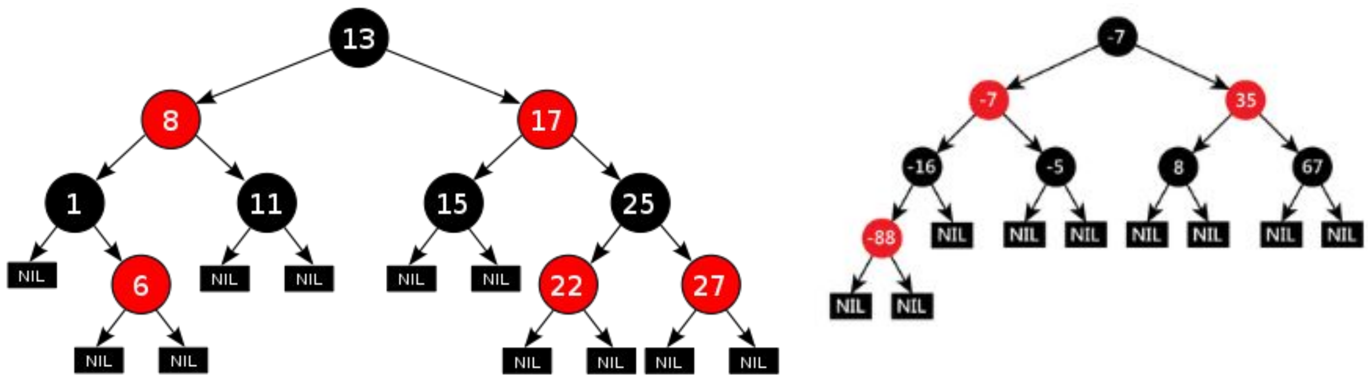
# Абстрактные структуры данных. Красно-черные деревья.

**Красно-черное дерево** – это дерево двоичного поиска, у которого выполняются следующие условия:

1. каждая вершина имеет цвет: *красный* или *черный*;
2. каждый лист имеет двух *фиктивных потомков,* которые окрашены в черный цвет; если у вершины только один реальный потомок, то второй будет фиктивным и окрашен в черный;
3. каждый красный узел имеет двух черных потомков;
4. на каждом пути от корня до листа содержится одинаковое количество черных вершин, которое называется *черной высотой*.

# Абстрактные структуры данных. Красно-черные деревья.

## Примеры КЧ-деревьев

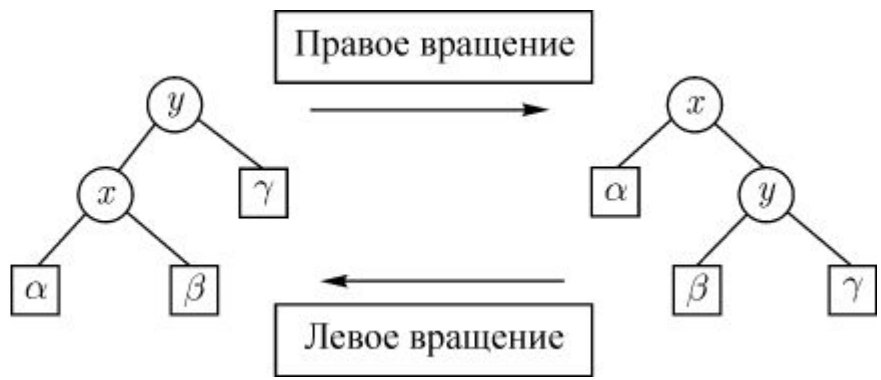



## Свойства сбалансированности КЧ-деревьев:

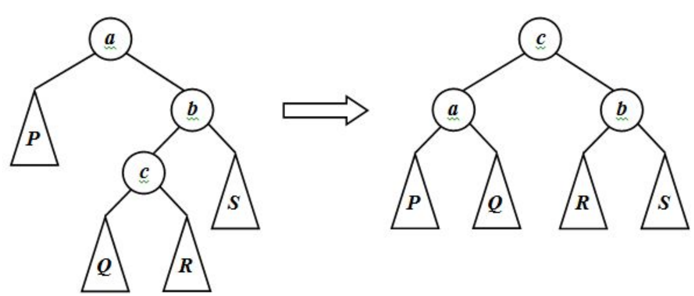
1) для каждого узла высота левого и правого поддерева отличается не более, чем в 2 раза;
2) высота КЧ-дерева, содержащего *n* вершин, не превосходит $2\log_2(n+1)$.

## Операция вращения ДДП

Операция вращения выполняется за константное время и позволяет преобразовать одно ДДП в другое ДДП (тот же набор ключей, но другая структура).




Прим. Данная операция позволяет выравнивать высоту ДДП.



**<u>Операция вставки в красно-черное дерево.</u>**

1) Вставка элемента *X* как в обычное ДДП; новая вершина *X* помечается красным цветом. Она имеет двух фиктивных черных потомков.
2) При вставке новой красной вершины *X* могло нарушиться только 3-е условие (имеет красного предка).

   <u>Возможны 2 ситуации:</u>

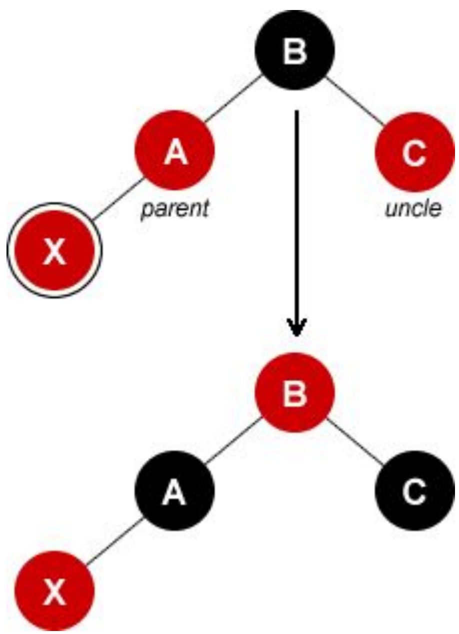
   а. красный «предок», красный «дядя»

   б. красный «предок», черный «дядя»

Абстрактные структуры данных.
Красно-черные деревья.

## Операция вставки в красно-черное дерево.

а. красный «предок», красный «дядя»

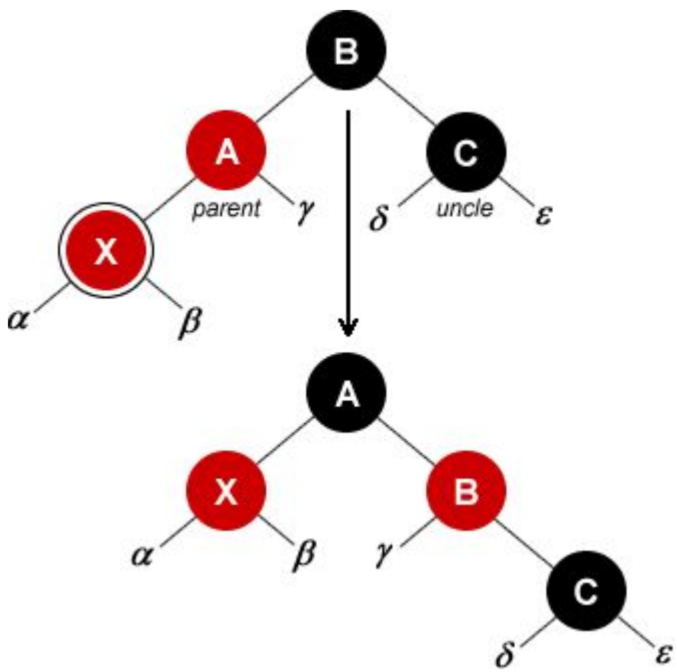



1. Перекрашиваем «предка» и «дядю» в черный цвет, а «дедушку» вершины *X* – в черный. При этом черная высота дерева не изменится.
2. Необходимо проверить предка вершины B. Если он окажется красным, то применяем перекрашивание вершин дальше, пока не будет выполнено условие 3 из определения.

# Абстрактные структуры данных. Красно-черные деревья.

## <u>Операция вставки в красно-черное дерево.</u>

<u>б. красный «предок», черный «дядя»</u>




1. Перекрашиваем «предка» в черный цвет, а «дедушку» - в красный. Таким образом добиваемся выполнения условия 3 из определения, но тогда нарушается условие 4 (о равенстве черной высоты).
2. Делаем правый поворот для выравнивания черной высоты.