

Алгоритмы с возвратом

Лекция 20

План лекции

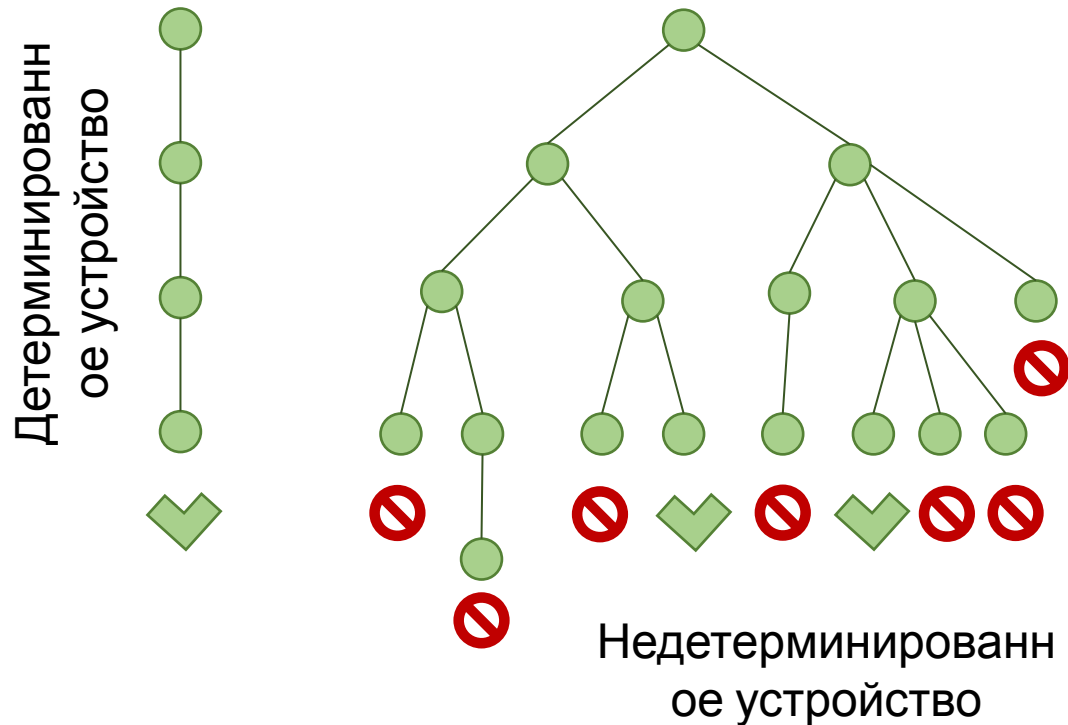
- Элементы теории сложности вычислений
 - Классы задач P и NP, сводимость, NP-полные задачи
- Метод поиска с возвратом
- Алгоритмы решения классических задач комбинаторного поиска

Понятие задачи

- Задачи – это подмножества множества входных данных
- «Решить задачу P для входных данных x » = «Проверить истинность $x \in P$ »
- Детерминированное исполняющее устройство
 - в математике – обычная машина Тьюринга
 - в реальности – компьютер
 - Размер ленты у машины Тьюринга не ограничен, а размер памяти у компьютера ограничен
- Недетерминированное исполняющее устройство
 - в математике – машина Тьюринга с неограниченным числом лент
 - в реальности – нет
 - Компьютер, с неограниченным числом процессоров

Разница между исполняющими устройствами

- Состояния устройства при выполнении четырех команд



- Работу недетерминированного устройства можно эмулировать на детерминированном устройстве
- Для эмуляции N команд недетерминированного устройства достаточно $\leq c^N$ команд детерминированного устройства
 - В худшем случае не \leq , а \approx

Понятие класса сложности задач

- $\text{Size}(x)$ – размер входных данных x
 - Обычно число битов в двоичном представлении x
- $\text{MaxOp}(n)$ – ограничение на число исполненных команд в зависимости от размера входных данных
 - Например, $\text{MaxOp}(n) = n * \log_2(n)$ и т.п.
- Класс сложности – множество задач, таких что для любых входных данных x для решения задачи требуется исполнить не более $C * \text{MaxOp}(\text{Size}(x))$ команд на исполняющем устройстве
 - Константа C зависит от задачи и не зависит от x

Класс P

- P = deterministic Polynomial
- Число команд при решении на детерминированной машине Тьюринга ограничено полиномом от размера входных данных
 - проверка делимости чисел
 - проверка связности графа
 - проверка кратчайшего расстояния между двумя вершинами в графе на $\leq \text{const}$
 - Как узнать это расстояние точно, решив $\log_2(\text{сумма длин всех дуг})$ таких задач?
 - ...

Класс NP

- NP = Non-deterministic Polynomial
- Число команд при решении на недетерминированной машине Тьюринга ограничено полиномом от размера ВХОДНЫХ ДАННЫХ
 - Все задачи класса P
 - Почему?
 - Приведите конкретные примеры
 - Приведите пример задачи НЕ из класса NP

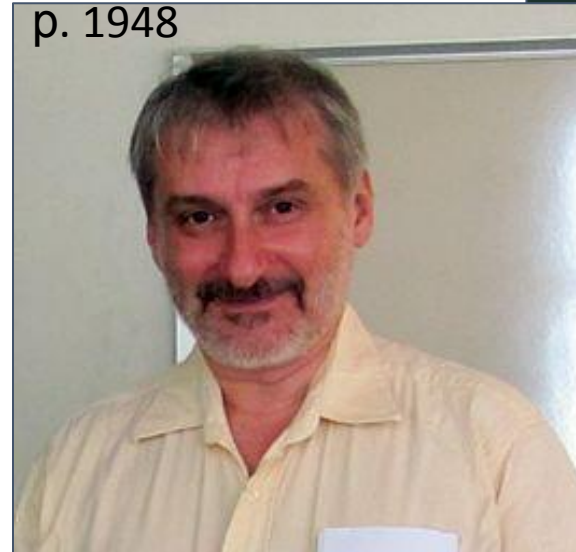
NP-полные задачи

- Задача P сводится к задаче Q , если существует функция f, такая что
 - f «вычислима за полиномиальное время»
 - для любых входных данных x «решить задачу P для x» равносильно «решить задачу Q для f(x)», т.е. $\forall x (x \in P \Leftrightarrow f(x) \in Q)$
- Задача является NP-полной, если она принадлежит классу NP и к ней сводится любая задача класса NP
 - Задача является NP-трудной, если к ней сводится любая задача класса NP, но сама она не обязательно из класса NP

Теорема Левина-Кука

- Проверка выполнимости произвольных булевых формул в КНФ является NP-полной задачей
 - Cook, Stephen (1971). "The complexity of theorem proving procedures". Proceedings of the Third Annual ACM Symposium on Theory of Computing. pp. 151–158.
 - Л. А. Левин. Универсальные задачи перебора (рус.) // Проблемы передачи информации. — 1973. — Т. 9, № 3. — С. 115—116.

Левин,
Леонид
Анатольевич
р. 1948

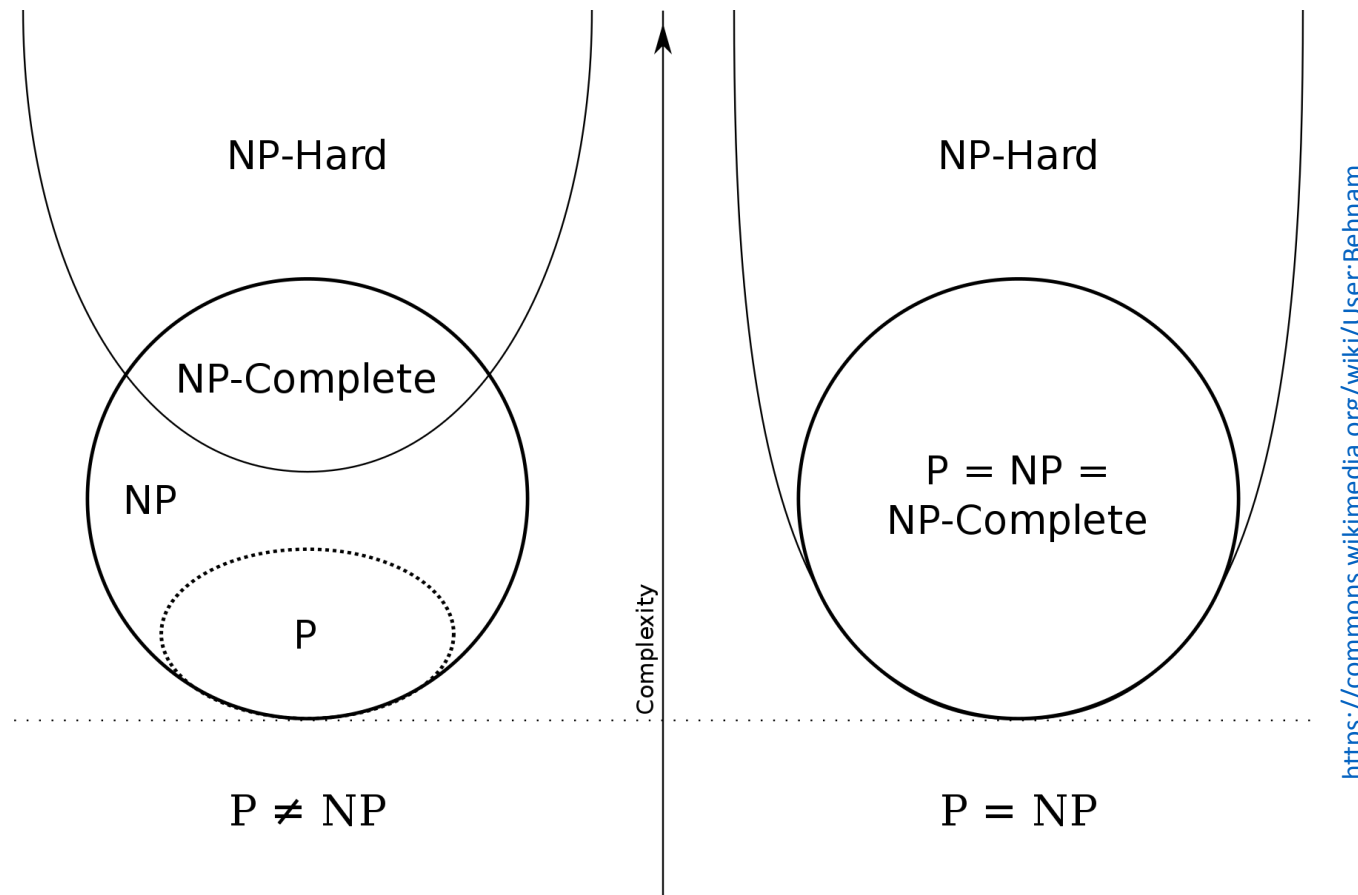


Cook,
Stephen Arthur
b. 1939

Примеры других NP-полных задач

- Существует ли в графе цикл, содержащий все вершины по одному разу? («задача коммивояжёра»)
- Существует ли в графе путь из одной вершины в другую длины не менее K ?
- Можно ли раскрасить вершины графа в C цветов так, чтобы концы каждого ребра были разного цвета? («раскраска графа»)
 - NP-полная начиная с $C = 3$
- Дано расположение дамек (простых шашек нет) на доске размером $N \times N$. Есть ли у белых выигрыш в данной позиции?
- «Задача о рюкзаке»
- ...

Возможные отношения между P и NP



Метод поиска с возвратом

- Метод проб и ошибок, backtracking
 - Примерно 1950 год
 - Derrick Henry Lehmer, 1905-1991
- Популярный метод в раннем искусственном интеллекте
- Эмуляция недетерминированных исполняющих устройств на обычном компьютере

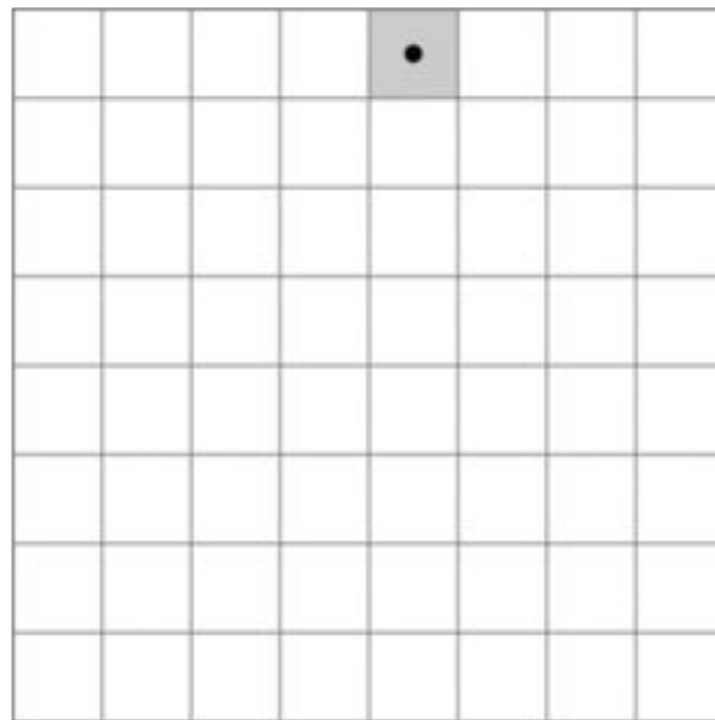


Обход доски шахматным конём

- Найти последовательность ходов шахматного коня, начинающуюся с заданного поля доски $N \times N$, такую что конь посещает каждое поле доски ровно один раз
- К какой NP-полной задаче сводится обход доски шахматным конем?



Пример обхода доски 5x5 и 8x8



Недетерминированное исполняющее устройство

- Состояние

- матрица $N \times N$, частично заполненная номерами ходов коня от 1 до $M \leq N^2$ и частично значением 0 («поле не посещено»)
 - Можно хранить список полей в порядке их посещения, но будет труднее проверять пройдено поле или нет

- Команды

- `GetNextBoard(board)`
 - Если возможно, то сделать следующий ход; иначе «неудача»
 - Недетерминированная команда

Обход доски шахматным конём на недетерминированном устройстве

```
BuildKnightTour(startSquare):
```

```
    board[startSquare] = 1
```

```
    for freeSquareCount in GetSquareCount(board) - 1 ... 1:
```

```
        board = GetNextBoard(board)
```

```
    return board
```

Детерминированная реализация

```
struct TBoard {
    int Size, Row, Column;
    int** Squares;
};
enum { MoveCount = 8 };
int BuildTour(int freeSquareCount, struct TBoard* board) {
    if (freeSquareCount == 0) {
        return 1;
    }
    struct TBoard nextBoard = MakeBoard(board->Size);
    int success = 0;
    for (int idx = 0; !success && idx < MoveCount; ++idx) {
        CopyBoard(*board, &nextBoard);
        success = TryMove(idx, &nextBoard)
            && BuildTour(freeSquareCount - 1, &nextBoard);
    }
    DestroyBoard(nextBoard);
    return success;
}
```

```
int TryMove(int idx, struct TBoard* board) {
    int row = board->Row, column = board->Column;
    int** squares = board->Squares;
    int count = squares[row][column];
    int change[MoveCount] = { 1, -1, -2, -2, -1, 1, 2, 2 };
    row += change[MoveCount - 1 - idx];
    column += change[idx];
    int isValid = Min(row, column) >= 0
        && Max(row, column) < board->Size;
    if (isValid && !squares[row][column]) {
        squares[row][column] = count + 1;
        board->Row = row;
        board->Column = column;
    }
    return isValid;
}
```

Пример эвристики

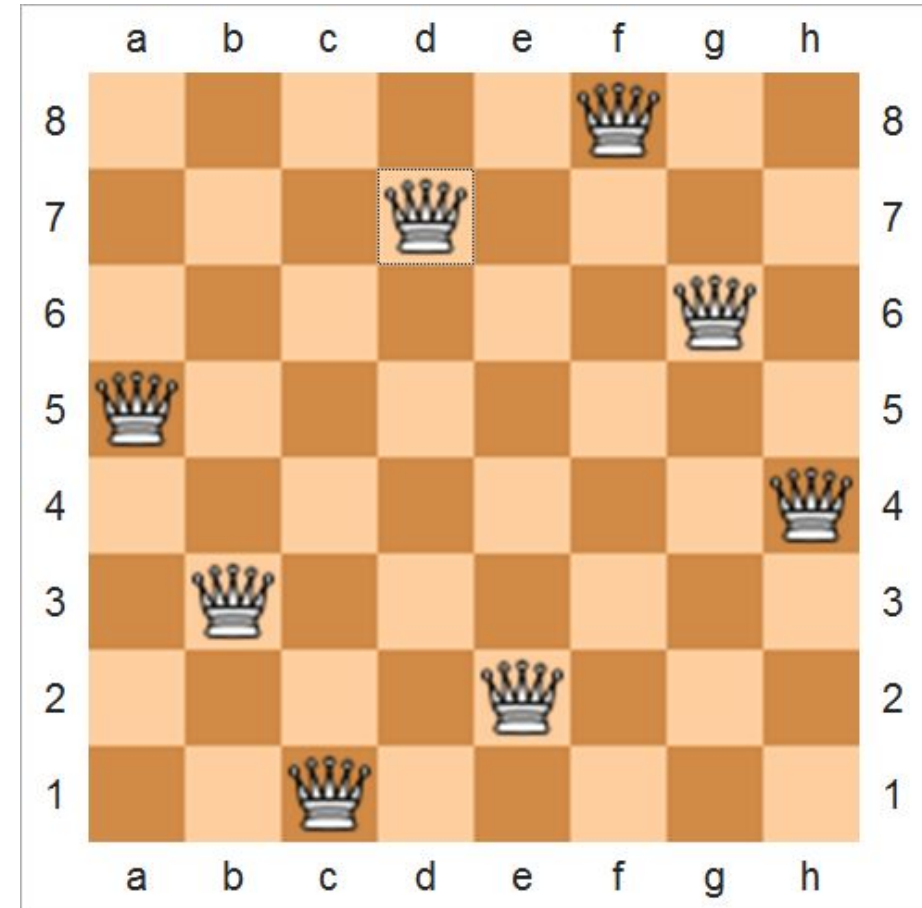
- Эвристика Варнсдорфа (Warnsdorff), 1823
 - На каждом ходу ставь коня на такое поле, из которого можно совершить наименьшее число ходов на еще не пройденные поля. Если таких полей несколько, берем любое из них.
- Позволяет обойти без возвратов доски от 5x5 до 76x76

Что известно из теории

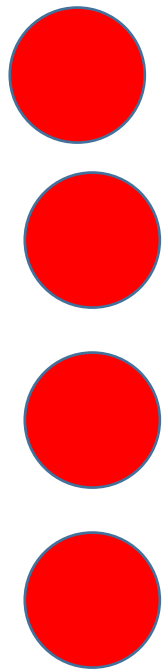
- Для любой прямоугольной доски с наименьшей стороной ≥ 5 существует (возможно незамкнутый) обход шахматным конем
 - Conrad, A.; Hindrichs, T.; Morsy, H. & Wegener, I. (1994). "Solution of the Knight's Hamiltonian Path Problem on Chessboards". Discrete Applied Mathematics. 50 (2): 125–134.
<https://doi.org/10.1016%2F0166-218X%2892%2900170-Q>
 - Cull, P.; De Curtins, J. (1978). "Knight's Tour Revisited" (PDF). Fibonacci Quarterly. 16: 276–28.
<http://www.fq.math.ca/Scanned/16-3/cull.pdf>
- Для любой доски $m \times n$ ($m \leq n$) существует замкнутый обход шахматным конем, за исключением случаев, когда выполнены одно или более из следующих условий:
 - m и n оба нечетные
 - $m = 1, 2$, или 4
 - $m = 3$ и $n = 1, 2, 3, 5$ или 6
 - Allen J. Schwenk (1991). "Which Rectangular Chessboards Have a Knight's Tour?". Mathematics Magazine: 325–332

Задача о расстановке ферзей

- «Требуется расставить 8 ферзей на шахматной доске так, чтобы ни один ферзь не угрожал другому»
- Формулировка -- Max Bezzel, 1848
- Первое решение -- Franz Nauck, 1850
 - Перечислил все 92 решения
 - Расширил на N ферзей на доске NxN
- Используется для проверки скорости работы алгоритмов с возвратом



Пример расстановки 4 ферзей



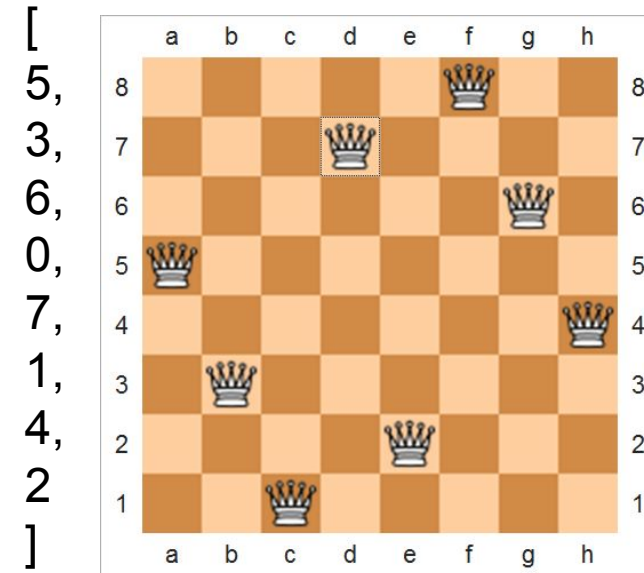
Недетерминированное исполняющее устройство

- Состояние

- вектор длины $M \leq N$,
заполненный номерами
вертикалей, в которых
находятся ферзи в
горизонталях 0 до $M-1$

- Команды

- PlaceNextQueen(board)
 - Если возможно, то добавить в
конец вектора board следующего
ферзя; иначе «неудача»
 - Недетерминированная команда



Расстановка ферзей с помощью недетерминированного устройства

```
PlaceQueens(Count):  
    board = []  
    for queenIdx in 1 ... Count:  
        board = PlaceNextQueen(board)  
    return board
```


Детерминированная реализация

```
struct TBoard {
    int Size;
    int QueenCount;
    int* QueenColumns;
};

int PlaceQueens(int queenIdx, struct TBoard* board) {
    if (queenIdx > board->Size) {
        return 1;
    }
    struct TBoard nextBoard = MakeBoard(board->Size);
    int success = 0;
    for (int col = 0; !success && col < board->Size; ++col) {
        CopyBoard(board, &nextBoard);
        success = TryPlaceQueen(col, &nextBoard)
            && PlaceQueens(queenIdx + 1, &nextBoard);
    }
    DestroyBoard(nextBoard);
    return success;
}
```

```
int TryPlaceQueen(int column, struct TBoard* board) {
    int upDiagonalIdx = column + board->QueenCount;
    int downDiagonalIdx = column - board->QueenCount;
    int* queens = board->QueenColumns;
    int isSafe = 1;
    for (int idx = 0; isSafe && idx < board->QueenCount; ++idx)
    {
        isSafe = column != queens[idx]
            && upDiagonalIdx != queens[idx] + idx
            && downDiagonalIdx != queens[idx] - idx
    }
    if (isSafe) {
        board->QueenColumns[board->QueenCount] = column;
        ++board->QueenCount;
    }
    return isSafe;
}
```

Что известно из теории

- Расстановка N ферзей за $O(N)$
 - E. J. Hoffman et al., "Construction for the Solutions of the m Queens Problem". Mathematics Magazine, Vol. XX (1969), pp. 66–72
<http://penguin.ewu.edu/~trolfe/QueenLasVegas/Hoffman.pdf>

Задача о рюкзаке

- Дано n вещей
 - i -я вещь имеет вес w_i , и стоимость c_i
- Дано число K – вместимость рюкзака
- Найти набор вещей максимальной стоимости при условии, что их общий вес не превышает K
 - $t_i = 0$, если вещь не взята
 - $t_i = 1$, если вещь взята

$$\sum_{i=1}^{i \leq n} t_i w_i \leq K$$

$$\sum_{i=1}^{i \leq n} t_i c_i \rightarrow \max$$

Схема перебора всех решений и выбора оптимального

```
Try(int i)
{
    if (включение приемлемо)
    {
        включение i-й вещи;
        if (i < n) Try(i+1);
        else проверка оптимальности;
        исключение i-й вещи;
    }
    if (приемлемо невключение)
    {
        if (i < n) Try(i+1);
        else проверка оптимальности;
    }
}
```

Метод ветвей и границ

- Вариант полного перебора
- Нахождение оптимальных решений среди допустимых
- Отсечение заведомо неоптимальных допустимых решений
- Ленд и Дойг 1960 общая задача целочисленного линейного программирования
 - A. H. Land and A. G. Doig An automatic method of solving discrete programming problems
- Литтл, Мурти, Суини и Кэрел 1963 задача коммивояжера

Метод ветвей и границ

- Целевая функция

- В задаче о рюкзаке это

$$\sum_{i=1}^{i \leq n} t_i c_i \rightarrow \max$$

- Ограничения

- В задаче о рюкзаке это

$$\sum_{i=1}^{i \leq n} t_i w_i \leq K$$

- Допустимые решения удовлетворяют ограничениям
- Оптимальные решения – это допустимые решения, дающие максимальное значение целевой функции

Метод ветвей и границ

- Разбиение множества допустимых решений на подмножества меньших размеров
- Подмножества допустимых решений образуют *дерево поиска (дерево ветвей и границ)*
- Для каждого подмножества допустимых решений оцениваем *снизу* и *сверху* множество значений целевой функции
 - Если нижняя граница совпадает с верхней границей, то Ц.Ф. достигает максимума (минимума) на данном подмножестве допуст. решений
 - Если *нижняя* граница для значений Ц.Ф. на подмножестве А больше *верхней* границы для значений Ц.Ф. на подмножестве В, то А не содержит минимума Ц.Ф., а В не содержит максимума Ц.Ф.

Метод ветвей и границ

- Ищем оптимальное решение при помощи обхода дерева ветвей и границ
 - Вид обхода выбираем в зависимости от задачи
- На каждом шаге обхода проверяем, содержит ли данное подмножество допустимых решений оптимальное решение
 - да, если верхняя граница == нижняя граница
 - обновляем известный min (max)
 - нет, если нижняя граница > известный min (верхняя граница < известный max)
 - не исследуем (пропускаем) подмножество допустимых решений
 - НЕИЗВЕСТНО
 - разбиваем подмножество допустимых решений на части и добавляем в дерево новые вершины

Метод ветвей и границ для решения задачи о рюкзаке

- Множество допустимых решений задаём массивом $t[]$ и номером x рассматриваемой вещи
 - значения $t[0] \dots t[x]$ уже зафиксированы
 - $t[0]*w[0]+t[1]*w[1]+\dots+t[x]*w[x] \leq K$
 - значения $t[x+1] \dots t[n]$ еще не зафиксированы
- Оценка снизу для множества допустимых решений t, x
 - тривиальная -- $t[0]*c[0]+t[1]*c[1]+\dots+t[x]*c[x]$
 - приведите примеры более "умных" оценок

Схема перебора всех решений и выбора оптимального (копия)

```
Try(int i)
{
  if (включение приемлемо)
  {
    включение i-й вещи;
    if (i < n) Try(i+1);
    else проверка оптимальности;
    исключение i-й вещи;
  }
  if (приемлемо невключение)
  {
    if (i < n) Try(i+1);
    else проверка оптимальности;
  }
}
```

Детализация метода ветвей и границ для задачи о рюкзаке

- Обозначим
 - tw – общий вес рюкзака к данному моменту
 - av – оценка сверху на конечную ценность рюкзака
 - $maxv$ – максимум, известный на данный момент

- "Включение приемлемо"

$$tw + w[i] \leq K$$

- "Проверка оптимальности"

```
if (av > maxv) {  
    opts = t;  
    maxv = av;  
}
```

- "Приемлемо невключение"

$$av < maxv$$

Заключение

- Классы задач P и NP, сводимость, NP-полные и NP-трудные задачи
- Метод поиска с возвратом
- Алгоритмы решения классических задач комбинаторного поиска
 - Обход доски шахматным конем
 - Расстановка ферзей

Задача о кубике

Задано описание кубика и входная строка.

Можно ли получить входную строку, прокатив кубик?

Перенумеруем грани кубика с 123456 на 124536:

1 – нижняя;

6 – верхняя; ($1+6 = 7$)

3 – фронтальная;

4 – задняя; ($3+4 = 7$)

2 – боковая левая;

5 – боковая правая ($2+5 = 7$).

Тогда соседними для i -й будут все, кроме i -й и $(7-i)$ -й.

Попробуем построить слово, начиная со всех шести граней.

Результат (в переменной q) 1, если можно получить слово, записанное в глобальной строке w , начиная n -го символа, перекатывая кубик, лежащий g -ой гранью.

```
int chkword(g, n) {
    if((n>strlen(w)) || (w[n]== '\ '))
        return 1;
    if(CB[g] != w[n]) break;
    for(i=1; i<=6; i++) {
        if((i != g) && (i+g != 7))
            q=chkwrд(i, n+1);
            if (q) return 1;
    }
}
```

Задача о стабильных браках

Имеются два непересекающихся множества A и B .

Нужно найти множество пар $\langle a, b \rangle$, таких, что $a \in A$, $b \in B$, и они удовлетворяют некоторым условиям.

Для выбора таких пар существует много различных критериев; один из них называется «правилом стабильных браков».

Пусть A — множество мужчин, а B — женщин. У каждого мужчины и женщины есть различные предпочтения возможного партнера.

Если среди n выбранных пар существуют мужчины и женщины, не состоящие между собой в браке, но предпочитающие друг друга, а не своих фактических супругов, то такое множество браков считается нестабильным.

Если же таких пар нет, то множество считается стабильным.

Алгоритм поиска супруги для мужчины m

Поиск ведется в порядке списка предпочтений именно этого мужчины.

```
Try(m) {
    int r;
    for (r=0; r<n; r++) {
        выбор r-ой претендентки для  $m$ ;
        if (подходит) {
            запись брака;
            if ( $m$  - не последний) Try( $m+1$ );
            else записать стабильное множество;
        }
        отменить брак;
    }
}
```


Выбор структур данных

Будем использовать две матрицы, задающие предпочтительных партнеров для мужчин и женщин:
ForLady и *ForMan*.

ForMan [m][r] — женщина, стоящая на r -м месте в списке для мужчины m .

ForLady [w][r] — мужчина, стоящий на r -м месте в списке женщины w .

Результат — массив женщин x , где $x[m]$ соответствует партнерше для мужчины m .

Для поддержания симметрии между мужчинами и женщинами и для эффективности алгоритма будем использовать дополнительный массив y : $y[w]$ — партнер для женщины w .

Конкретизация схемы

Предикат “подходит” можно представить в виде конъюнкции `single` и `stable`, где `stable` — функция, которую нужно еще определить.

```
Try (int m) {
    int r, w;
    for (r=0; r<n; r++) {
        w = ForMan[m][r];
        if (single[w] && stable) {
            x[m]= w; y[w]= m;
            single[w]=0;
            if (m < n) Try(m+1);
            else record set;
        }
        single[w]=1;
    }
}
```

Стабильность системы

Мы пытаемся определить возможность брака между t и w , где w стоит в списке t на r -м месте. Возможные источники неприятностей могут быть:

- 1) Может существовать женщина pw , которая для t предпочтительнее w , и для pw мужчина t предпочтительнее ее супруга.
- 2) Может существовать мужчина pt , который для w предпочтительнее t , причем для pt женщина w предпочтительнее его супруги.

1) Исследуя первый источник неприятностей, мы сравниваем ранги женщин, которых m предпочитает больше w . Мы знаем, что все эти женщины уже были выданы замуж, иначе бы выбрали ее.

```
stable = 1; i = 1;
while((i<r) && stable) {
    pw = ForMan[m][i];
    i = i+1;
    if(single[pw]) {
        stable = (ForLady[pw][m] > ForLady[pw][y[pw]]);
    }
}
```

2) Нужно проверить всех кандидатов pm , которые для w предпочтительнее «суженому». Здесь не надо проводить сравнение с мужчинами, которые еще не женаты. Нужно использовать проверку $pm < m$: все мужчины, предшествующие m , уже женаты.

Напишите проверку 2) самостоятельно!

Перебор ходов

- Из поля (x, y) достижимы не более 8 полей

$$(u, v) = (x + D[0,k], y + D[1,k]), k = 0, 1, \dots, 7$$

где массив $D[2][8]$ заполнен следующим образом

$$D = \begin{bmatrix} 1 & -1 & -2 & -2 & -1 & 1 & 2 & 2 \\ 2 & 2 & 1 & -1 & -2 & -2 & -1 & 1 \end{bmatrix}$$

- Для (x, y) вблизи края доски не рассматриваем k , для которых (u, v) лежат за пределами доски