Модуль 1. Полный перебор

Лекция 2

Генерация строк в произвольном алфавите, сочетаний и перестановок.

План

- Генерация слов в произвольном заданном алфавите
  - Алгоритм генерации слов.
  - Задача об упаковке ящиков.
- Генерация сочетаний
  - Алгоритм генерации сочетаний.
  - Задача о вершинном покрытии (Vertex Cover)
  - Задание 2.
- Генерация перестановок
  - Алгоритм генерации перестановок.
  - Задача коммивояжёра (TSP)
  - Полный перебор (строки, перестановки) для TSP.

Строки в произвольном алфавите

Вход:

- алфавит $A = \{a_1,\ldots,a_m\}$;
- натуральное число n.

Задача: последовательно сгенерировать все строки длины n в алфавите A.

Строки в произвольном алфавите

ProcessStrings(*A*, *k*)

```
if k = 0 then Process(A)
else
    for i:=0 to |A|-1 do
        A[k] := a_i;
        ProcessStrings(A, k-1);
```

Строки в произвольном алфавите

- Обратите внимание: в цикле по *i* можно устанавливать разные границы для разных *k*:

  for *i*:=0 to *m*[*k*]-1 do

- ! Придумайте нерекурсивный алгоритм для перебора всех строк в заданном алфавите.

Задача „Упаковка ящиков“

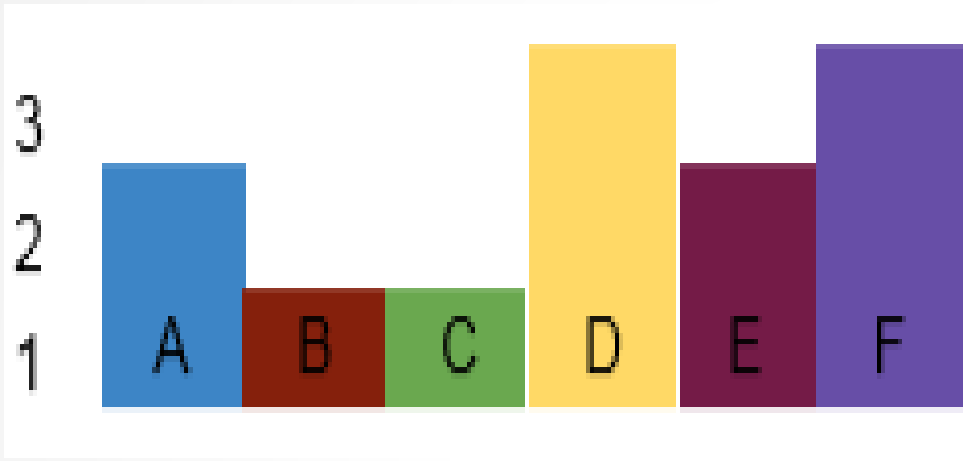
Задача «Упаковка ящиков» (1D Bin Packing)

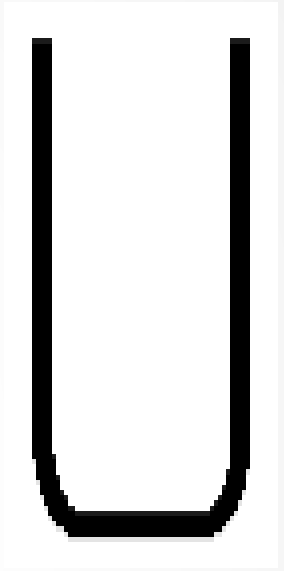
Дано:

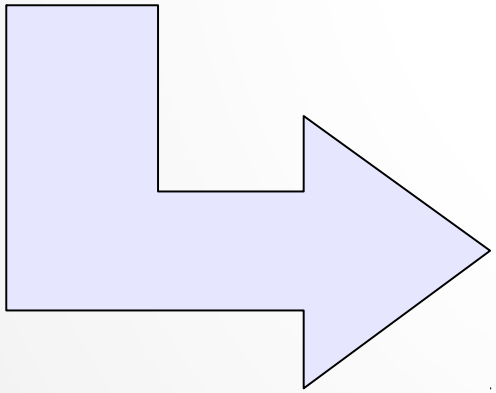
- n предметов, для каждого задан вес w_i.
- вместимость ящика b.

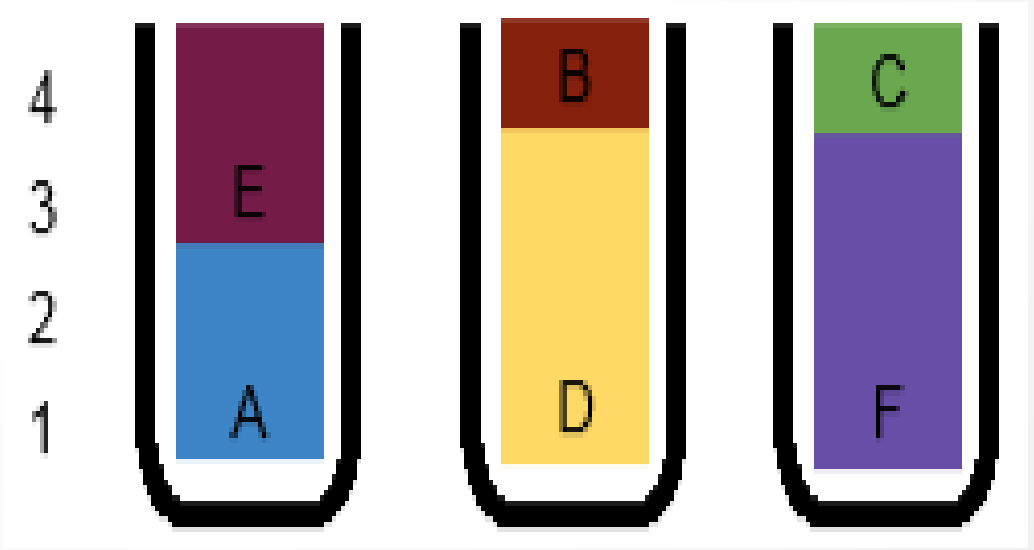
Задача: разместить (упаковать) данные предметы в минимальное количество ящиков заданной вместимости.

Задача „Упаковка ящиков“









Задача „Упаковка ящиков“

Идея переборного решения:

- Выбрать m — оценку сверху для количества ящиков.
- Перебрать все слова в алфавите $\{1,\ldots, m\}$.
- $A[i] = j$ означает, что i-й предмет кладём в j-й ящик.

Тривиальный выбор: $m = n$.

! Подумайте, как сократить перебор, если вам уже известно допустимое решение, использующее m' ящиков.

Генерация сочетаний

- Сочетание — подмножество заданной мощности. Сочетание из *n* по *k* — подмножество мощности *k* элементов из множества мощности *n*.
- Пример задачи, которую можно свести к перебору сочетаний:
  - покрытие множества (Set Cover, Set Covering, Set Coverage)
  - вершинное покрытие (Vertex Cover)

Вершинное покрытие

Дан граф $G(V, E)$.

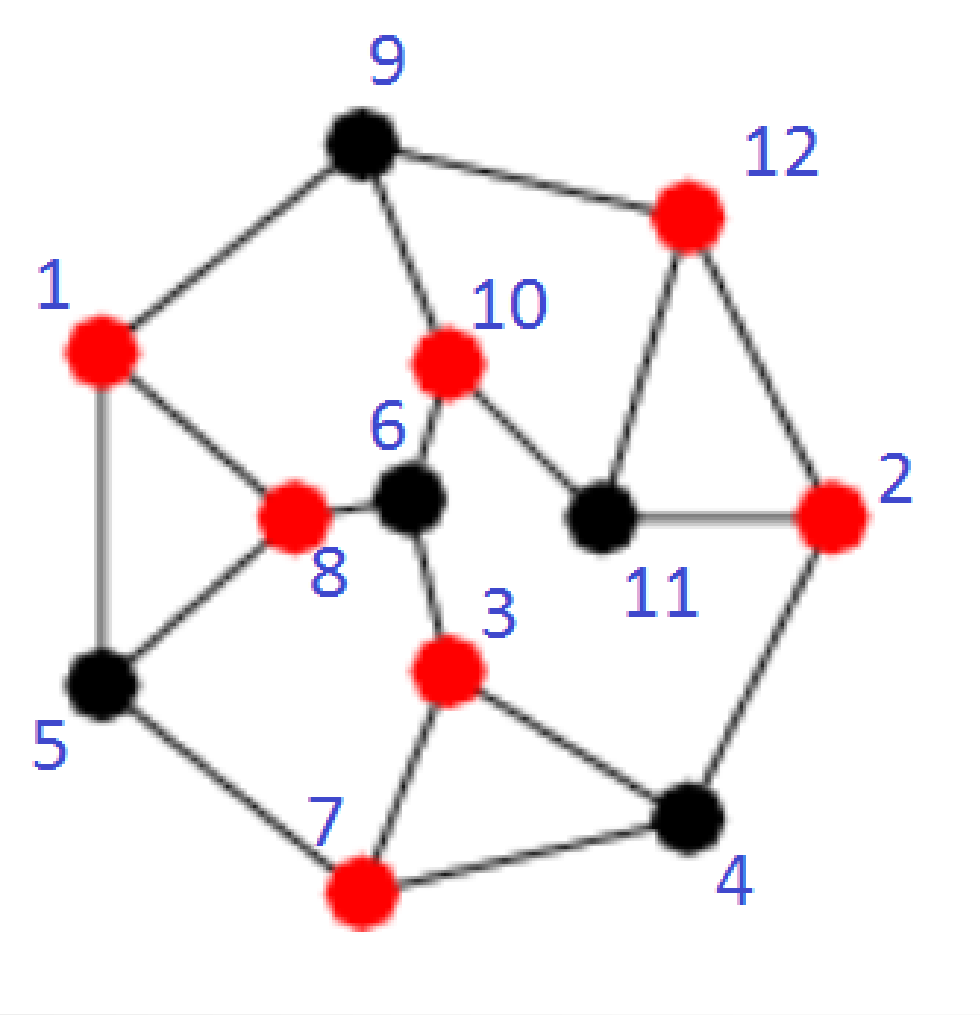
Вершинное покрытие:

$U \subseteq V$: $\forall\ e \in E \quad \exists\ u \in U$: ребро e инцидентно вершине u.

Задача: найти минимальное по мощности вершинное покрытие U.

Вершинное покрытие

http://mathworld.wolfram.com/VertexCover.html



Красные вершины образуют минимальное вершинное покрытие

Покрытие множества

Алгоритм решения задачи:

for k = 1 to n

Генерировать все сочетания из n по k.

Для каждого сочетания проверять, является ли оно покрытием.

Как только нашли покрытие - выход

Генерация сочетаний

1. Массив A[1..*k*]
2. *ProcessCombinations*(*A*,*n*,*k*)

ProcessCombinations(*A*,*n*,*k*)

```
if k = 0 then Process(A)
else
    for i = k to n do
        A[k] := i;
        ProcessCombinations(A, i-1, k-1)
```

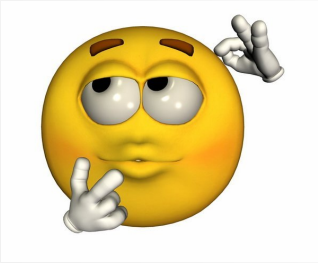
Генерация сочетаний

Теорема. Алгоритм ProcessCombinations

1) Корректен, т. е.

1. Обрабатывает только сочетания из n по k.
2. Обрабатывает все сочетания.
3. Каждое сочетание обрабатывается только 1 раз.

2) Оптимален по затратам для $k \leq n/2$.



Как построить оптимальный алгоритм для случая $k > n/2$?

Задание 2

- Задача о вершинном покрытии.
- Тестовый набор: приложен к заданию в Moodle.
- Формат входного файла:

  1-я строка:

  n m

  где n - количество вершин, m - количество дуг

  Потом - m строк, в каждой строке номера вершин-концов ребра, через пробел.

  Вершины нумеруются от 1 до n.

Генерация перестановок

Задача: для заданного n сгенерировать и обработать все перестановки степени n.

Решение:

1. Храним в массиве $A[1..n]$.
2. Инициализация: $\forall\ i \quad A[i] := i$.
3. Для всех k последовательно переставляем $A[k]$ с элементами в позициях 1,..., k-1.

Генерация перестановок

Вызов: ProcessPermutations(*A*,*k*)

ProcessPermutations(*A*,*k*)

if *k* = 1 then Process(*A*)

else

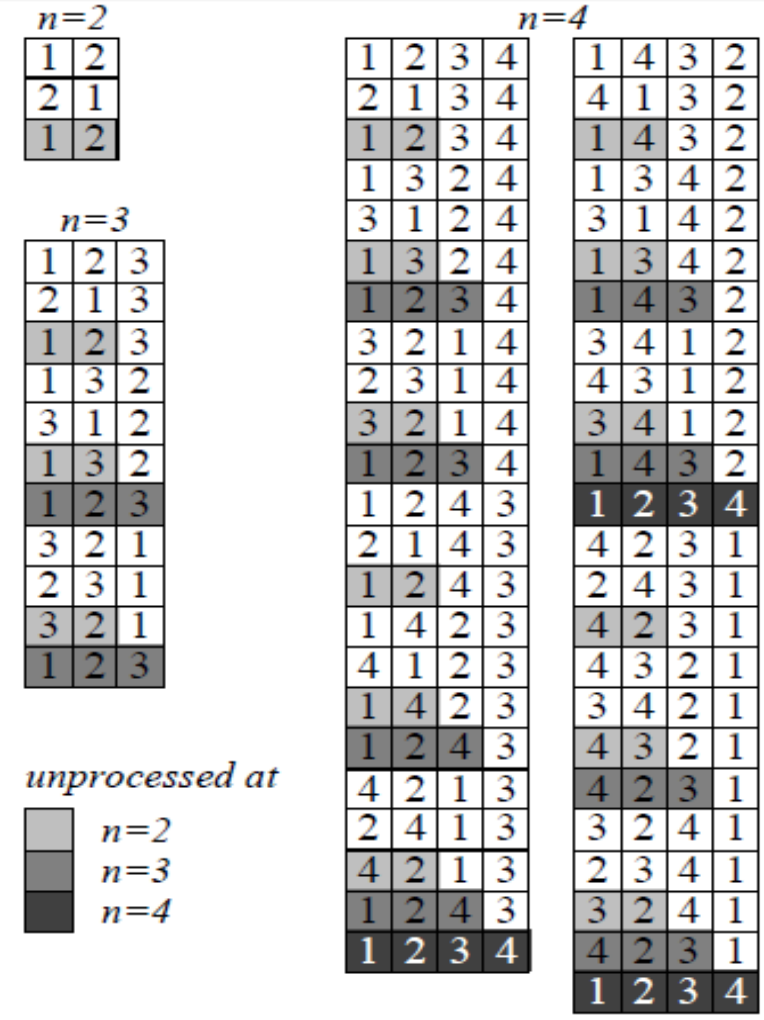
ProcessPermutations(*A*, *k*-1);

for *i* = *k*-1 downto 1 do

Поменять A[*k*] и A[*i*]

ProcessPermutations(*A*, *k*-1);

Поменять A[*k*] и A[*i*]



Задача коммивояжёра

Дано:

- Граф G(V, E).
- Стоимости дуг: c: $E \to R_+$.

Задача: найти на G гамильтонов контур Z минимальной стоимости.

- Гамильтонов контур = контур, проходящий по всем вершинам графа ровно 1 раз.
- Стоимость контура $c(Z)$ = сумма стоимостей всех дуг, входящих в контур.

Задача коммивояжёра

Алгоритм решения с помощью полного перебора:

- Перебор строк в заданном алфавите.
- Перебор перестановок.

Задача коммивояжёра

Алгоритм решения с помощью ***перебора слов в заданном алфавите***:

- Считаем, что граф задан списком смежности
- Пусть
  - *N*[*i*] — список смежности для *i*-й вершины
  - *D*[*i*] — степень *i*-й вершины, т. е. *D*[*i*] = | *N*[*i*] |
- Последовательность вершин Z храним в массиве A[1..*n*] в обратном порядке

$$A[n] \to A[n-1] \to \cdots \to A[2] \to A[1] \to A[n]$$

- В массиве A перебираем все «обобщённые» строки длины *n*. В обобщённой строке в *i*-й позиции может стоять символ из алфавита {1,..., *D*[A[*i*+1]]}.
- Чтобы не допускать появление дублей вершин в *Z*: в массиве *U*[*i*] отмечаем номера ещё не использованных вершин (т. е. *U*[*i*] = True <=> *i*-я вершина ещё не включена в *Z*).

Задача коммивояжёра

TSP_GeneralizedStrings

```
// Инициализация
for i:=1 to n-1 do   U[i] := True;
U[n] := False; A[n]:=n; // почему?

// Вызываем рекурсивную процедуру
TSP_ProcessGeneralizedStrings(A, U, n-1).
```

Задача коммивояжёра

```
TSP_ProcessGeneralizedStrings(A, U, k)
if k = 0 then Process(A)
else
    for j:=1 to D[A[k+1]] do
    m := N[A[k+1], j]
    if U[m] then
        U[m] := False;
        A[k] := m;
        TSP_ProcessGeneralizedStrings(A,U,k-1);
        U[m] := True;
```

Задача коммивояжёра

В процедуре Process необходимо:

- Проверить корректность: A задаёт замкнутый путь. Точнее, что его можно замкнуть, установив нужное значение в A[1].
- Оценить стоимость контура и сравнить со стоимостью рекордного решения.

Пусть $d = \max \{ \deg(v): v \in V \}$.

Тогда сложность алгоритма: $O(d^n)$

Задача коммивояжёра

Алгоритм решения с помощью перебора ***перестановок***:

- Пусть $n = |V|$.
- Контур можно задать как перестановку номеров вершин.
- Но считаем, что в конце всегда стоит v_n.
- Поэтому перебираем только перестановки чисел {1, …, n-1}.

Временная сложность: $O((n-1)!)$.

Задача коммивояжёра

Сравним, какой алгоритм лучше:

- Перебор строк в заданном алфавите: $O(d^n)$
- Перебор перестановок: $O((n\text{-}1)!)$

При $d < n / e$ алгоритм с перебором строк должен работать быстрее.

Надо проверить на практике...