Летняя практика 2018.

Вводная лекция по теме
„**Переборные алгоритмы**“

План

- Введение. Виды задач.
- Полный перебор вариантов решения
  - Генерация бинарных слов
  - Генерация слов в произвольном алфавите
  - Генерация сочетаний
  - Генерация перестановок
- Варианты обхода дерева.

Виды задач

- Распознавательные
- Вычислительные
- Оптимизационные

Виды задач

Распознавательная задача

- Формально: задача распознавания принадлежности входного слова заданному языку

  Дано: $L \subseteq A^*, \quad x \in A^*$

  Найти: верно ли, что $x \in L$?

- На практике: задача, в которой надо вернуть ответ „истина“ или „ложь“.

Виды задач

Вычислительная задача

- Множество входов: X
- Множество допустимых решений: S(x)
- Задача: для заданного $x \in X$ *найти* $s \in S(x)$

Виды задач

Оптимизационная задача

- Множество входов: X
- Множество допустимых решений: S(x)
- Функция стоимости решения: c: $S(x) \rightarrow R^+$
- Задача: для заданного $x \in X$ найти допустимое решение $s^* \in S(x)$:

  $\forall\ s \in S(x)$

  $c(s^*) \geq c(s)$ // максимизация

  $c(s^*) \leq c(s)$ // минимизация

- Обозначение: $s^* = opt(x)$; $c(s^*) = c^*(x)$

Метод полного перебора

- Полный перебор (Brute Force)
  - Последовательно генерировать все возможные решения
  - Для каждого сгенерированного решения x выполнять проверку на допустимость / оптимальность: *Process*(x)

Бинарные строки

Вход: натуральное число n.

Задача: последовательно сгенерировать все бинарные (битовые) строки длины n.

Бинарные строки

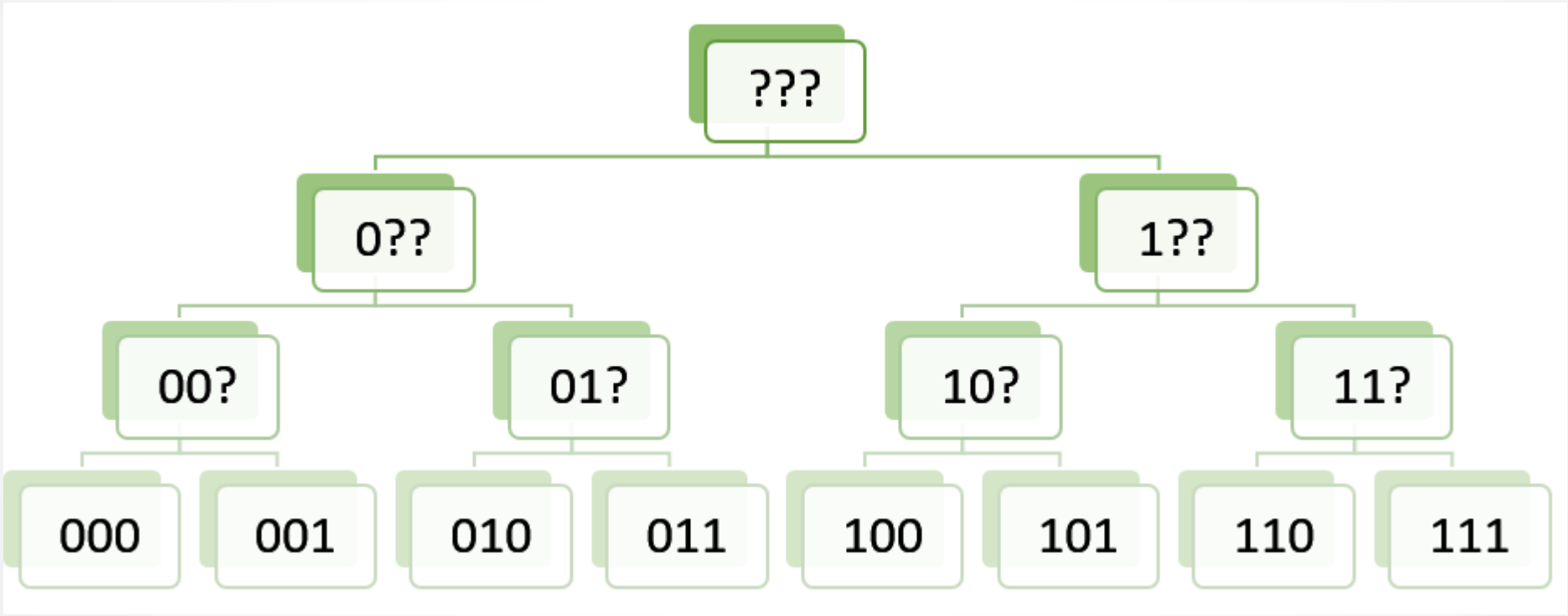
1. Массив S[1..*n*]
2. S := [null, null, ..., null]
3. *ProcessBinaryStrings*(*S*, 1, *n*)

Бинарные строки

ProcessBinaryStrings(*S*, *k*, *n*)

```
if k > n then Process(S)
else
    S[k] := 0;
    ProcessBinaryStrings(S, k+1);
    S[k] := 1;
    ProcessBinaryStrings(S, k+1);
```

Бинарные строки



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(*S*, *k*, *n*)

if $k > n$ then *Process*(*S*)

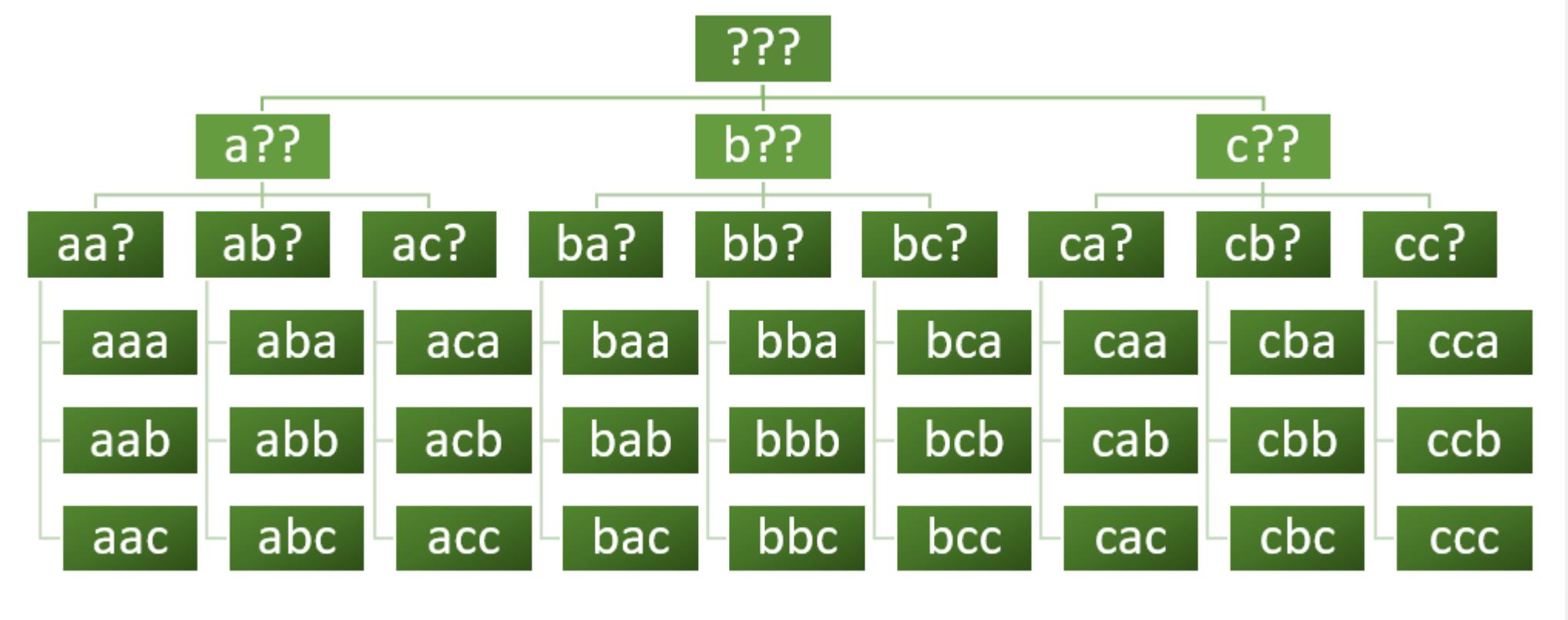
else

    for i:=0 to $|A|$-1 do

        $S[k] := a_i$;

        ProcessStrings(*S*, *k*-1, *n*);

Строки в произвольном алфавите



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.

Генерация сочетаний

- Сочетание — подмножество заданной мощности. Сочетание из n по k — подмножество мощности k элементов из множества мощности n.

Генерация сочетаний

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

ProcessCombinations(*S*,*n*,*k*,*j*)

if *j* > k then Process(*S*)

else

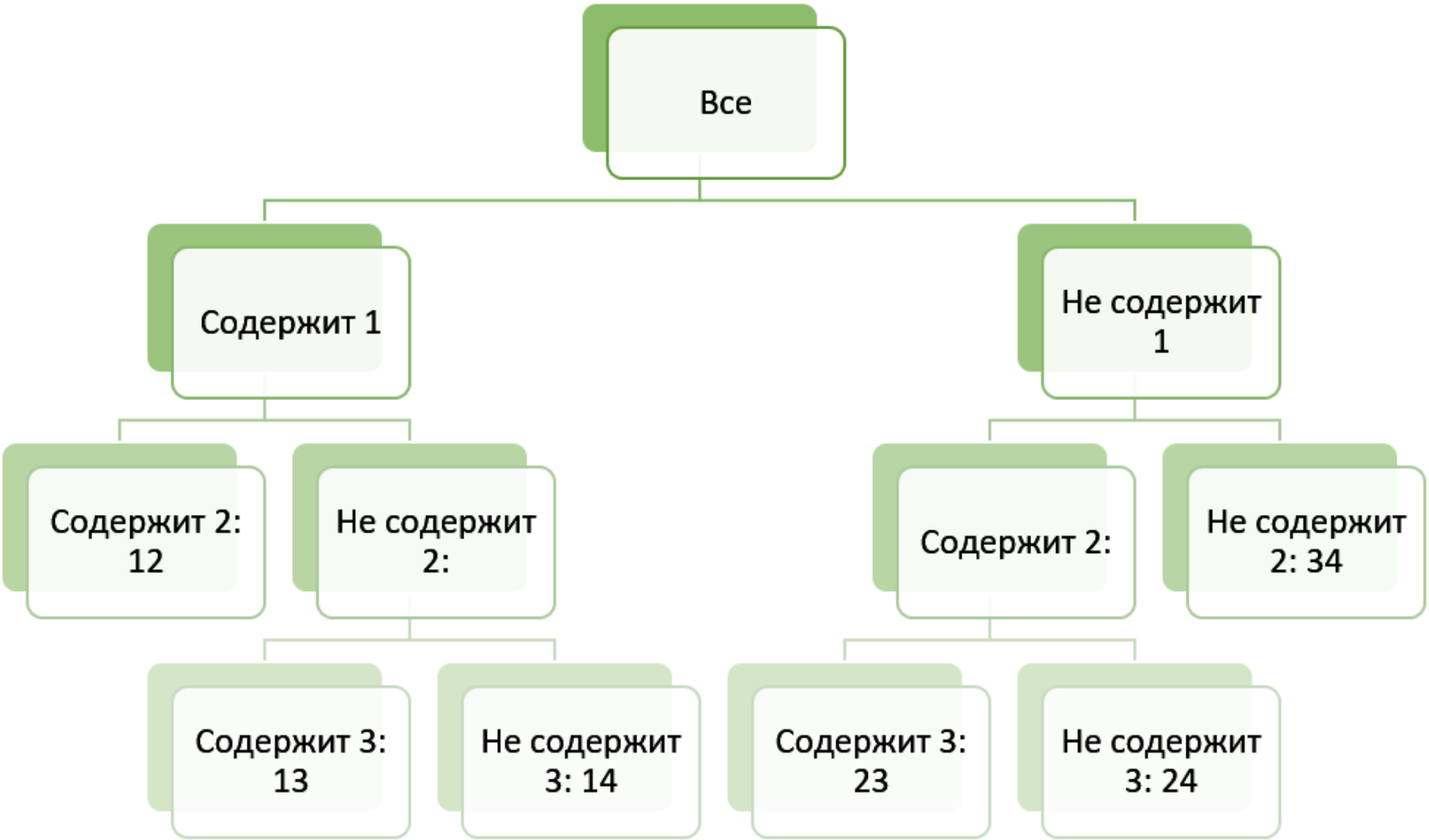
    if j=0 then lower := 1 else lower := S[j-1]+1;

    for *i* = *lower* to *n* do

        S[*k*] := *i*;

        ProcessCombinations(S, *n*, *k*, *j*+*1*)

Генерация сочетаний



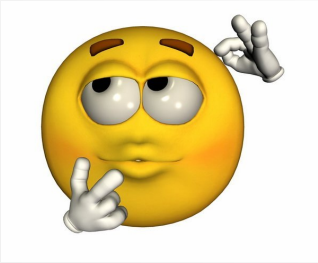
Генерация сочетаний

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 $k > n/2$?

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

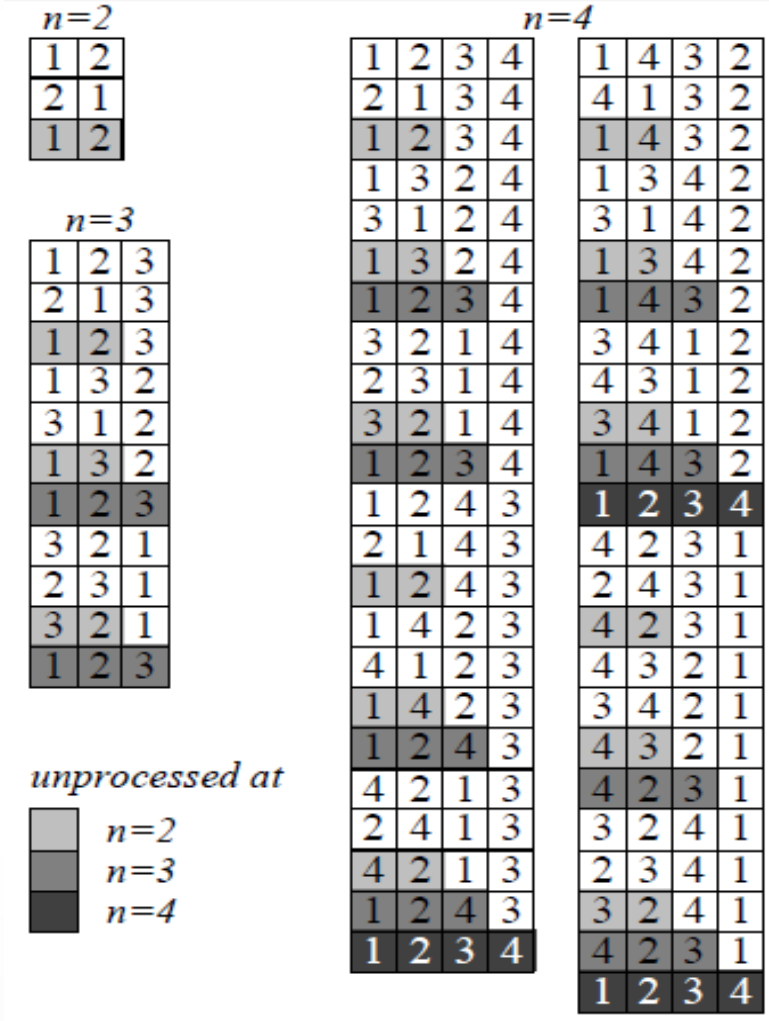
ProcessPermutations(*A*, *k*-1);

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

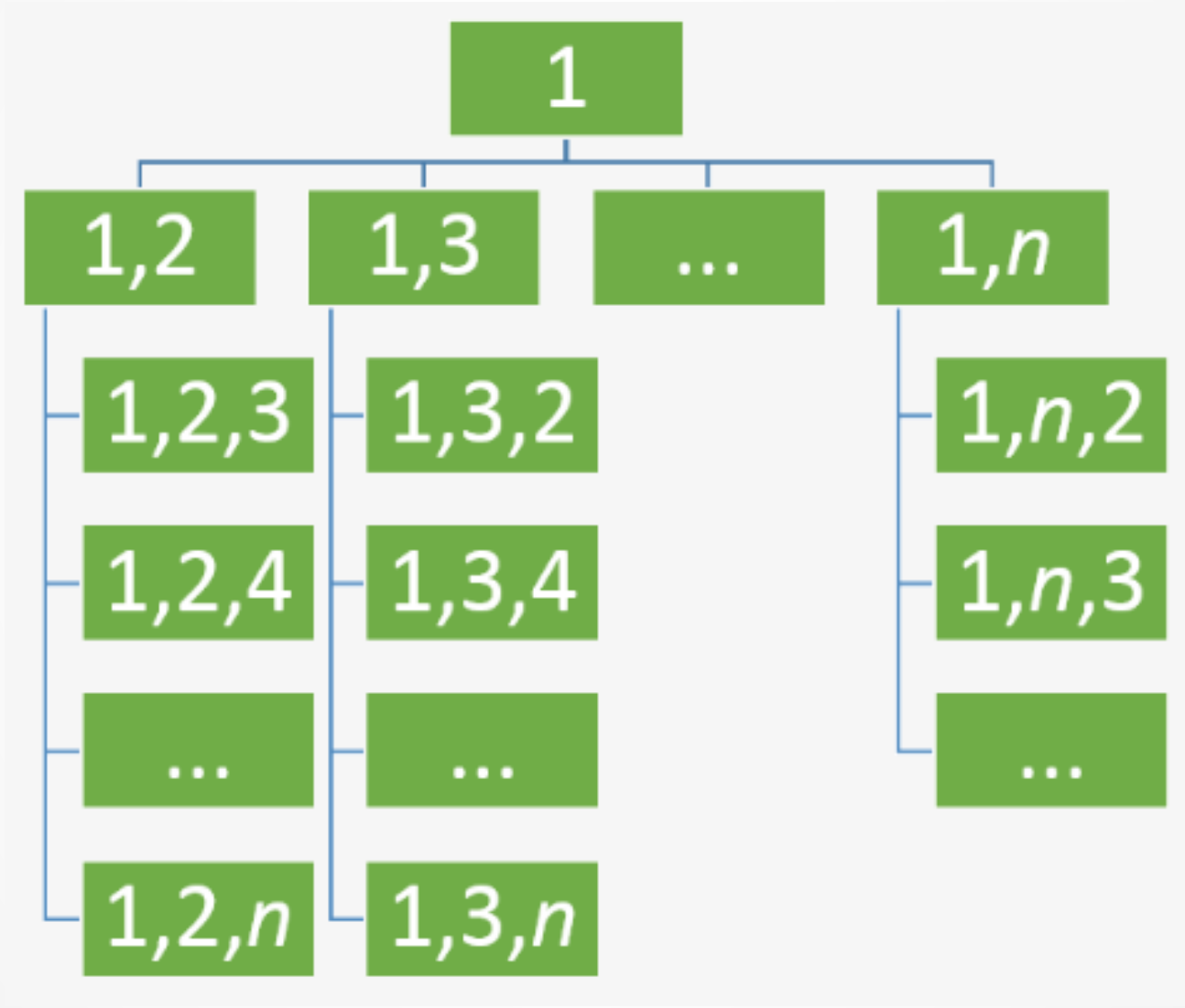
Поменять A[*k*] и A[*i*]

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

Поменять A[*k*] и A[*i*]



Генерация перестановок



Обход дерева вариантов

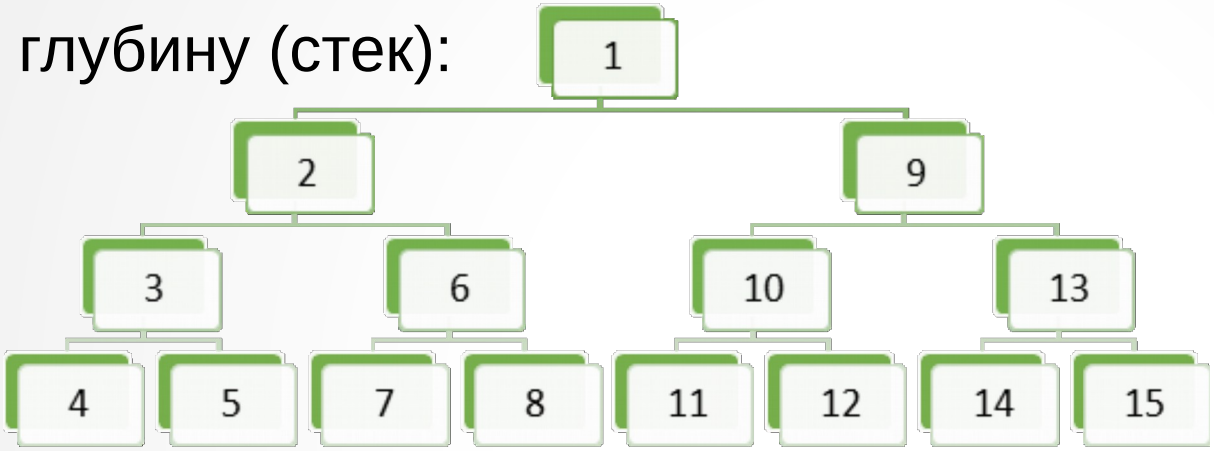
Все рассмотренные алгоритмы выполняют перебор вариантов решения в виде древовидной структуры (виртуальной! Хранить в памяти всё дерево не надо!). Допустимые решения задачи – листья дерева. Внутренние вершины – частичные решения = множества допустимых решений.

Существует два основных способа обходить дерево:

- В глубину („поиск в глубину“)
- В ширину („поиск в ширину“)

Обход дерева вариантов

В глубину (стек):



В ширину (очередь):