



Лекция 6

Перестановки

Перестановки

Перестановкой порядка N называется расположение N различных объектов в ряд в некотором порядке.

Например, для трех объектов — a , b и c — существует шесть перестановок:

$abc, acb, bac, bca, cab, cba.$

Для множества из N элементов можно построить $N!$ различных перестановок: первую позицию можно занять N способами, вторую — $(N - 1)$ способом, так как один элемент уже занят, и т. д. На последнее место можно поставить только один оставшийся элемент.

Следовательно, общее количество вариантов расстановки равно

$$N \cdot (N - 1) \cdot (N - 2) \cdot \dots \cdot 1 = N!$$

Далее будем рассматривать перестановки элементов множества $\{1, 2, 3, \dots, N\}$

Инверсии

Пусть даны базовое множество из N элементов $1, 2, 3, \dots, N$ и его перестановка $a_1, a_2, \dots, a_{N-1}, a_N$

Пара (a_i, a_j) называется *инверсией* (*инверсионной парой*) перестановки, если $a_i > a_j$ при $i < j$.

Например, перестановка $4, 1, 3, 2$ имеет четыре инверсии: $(4, 1)$, $(3, 2)$, $(4, 3)$ и $(4, 2)$.

Единственной перестановкой, не содержащей инверсий, является упорядоченная перестановка $1, 2, 3, \dots, N$.

Таким образом, каждая инверсия — это пара элементов перестановки, нарушающих ее упорядоченность.

Таблицей инверсий перестановки a_1, a_2, \dots, a_N называется последовательность чисел b_1, b_2, \dots, b_N , где b_j есть число элементов перестановки, больших j и расположенных левее j

(т. е. количество инверсий вида (x, j) , при $x > j$).

Например,

для перестановки **5 9 1 8 2 6 4 7 3**
таблица инверсий: **2 3 6 4 0 2 2 1 0.**

Свойство элементов таблицы инверсий:

$$b_N = 0,$$

...

$$0 \leq b_i \leq N - i,$$

...

$$0 \leq b_1 \leq N - 1.$$

Утверждение

Таблица инверсий единственным образом определяет соответствующую ей перестановку.

Построение перестановки по таблице инверсий

1 способ: проход по таблице инверсий справа налево

Создается заготовка перестановки из одного максимального числа. На каждом шаге в нее вставляется следующий по величине элемент с учетом того, сколько элементов, больших него, должно стоять перед ним.

Пример:

Таблица инверсий: 2 3 6 4 0 2 2 1 0

1.	9								
2.	9	8							
3.	9	8	7						
4.	9	8	6	7					
5.	5	9	8	6	7				
6.	5	9	8	6	4	7			
7.	5	9	8	6	4	7	3		
8.	5	9	8	2	6	4	7	3	
9.	5	9	1	8	2	6	4	7	3



Алгоритм П1:

построение перестановки по таблице инверсий справа налево

Вход:

$N > 0$ - количество элементов перестановки,

b_1, b_2, \dots, b_N – таблица инверсий,

$0 \leq b_j \leq N - j$.

P := пустая последовательность;

цикл по j от N вниз до 1

вставить число j в P после b_j элементов;

конец цикла;

Выход:

P – перестановка, соответствующая данной таблице инверсий

Построение перестановки по таблице инверсий

2 способ: проход по таблице инверсий слева направо

Создается заготовка пустой перестановки длины N .

На каждом шаге для каждого элемента перестановки, начиная с 1, отсчитывается в ней столько пустых ячеек, какое число записано в соответствующей позиции в таблице инверсий. В следующее за ними пустое место вставляется этот элемент.

Пример:

Таблица инверсий: 2 3 6 4 0 2 2 1 0



Перестановка :

5	9	1	8	2	6	4	7	3
---	---	---	---	---	---	---	---	---

Алгоритм П2:

построение перестановки по таблице инверсий слева направо

Вход:

$N > 0$ - количество элементов перестановки,
 b_1, b_2, \dots, b_N – таблица инверсий,
 $0 \leq b_j \leq N - j$.

$P :=$ последовательность из N пустых элементов;

цикл по i от 1 до N с шагом 1 выполнять

пропустить b_i пустых мест в P ;

поместить i на следующее пустое место;

конец цикла

Выход:

P – перестановка, соответствующая данной таблице инверсий

Инверсионный метод поиска всех перестановок

Таблица инверсий однозначно определяет перестановку и каждая перестановка имеет только одну таблицу инверсий.

Следовательно, если мы сумеем перебрать все таблицы инверсий, то с помощью алгоритмов П1 или П2 сможем по ним восстановить все перестановки.

Рассмотрим таблицу инверсий как N -значное число в такой необычной «системе счисления»: количество цифр, которое можно использовать в i -м разряде (с конца, начиная с 0) равно i .

Возьмем «минимальную» таблицу и будем последовательно прибавлять к ней, как к числу, единицу, пользуясь, например, алгоритмом сложения с переносом для многоразрядных чисел, модифицированным для нашей «системы счисления».

Генерация таблиц инверсии

4	3	2	1	0	
0	0	0	0	0	Шаг 0
0	0	0	1	0	Шаг 1
0	0	1	0	0	Шаг 2
0	0	1	1	0	Шаг 3
0	0	2	0	0	Шаг 4
0	0	2	1	0	Шаг 5
0	1	0	0	0	Шаг 6
0	1	0	1	0	Шаг 7
0	1	1	0	0	Шаг 8
0	1	1	1	0	Шаг 9
0	1	2	0	0	Шаг 10
...
4	3	2	1	0	Шаг 119

Алгоритм ИІ:

нахождение следующей таблицы инверсий

Пусть $B = b_1, b_2, \dots, b_N$ – таблица инверсий, построенная на предыдущем шаге.

Тогда следующая таблица инверсий получается из нее прибавлением к ней единицы:

$i := N$;

flag := **истина**;

пока flag **выполнять**

$x := b_i + 1$;

если $x > N - i$

то

начало

$b_i := 0$;

$i := i - 1$;

конец

иначе

начало

$b_i := x$;

 flag := **ложь**;

конец

конец пока

Алгоритм Дейкстры: поиск следующей по алфавиту перестановки

Пусть даны две перестановки

$$b = b_1, b_2, \dots, b_N \text{ и}$$

$$c = c_1, c_2, \dots, c_N$$

набора $1, 2, \dots, N$.

Говорят, что перестановка b предшествует перестановке c в алфавитном (лексикографическом) порядке, если для минимального значения k , такого что $b_k \neq c_k$, справедливо $b_k < c_k$.

Например, перестановка **1 2 3 4 5** предшествует перестановке **1 2 4 5 3** (здесь $k = 3$).

Первой перестановкой в алфавитном порядке является перестановка **1, 2, 3, . . . , N**,
а последней — **N, N-1, N-2, . . . , 1**

Идея алгоритма Дейкстры:

определить каким-либо образом функцию, которая по заданной перестановке выдает непосредственно следующую за ней в алфавитном порядке, и применять ее последовательно к собственным результатам начиная с самой первой перестановки, пока не будет получена последняя.

Например, для перестановки

1 4 6 2 9 5 8 7 3

следующей по алфавиту является
перестановка

1 4 6 2 9 7 3 5 8.

Алгоритм Дейкстры:

генерация следующей по алфавиту перестановки

Вход: $N > 0$ — количество элементов;

$a_1, a_2, \dots, a_{N-1}, a_N$ — предыдущая перестановка.

Шаг 1. Просматривая перестановку, начиная с последнего элемента, найдем такой номер i , что $a_{i+1} > \dots > a_N$ и $a_i < a_{i+1}$.

Если такого i нет, то последовательность упорядочена по убыванию и следующей перестановки нет: конец алгоритма.

Шаг 2. Найти в «хвосте» a_{i+1}, \dots, a_N элемент a_j , такой что $i+1 \leq j \leq N$, a_j есть наименьшее значение, удовлетворяющее условию $a_j > a_i$. После этого поменять местами a_i и a_j .

Шаг 3. Упорядочить «хвост» a_{i+1}, \dots, a_N по возрастанию. Для этого достаточно его инвертировать (обернуть в обратном порядке).

Выход: следующая по алфавиту перестановка за данной.

Пример построения следующей по алфавиту перестановки

Для перестановки

1 4 6 2 9 5 8 7 3

Найти следующую по алфавиту.

Шаг 1: 1 4 6 2 9 5 8 7 3

i *j*

Шаг 2: Поменять местами



Шаг 3: Обернуть хвост



Рекурсивный метод поиска всех перестановок

Метод рекурсивного перебора перестановок основан на идее сведения исходной задачи к аналогичной задаче на меньшем наборе входных данных.

Система рекуррентных соотношений, определяющих множество $Per(M)$ всех перестановок базового множества M произвольной природы:

$$Per(0) = \{""\},$$

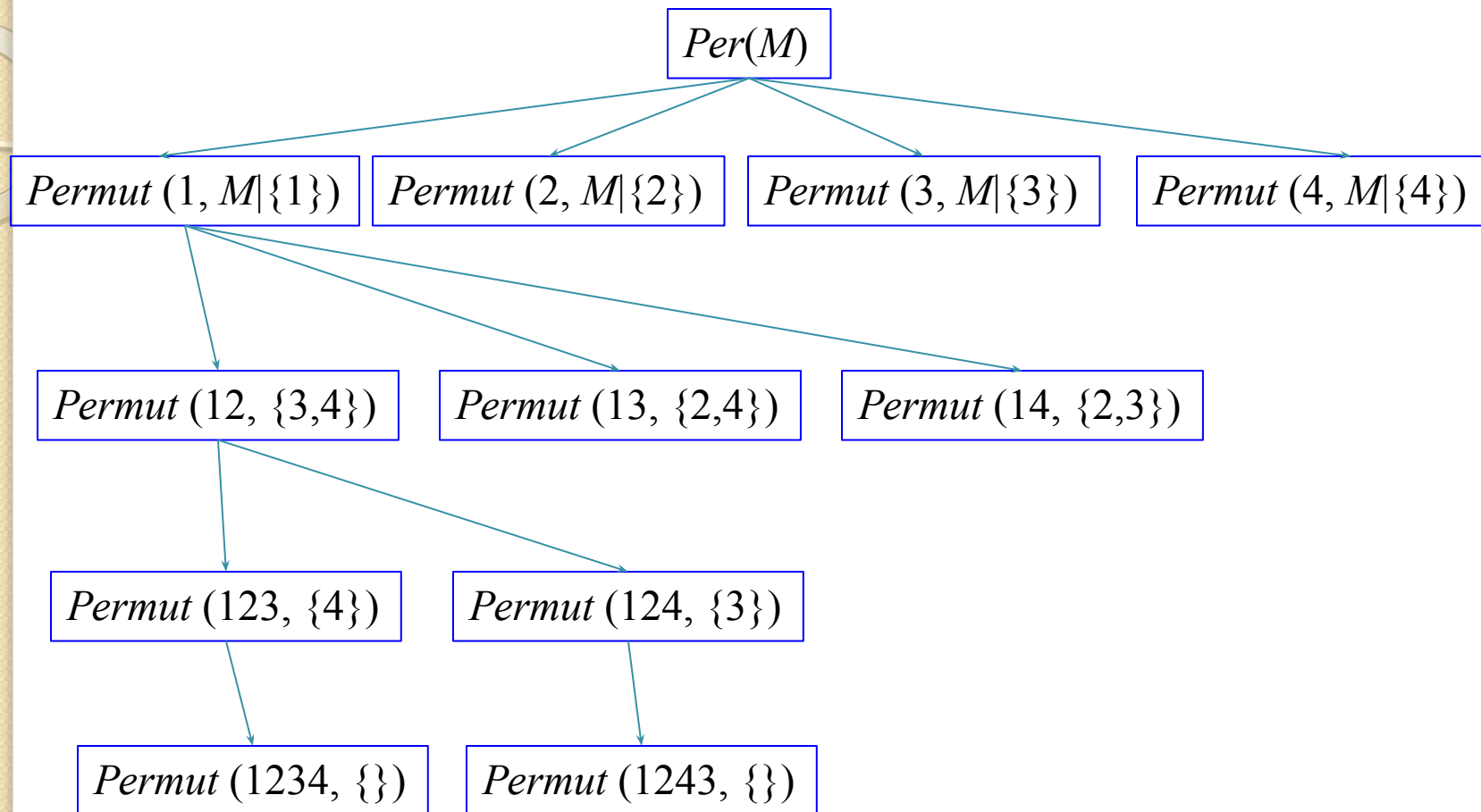
$$Per(M) = Permut(i, M \setminus \{i\}),$$

$$Permut(i, S) = \{ "i" + s \mid s \in Per(S) \}.$$

Первое равенство задает условие обрыва рекурсивного спуска: пустое множество элементов порождает пустую перестановку.

Два последних равенства определяют правила рекурсивного перехода.

Пример рекурсивного перебора для $M = \{1,2,3,4\}$



На языке Си этот процесс можно описать следующим образом:

```
typedef char string[256];
void permut(string start, string rest)
{
    int lenr = strlen(rest);
    int lens = strlen(start);
    int i=0; string s1=""; string s2="";
    if (lenr == 0) Printf("%s\n", start);
    else
    {
        for (i = 0; i < lenr; i++)
        {
            /* Добавляем i-ый символ к строке start */
            strcpy(s1, start);
            strncpy(s1+lens, rest+i, 1);
            strncpy(s1+lens+1, "\0", 1);
            /* Удаляем i-ый символ из строки rest */
            strncpy(s2, rest, i);
            strncpy(s2+i, rest+i+1, lenr-i-1);
            strncpy(s2+lenr-1, "\0", 1);
            /* Рекурсивный переход */
            permut( s1, s2 );
        }
    }
}
```

Генерация всех перестановок методом Кнута

Идея:

если построены все перестановки длины N , то для каждой такой перестановки можно построить $N+1$ перестановку длины $N+1$.

Пример:

Для перестановки **3241** можно построить 5 различных перестановок длины 5:

53241

35241

32541

32451

Генерация перестановок методом Кнута – I способ

Пусть дана перестановка длины N .

- Дописать в конец перестановки числа $(2i+1)/2$ ($0 \leq i \leq N$).
- Перенумеровать элементы полученных перестановок в порядке их возрастания.

Пример: дана перестановка **3241**.

3 2 4 1 0.5 → **4 3 5 2 1**

3 2 4 1 1.5 → **4 3 5 1 2**

3 2 4 1 2.5 → **4 2 5 1 3**

3 2 4 1 3.5 → **3 2 5 1 4**

3 2 4 1 4.5 → **3 2 4 1 5**

Генерация перестановок методом Кнута – 2 способ

Пусть дана перестановка длины N : $a_1 a_2 \dots a_N$.

- Дописать в конец перестановки числа k ($1 \leq k \leq N + 1$).
- Для всех $a_i \geq k$ заменить их на $a_i + 1$.

Пример: дана перестановка 3241.

3 2 4 1 1 \rightarrow 4 3 5 2 1

3 2 4 1 2 \rightarrow 4 3 5 1 2

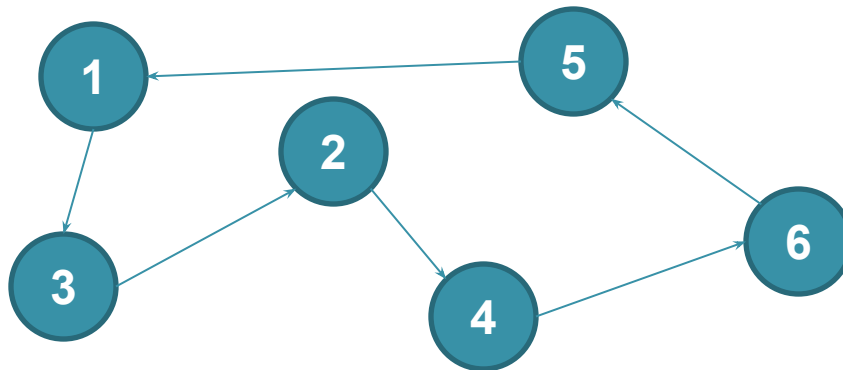
3 2 4 1 3 \rightarrow 4 2 5 1 3

3 2 4 1 4 \rightarrow 3 2 5 1 4

3 2 4 1 5 \rightarrow 3 2 4 1 5

Задача коммивояжера

Дано N городов. Необходимо объехать все, побывав в каждом городе только один раз и вернуться в исходный город, затратив при этом минимальное количество времени (денег, ...).



Решение: перебрать все последовательности городов, т.е. построив все перестановки элементов этого множества. Например, путь, отображенный на рисунке соответствует перестановке

1 3 2 4 6 5