Лекция 23 Алгоритмы перебора множеств

23.1. ***Постановка задачи***.

23.1.1. Перестановка некоторого набора элементов – это упорядоченная последовательность из этих элементов. Например, множество {1, 2, 3} имеет 6 различных перестановок (1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)

23.1.2. Для любого множества из n элементов существует ровно $n! = 1 \cdot 2 \cdot ... \cdot n$ различных перестановок.
В самом деле, пусть $P(n)$ – число перестановок n-элементного множества (для определённости рассмотрим множество $M = \{1, 2, ..., n\}$). На первом месте в перестановке может стоять одно из чисел 1, ..., n. Перестановок множества M, в которых на первом месте стоит число i ровно столько, сколько есть различных перестановок оставшихся n - 1 чисел, то есть $P(n-1)$. Поэтому, $P(n) = n \cdot P(n-1)$. По индукции получаем $P(n) = 1 \cdot 2 \cdot ... \cdot n = n!$

23.1.3. ***Для справок***: таблица $n!$ $n \cdot n!$ и $4 \cdot n \cdot n!$ для $10 \le n \le 14$.

n	$n!$	$n \cdot n!$	$4 \cdot n \cdot n!$
10	**3,628,800**	**36,288,000**	**145,152,000**
11	**39,916,800**	**439,117,800**	**1,758,471,200**
12	**479,001,600**	**5,748,019,200**	**22,992,076,800**
13	**6,227,020,800**	**80,951,270,400**	**323,805,081,600**
14	**87,178,291,200**	**1,220,496,076,800**	**4,881,984,307,200**

256 Gbyte = 274,877,906,944 Byte

Одна перестановка из n элементов занимает n байтов (если $0 \le n \le 255$), а все перестановки занимают $n \cdot n!$ байтов. Из таблицы видно, что при $n = 14$ все перестановки из n элементов не поместятся на жестком диске (винчестере) емкостью 256 Gbyte.

23.1.4. ***Задача*** состоит в том, чтобы написать программу, которая выводит все перестановки множества $\{1,2,..., n\}$ в лексикографическом порядке (перестановки можно рассматривать как слова в алфавите $B = \{1, 2, ..., n\}$)

23.2. ***Рекурсивный алгоритм генерации перестановок***.

23.2.1. ***Алгоритм***. Будем перебирать перестановки чисел $\{1, 2, ..., n\}$ в глобальном массиве $a[n]$, последовательно заполняя его числами 1, ..., n в различном порядке. *Для перебора всех перестановок применим ту же рекурсивную идею, что и при доказательстве формулы числа перестановок*: будем записывать на первое место в массиве a по очереди числа 1, ..., n, и для каждого числа рекурсивно вызывать функцию генерации перестановок оставшихся $n - 1$ чисел.

23.2.2. ***Си-программа***.

```
#include <stdio.h>
#include <stdlib.h>
```

```
void PrintPerm(int *a, int n) {
   int i;
   for(i = 0; i < n; i++) {
          printf("%3d", a[i]);
   }
   printf("\n");
}

void GenPerm(int *a, int *b, int i, int n) {
   if(i == n) {
     PrintPerm(a, n);
   }
   else {
     int j;
     for(j = 0; j < n; j++) {
          if(b[j] == 0) {
               b[j] = 1;
               a[i] = j + 1;
               GenPerm(a, b, i+1, n);
               b[j]  = 0;
          }
     }
   }
}

int main() {
   int *a, *b;
   int n;
   scanf("%d", &n);
   a = (int*)malloc(n*sizeof(int));
   b = (int*)malloc(n*sizeof(int));
   GenPerm(a, b, 0, n); /* первый вызов генератора */
   free(a);
   free(b);
   return 0;
}
```

23.2.3. Отметим, что рассмотренный рекурсивный алгоритм, который очень прост в реализации, достаточно эффективен (в соответствии с определением 23.3)

23.3. ***Определение***. *Переборный алгоритм называется* ***эффективным***, *если он работает время, которое растёт пропорционально суммарному размеру перебираемых им элементов, то есть произведению числа элементов на размер описания одного элемента.*

В случае перестановок объектов типа **char** каждая перестановка занимает n байт, так что произведение числа элементов на размер описания одного элемента равно $n \cdot n!$

В случае перестановок объектов типа **`int`** каждая перестановка занимает $4 \cdot n$ байт, так что произведение числа элементов на размер описания одного элемента равно $4 \cdot n \cdot n!$

23.4. ***Нерекурсивный алгоритм.*** Задачу посещения (перебора) всех перестановок заданного множества можно свести к следующей задаче: по данной перестановке сгенерировать следующую за ней перестановку (например, в лексикографическом порядке). Один из первых алгоритмов решения этой задачи придумал индийский математик Пандит Нарайана еще в XIV веке (он изучал свойства магических квадратов).

23.4.1. ***Алгоритм Нарайаны*** (*генерация лексикографически следующей перестановки*). Алгоритм Нарайаны по любой данной перестановке из n элементов $a_1\ a_2\ \ldots\ a_n$ генерирует следующую (в лексикографическом порядке) перестановку.
Если алгоритм Нарайаны применить в цикле к исходной последовательности n элементов $a_1\ a_2\ \ldots\ a_n$, отсортированных так, что $a_1 \le a_2 \le \ldots \le a_n$, то он сгенерирует все перестановки элементов множества $\{a_1\ a_2\ \ldots\ a_n\}$, в лексикографическом порядке. (Например, исходя из перестановки {1,2,2,3}, сгенерирует остальные перестановки в следующем порядке: 1223, 1232, 1322, 2123, 2132, 2213, 2231, 2312, 2321, 3122, 3212, 3221).

23.4.2. Сначала рассмотрим ***пример***. Пусть $n = 5$ и пусть элементами множества являются числа 1, 2, 3, 4 и 5. Рассмотрим перестановку 3241**1**5; лексикографически следующими будут: 32451, 32514, 32541, 34125, …

23.4.3. ***Алгоритм Нарайаны***.

Шаг 1. [Первая перестановка.] Составить перестановку $a_1\ a_2\ \ldots\ a_n$ $(a_1 < a_2 < \ldots < a_n)$.

Шаг 2. [Найти j, для которого $a_j < a_{j+1}$] Установить $j \leftarrow n - 1$. Если $a_j \ge a_{j+1}$, уменьшать j на 1 повторно, пока не выполнится условие $a_j < a_{j+1}$. Если окажется, что $j = 0$, завершить алгоритм. (На шаге 2 j является наименьшим индексом, для которого были посещены все перестановки, начиная с $a_1\ \ldots\ a_j$. Следовательно, лексикографически следующая перестановка увеличит значение a_j).

Шаг 3. [Увеличить a_j] Установить $l \leftarrow n$. Если $a_j \ge a_l$, уменьшать l на 1 повторно, пока не выполняется условие $a_j < a_l$. Затем поменять местами $a_j \leftrightarrow a_l$.
Пояснение. Поскольку $a_{j+1} \ge \ldots \ge a_n$, элемент a_l является наименьшим элементом, который больше a_j, и который может следовать за $a_1\ \ldots\ a_{j-1}$ в перестановке. Перед заменой выполнялись отношения $a_{j+1} \ge \ldots \ge a_{l-1} \ge a_l \ge a_j \ge a_{l+1} \ge \ldots \ge a_n$, а после замены – выполняются отношения $a_{j+1} \ge \ldots \ge a_{l-1} \ge a_j \ge a_l \ge a_{l+1} \ge \ldots \ge a_n$

Шаг 4. [Обратить $a_{j+1}\ \ldots\ a_n$] Установить $k \leftarrow j + 1$ и $l \leftarrow n$. Затем, если $k < l$, поменять местами $a_k \leftrightarrow a_l$, установить $k \leftarrow k + 1$, $l \leftarrow l - 1$ и повторять, пока не выполнится условие $k \ge l$. Вернуться к шагу 1.

23.4.4. ***Программа на языке Си***.

```
#include <stdio.h>
#include <stdlib.h>

int NextPerm(int *a, int n) {
      /* массив *a содержит одну из перестановок n чисел */
```

```
int i, k, t, tmp;
      /* находим k такое что: a[k] < a[k + 1] > .... > a[n - 1]*/
   for(k = n - 2; (a[k] > a[k + 1]) && (k >= 0); k--);
   if(k == -1) return 0; /* последняя перестановка */
      /* находим t > k такое, что среди a[k + 1],..., a[n - 1]
      */
      /* a[t] - минимальное число, большее a[k] */
   for(t = n - 1; (a[k] > a[t]) && (t >= k + 1); t--);

   tmp = a[k];
   a[k] = a[t];
   a[t] = tmp;
      /* участок массива a[k + 1],..., a[n - 1] записываем в */
      /* обратном порядке */
   for(i = k + 1; i <= (n + k)/2; i++) {
        t = n + k - i;
        tmp = a[i];
        a[i] = a[t];
        a[t] = tmp;
   }
   return i;
}

void PrintPerm(int *a, int n) {
   int i;
   for(i = 0; i < n; i++) {
          printf("%3d", a[i]);
   }
   printf("\n");
}

int main() {
   int *a, n, i;
   scanf("%d", &n);
   a = (int*)malloc(n*sizeof(int));

      /* a ← {1, 2, ..., n} */
   for(i = 0; i < n; i++) a[i] = i + 1;
   do {
        PrintPerm(a, n); }
   while (NextPerm(a, n));
   return 0;
}
```