**Перестановки**

Перестановка – массив (обозначается p)

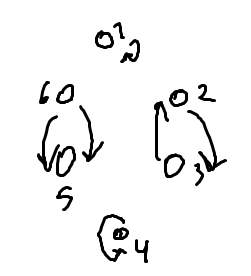
P = 1 3 2 4 6 5 (1 – n, все различные)

Нумерация: 0, n-1 (1, n)

Как проверить: все ли различные?

Сет, сайз и другие способы тоже норм

Перестановку можно представить как граф (точки взять за вершины):

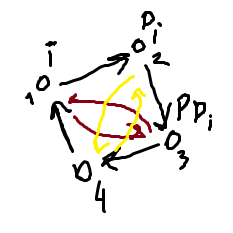


5 – 6 - двойной цикл (цикл длины 1)

4 и 1 – одиночный циклы (цикл длины 2)

Граф – набор циклов

P[i] – вершина исходящая из i



Бывают композиции перестановок

Композиция двух даст третью

**Циклы**

Значения: 1, 3, 2, 4, 6, 5

Номера: 1 2 3 4 5 6

Тогда циклы:

1

3 – 2 = 2 – 3 (2 уже принадлежит одному циклу)

4

6 – 5 = 5 – 6

Псевдо-код для нахождения всевозможных пар циклов:

cc = Vector<vector<int>>

used = vector<int>(n, 0)

for (int i = 0; I < n; ++i) {  
 if used(i){ used(i) – i уже имеет цикл, в который входит

Continue;

}

c // c – vector<int>()

v = i;

while c – ещё не занято циклом {

c.push\_back(v);

used(v) = true;

v = p[v]

}

cc.push\_back(c);

}

Кол-во перестановок равняется n!

Простой пример:

n = 4: \_ \_ \_ \_

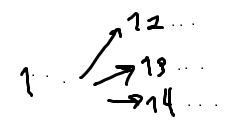
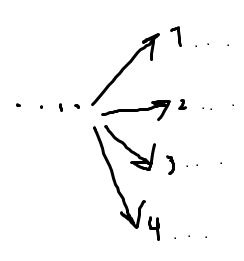
1 1 2 2

2 2 4

3 4 (кол-во всевозможных способов) = 4!

4

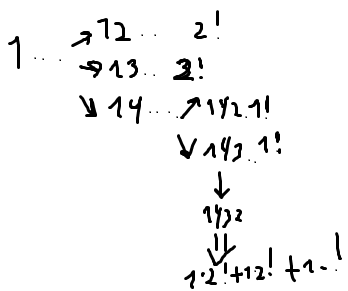
Сравнение и сортировка



Все перестановки на 1 автоматически меньше остальных

Как понять номер:

1. В каком блоке первая буква?
2. В какой части начинается вторая буква?
3. …



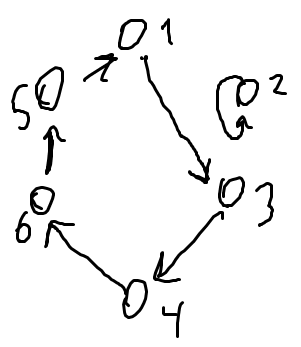
1 \* 2! + 1 \* 2! + 1 \* 1!

1 – (в нашем случае) кол-во пропущенных ветвей

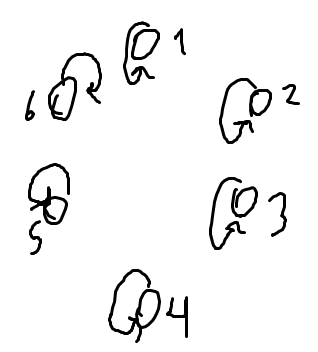
2!, 1! – кол-во вариантов в каждой из ветвей

Р: 3 2 4 6 1 5

Наша перестановка:



Нужно:



Свапнем двух рандомных челиков:

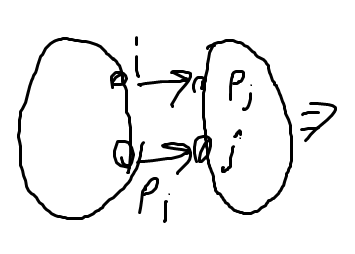
i j

pi pj

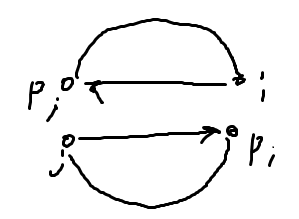
⇓ ⇓

pj pi

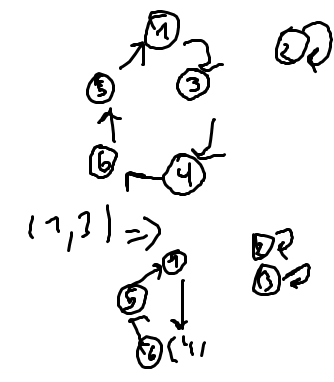
Два возможных пути:



* На один цикл меньше (цикл становится больше)



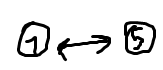
* На один цикл больше (чем больше, тем лучше, в конечно итоге придём к единичным циклам, что нам и нужно)



(1, 4)



(1, 6)



(1, 5) - ✔

for (int i = 0; I < n; ++i) {

while (p[i] != i) {

ans <- (I, p[i]);

j = p[i];

swap(p[i], p[j]);

}

}

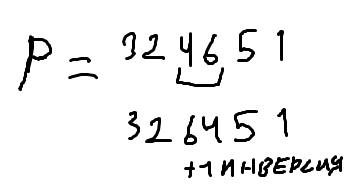
**Инверсия в массиве**

P = i < j: pi > pj

P = 1 2 3 4 (инверсий 0)

|p| = n

(n - 1) + (n - 2) + (n - 3) + … + 1 = n \*(n – 1) / 2



**ordercd\_set**

GNU C++ / MinGw

pbds library

мощно для нахождения инверсий (время: n \* log n)

**Генерация всех перестановок**

В плюсах есть функция:

bool next\_permutation(p.begin(), p.end())

применение do для вывода:

do {  
 cout << p << ‘\n’;

} while (next\_permutation(p.begin(), p.end()));