Математичка, ты довольна?
Часть 3.

Сапожников Денис

Contents

1 Комбинаторика

Комбинаторика – это раздел математики, который занимается подсчётом количества способов что-то выбрать.

Задача 1. Есть коробка шаров, в которой 5 красных и 3 белых шара. Сколько способов выбрать из коробки 2 белых и 1 красный шар?

Решение довольно простое: нам нужно выбрать *независимо* 1 красный шар и 2 белых. Один красный из 5 шаров можно выбрать 5 способами, два белых из трёх – тремя. Итого получается $5 \cdot 3 = 15$ способов.

1.1 Комбинаторные объекты и подсчёт их количества

Размещением из n элементов по k с повторением называется всякий набор из k элементов n-элементного множества. Легко понять, что таких наборов всего

$$\bar{A}_n^k = n^k$$

Пример 2. Сколько существует чисел в семеричной системе счисления длины не больше 5. Ответ: 7^5.

Перестановкой из n элементов (например чисел $1, 2, \ldots, n$) называется всякий упорядоченный набор из этих элементов. Тут тоже все достаточно просто: на первое место мы можем поставить любой из n элементов. На второе *независимо* любой из $n-1$ оставшихся и т.д. То есть получим $n(n-1) \times \ldots \times 1 = n!$ способов.

Размещением из n элементов по k без повторений (далее это будет по умолчанию, если не оговорено иное) называется **упорядоченный** набор из k различных элементов некоторого n-элементного множества. Как посчитать A_n^k — количество размещений из n по k? Выпишем все перестановки лексикографическом, то есть в «алфавитном» порядке:

$$\begin{array}{ccccc}
1 & \ldots & n-2 & n-1 & n \\
1 & \ldots & n-2 & n & n-1 \\
1 & \ldots & n-1 & n-2 & n \\
1 & \ldots & n-1 & n & n-2 \\
1 & \ldots & n & n-2 & n-1 \\
\vdots & & & & \vdots \\
2 & \ldots & n-2 & n-1 & n \\
\vdots & & & & \vdots \\
n & \ldots & 3 & 2 & 1
\end{array}$$

Мы хотим посчитать, а сколько есть различных префиксов длины k? Заметим, что у первых $(n-k)!$ строк одинаковые префиксы длины k (так как происходят изменения только в последних $n-k$ элементах). Аналогично для следующих $(n-k)!$ строк. Получается, что всего $n!$ строк и уникальных среди них можно вычислить по формуле:

$$A_n^k = \frac{n!}{(n-k)!}$$

Пример 3. Есть 10 различных домиков и 7 одинаковых котят. Сколько существует способов расселения котят по домикам? Ответ: A_{10}^7.

Сочетанием C_n^k из n по k называется набор k элементов, выбранных из данных n элементов. Наборы, отличающиеся только порядком следования элементов (но не составом), считаются одинаковыми, этим сочетания отличаются от размещений.

По аналогии с предыдущим пунктом мы можем выписать все перестановки, но теперь нам нужно ещё, чтобы первые k элементов тоже были различны. Легко убедиться, что получается следующая формула:

$$C_n^k = \frac{n!}{k!(n-k)!}$$

Пример 4. Теперь мы можем легко решить задачу 1 уже в терминах сочетаний (и обобщить): нам нужно выбрать 2 элемента из 3 и 1 из 5 независимо друг от друга, получим $C_3^2 \cdot C_5^1 = 15$.

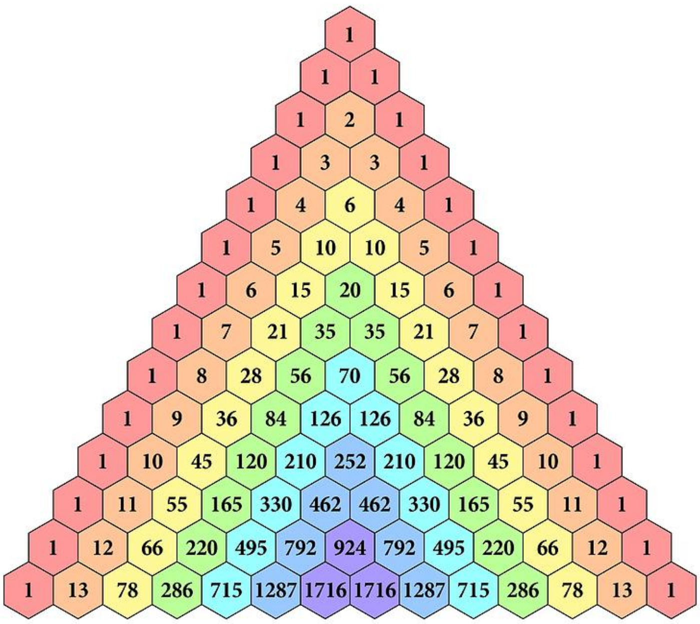
Сочетанием с повторением из n различных типов. Сколькими способами можно сделать из них комбинацию из k элементов, если не принимать во внимание порядок элементов внутри комбинации, а элементы одного типа могут повторяться.

Тут уже придётся применить комбинаторную идею: сделаем множество из n шаров и $k-1$ перегородки. Мы можем расставить из C_{n+k-1}^k способами, при этом на выходе будем получать k множеств и все способы будут различны. Таким образом, получаем формулу

$$\bar{C}_n^k = C_{n+k-1}^k$$

1.2 Сочетания

Сочетания чаще всего встречаются в задачах, потому на них стоит остановиться подольше.
Прямой проверкой можно показать формулу:
$C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$
На основании этой формулы можно составить так называемый треугольник Паскаля:



k-й элемент в n-й строке — это C_n^k, а сам элемент равен сумме двух соседних элементов, стоящих строкой выше.

Но более того, давайте рассмотрим сумму $(a+b)^n$ — это называется бином Ньютона. Мы можем раскрыть скобки и получим сумму $(a+b)^n = (a+b)(a+b)\ldots(a+b) = \sum\limits_{k=0}^{n} C_n^k a^k b^{n-k}$.

Ещё интересный факт: сумма элементов в строке n равна 2^n:

$$C_n^0 + C_n^1 + \ldots + C_n^n = 2^n$$

Это легко доказать, используя комбинаторное рассуждение. Мы считаем количество способов выбрать из n элементов $0, 1, 2, \ldots, n$, то есть все возможные подмножества, получая таким образом все возможные варианты выбрать люблe подмножество из n-элементного множества, а всего подмножеств 2^n.

Ещё пара полезных формул:
$C_n^k = C_n^{n-k}$ – прямая проверка.
$C_n^k + C_{n-1}^k + \ldots + C_k^k = C_{n+1}^{k+1}$ – подумайте над комбинаторным смыслом.

1.3 Задачи

1. Сколькими способами можно расставить n единиц и m нулей так, чтобы никакие 2 единицы не стояли рядом?

   Выпишем m нулей в ряд. На любое из мест между нулями, а так же по краям можно поставить единицу, таким образом получили C_{m+1}^n способов.

2. На окружности отмечены n точек. Сколько существует многоугольников, вершинами которых является подмножество отмеченных точек? Сколько выпуклых из них?

   Каждый m-угольник определяется выбором m точек из n, взятых в определённом порядке, при чём циклическая перестановка и изменения обхода не меняет многоугольник, потому различных m-угольников $\frac{1}{2m}A_n^m$, а всего их $\sum\limits_{m=3}^{n} \frac{1}{2m}A_n^m$. Число выпуклых многоугольников равно $\sum\limits_{m=3}^{n} \frac{1}{2m}C_n^m$.

1.4 Генерация комбинаторных объектов

Мы поговорили о комбинаторных объектах, посчитали их, даже задачи порешали, осталось научиться их генерировать.

1.4.1 Перестановки

Генерировать все перестановки с ходу кажется сложным. Вместо этого научимся по перестановке генерировать следующую перестановку. В C++ уже есть встроенная функция `std::next_permutation (itbegin, itend)`, лежащая в <algorithm>, которая возвращает false, если следующей перестановки нет, а иначе получает следующую перестановку.

Чтобы понять, как получается следующая перестановка, напишем какой-нибудь пример: $(1, 5, 3, 6, 4, 2)$ и $(1, 5, 4, 2, 3, 6)$. На самом деле мы уже видим, как работает функция: мы берём первый элемент после возрастающей с конца последовательности, находим первый элемент, который меньше него в последовательности, свопаем и ревёрсим.

```
bool next_permutation(vector<int> &p) {
  int n = p.size();
  int pos = n - 2;
  for (; pos >= 0; pos--)
```

```
    if (a[pos] < a[pos + 1])
      break;
  if (i == -1)
    return false;
  for (int i = n - 1; i >= 0; i--)
    if (a[pos] > a[i]) {
      swap(a[pos], a[i]);
      break;
    }
  reverse(p.begin() + pos, p.end());
  return true;
}
```

1.4.2 Сочетания

У нас была хорошая формула $C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$, которая подсказывает, как можно генерировать все сочетания. Она говорит, что в одном случае давайте возьмём элемент n и тогда решим задачу меньше, а в другом случае – не возьмём и опять перейдём к задаче меньше.

```
int p[N]; // 1-indexed
void combination(int n, int k) {
  if (k > n || k < 0)
    return;
  if (n == 0) {
    print();
    return;
  }
  p[k] = n;
  combination(n - 1, k - 1);
  combination(n - 1, k);
}
```

А ещё иногда в задачах просят научиться генерировать сочетания, которые отличаются ровно в одном элементе. Мы на самом деле почти это сделали, нужно лишь поставить вызовы рекурсии в нужном порядке и отреверсить одну часть. В общем то, формула у нас будет такая: $\{C_n^k\} = \{C_{n-1}^k\} \cup \{C_{n-1}^{k-1} \cup n\}^R$

Можно либо написать это и ревёрсить ответ, либо изощратся в порядке, приведём код второго способа.

```
bool used[N];

void gen(int n, int k, int l, int r, bool rev, int old_n) {
  if (k > n || k < 0)
    return;
  if (n == 0) {
    for (int i = 0; i < old_n; ++i)
      if (used[i])
        cout << ans[i] << ' ';
    cout << '\n';
    return;
  }
  used[rev?r:l] = false;
  gen(n-1, k, !rev?l+1:l, !rev?r:r-1, rev, old_n);
  used[rev?r:l] = true;
```

```
  gen(n-1, k-1, !rev?l+1:l, !rev?r:r-1, !rev, old_n);
}

gen(n, k, 0, n-1, false, n);
```

1.4.3 Разбиения

Есть ещё один комбинаторный объект, о котором можно поговорить — это разбиение числа, то есть разложение его на сумму нескольких строго положительных слагаемых, обозначение: n_λ. Генерировать все разбиения тоже очень просто: будем генерировать их в лексикографически убывающем порядке.

```
int top = 0;
int p[N];

void gen(int n, int last = n) {
  if (n == 0) {
    print();
    return;
  }
  top++;
  for (int i = last; i >= 1; i--) {
    p[top - 1] = i;
    gen(n - i, i);
  }
}
```

1.5 Литература

1. «Комбинаторика для программистов» В. Липсицкий
2. «Комбинаторика» Виленкин
3. http://e-maxx.ru/algo/
4. Раздел «комбинаторика» на http://neerc.ifmo.ru/wiki/