Модули на ACM

Владимир Милосердов, Владимир Шабанов

21 октября 2014 г.

Оглавление

Глава 1

Геометрия

Глава 2

Графы

Граф - множество вершин и ребер(заданных явно или не явно). Понятия используемые в дальнейшем:

- Ациклический граф

  Граф без циклов

- Влентность/Степень вершины

  Количество ребер входящих/выходящих в вершину

- Взвешенный граф

  Граф в котором у каждого ребра есть стоимость

- Висячая вершина

  Вершина со степенью один

- Гамильтонов путь

  Путь в графе содержащий каждую вершину ровно один раз

- Гамильтонов цикл

  Цикл содержащий каждую вершину ровно один раз

- Компонента связности

  Множество вершин и ребер графа такое, что из каждой его вершины достижима любая другая вершина этого множества

- Компонента сильной связности

  Множество вершин и ребер ориентированного графа такое, что из каждой его вершины достижима любая другая вершина этого множества

- Кратные ребра

  Ребра связывающие одну и ту же пару вершин

- Минимальный каркас

  Множество ребер соеденяющих все вершины графа без циклов и имеющее минимальный суммарный вес

- Паросочетания

  Множество попарно не смежных ребер

- Точка сочленения

  Вершина после удаления которой количество компонент связности возрастает

- Эйлеров путь

  Путь в графе содержащий каждое ребро ровно один раз

- Эйлеров цикл

  Цикл содержащий каждое ребро ровно один раз

Глава 3

Строки

Глава 4

ДП

Глава 5

Алгебра

5.1 Алгоритм Евклида

5.1.1 НОД, НОК (gcd, lcm)

$$gcd(a,b) = \begin{cases} a & \text{если } a = 0 \\ b & \text{иначе} \end{cases}$$
$$lcm(a,b) = \frac{a \cdot b}{gcd(a,b)}$$

5.1.2 Расширенный алгоритм Евклида

```
1  int gcdex (int a, int b, int & x, int & y) {
2      if (a == 0) {
3          x = 0; y = 1;
4          return b;
5      }
6      int x1, y1;
7      int d = gcd (b%a, a, x1, y1);
8      x = y1 - (b / a) * x1;
9      y = x1;
10     return d;
11 }
```

Функция возвращает нод и коеффициенты x, y по сслыкам

5.1.3 Восстановление в кольце по модулю

```
1 int x, y;
2 int g = gcdex (a, m, x, y);
3 if (g != 1)
4     cout << "no␣solution";
5 else {
6     x = (x % m + m) % m;
7     cout << x;
8 }
```

5.2 Решето Эратосфена

5.2.1 Классический вариант

Дано число n. Требуется найти все простые в отрезке $[2; n]$. Решето Эратосфена решает эту задачу за $O(n \log \log n)$
Запишем ряд чисел $1 \dots n$, и будем вычеркивать сначала все числа, делящиеся на 2, кроме самого числа 2, затем деляющиеся на 3, кроме самого числа 3, затем на 5 и так далее ...

```
int n;
vector<char> prime (n+1, true);
prime[0] = prime[1] = false;
for (int i=2; i<=n; ++i)
    if (prime[i])
        if (i * 1ll * i <= n)
            for (int j=i*i; j<=n; j+=i)
                prime[j] = false;
```

5.2.2 Линейное решето

Чуть быстрее по времени, чем классическое $O(N)$. Цена вопроса - оверхед по памяти. Пусть $lp[i]$ – минимальный простой делитель числа $i, 2 \le i \le n$.

- $lp[i] = 0$ – число i – простое
- $lp[i] \neq 0$ – число i – составное

```
const int N = 10000000;
int lp[N+1];
vector<int> pr;

for (int i=2; i<=N; ++i) {
    if (lp[i] == 0) {
        lp[i] = i;
        pr.push_back (i);
    }
    for (int j=0; j<(int)pr.size() && pr[j]<=lp[i] && i*pr[j]<=N; ++j)
        lp[i * pr[j]] = pr[j];
}
```

Вектор $lp[\ ]$ можно как-то использовать для факторизации чисел

5.3 Разбор выражений

Дано выражение. Начинаем парсить его функцией E1, которая обрабатывает самые низкоприоритетные операции (в нашем случае '+', '-').

```
def E1():
    res = E2()
    while True:
        c = getc()
        if c == '+':
            res += E2()
```

```
7           elif c == '-':
8               res -= E2()
9           else:
10              putc(c)
11              return res
```

Сначала происходит обработка атома, стоящего слева от знака ('+', '-'), затем обработка каждого атома между знаками. Обработку этих атомов производит функция $E2()$. Это функция более низкого ранга, которая обрабатывает более приоритетные операции (в нашем случае '*' и '/').

```
1  def E2():
2      res = E3()
3      while True:
4          c = getc()
5          if c == '*':
6              res *= E3()
7          elif c == '/':
8              res /= E3()
9          else:
10             putc(c)
11             return res
```

Функция работает аналогично $E1()$. Таким образом мы можем поддерживать сколько угодно операций различных приоритетов, добавляя функции.
Перейдем к обработке скобок:

```
1  def E3():
2      if c == '(':
3          res = E1()
4          c = getc()
5          return res
6      else:
7          putc(c)
8          return E4()
```

Берём символ, если он скобка, тогда обрабатываем выражение внутри и считываем закрывающую скобку.
Теперь рассмотрим считывание числа. Ничего сложного:

```
1  def E4():
2      res = 0
3      while True:
4          c = getc()
5          if str.isdigit(c):
6              res = res * 10 + int(c)
7          else:
8              putc(c)
9              return res
```

Глава 6

Разное