# Standard Code Library

ONGLU

North Eastern University

August 2021

# Contents

# 初始化

## 数据结构

### ST 表

### 二维哈希

```
1  ull hs[109][109], pw1[10009], pw2[100009];
2  ull gethash(int lx, int ly, int rx, int ry) {
3      ull hs1 = hs[lx][ly] - pw2[ry - ly + 1] * hs[lx][ry + 1];
4      ull hs2 = hs[rx + 1][ly] - pw2[ry - ly + 1] * hs[rx + 1][ry + 1];
5      return hs1 - pw1[rx - lx + 1] * hs2;
6  }
7  pw1[0] = pw2[0] = 1;
8  for(int i = 1; i <= 1000; i++) pw1[i] = pw1[i - 1] * 19260817;
9  for(int i = 1; i <= 1000; i++) pw2[i] = pw2[i - 1] * 135;
10 for(int i = n; i >= 1; i--) {
11     for(int j = 1; j <= m; j++) {
12         if(i == n) hs[i][j] = sum[i][j] + 2;
13         else hs[i][j] = hs[i + 1][j] * 19260817 + sum[i][j] + 2;
14     }
15 }
16 for(int i = 1; i <= n; i++) {
17     for(int j = m - 1; j; j--) {
18         hs[i][j] = hs[i][j + 1] * 135 + hs[i][j];
19     }
20 }
```

### 轻重链剖分

```
1  void dfs1(int x, int pre) {
2      siz[x] = 1; mson[x] = 0;
3      dth[x] = dth[pre] + 1;
4      fa[x] = pre;
5      for(auto y : son[x]) if(y != pre) {
6          dfs1(y, x);
7          siz[x] += siz[y];
8          if(!mson[x] || siz[y] > siz[mson[x]])
9              mson[x] = y;
10     }
11 }
12 void dfs2(int x, int pre, int ntp) {
13     id[x] = ++idcnt;
14     ltp[x] = ntp;
15     if(mson[x]) dfs2(mson[x], x, ntp);
16     for(auto y : son[x]) {
17         if(y == mson[x] || y == pre) continue;
18         dfs2(y, x, y);
19     }
20 }
21 void link_modify(int x, int y, int z) {
22     z %= mod;
23     while(ltp[x] != ltp[y]) {
24         dth[ltp[x]] < dth[ltp[y]] && (x ^= y ^= x ^= y);
25         modify(1, n, id[ltp[x]], id[x], 1, z);
26         x = fa[ltp[x]];
27
28     }
29     dth[x] < dth[y] && (x ^= y ^= x ^= y);
30     modify(1, n, id[y], id[x], 1, z);
31 }
32 int link_query(int x, int y) {
33     int ans = 0;
34     while(ltp[x] != ltp[y]) {
35         dth[ltp[x]] < dth[ltp[y]] && (x ^= y ^= x ^= y);
36         ans = (1ll * ans + query(1, n, id[ltp[x]], id[x], 1)) % mod;
37         x = fa[ltp[x]];
38     }
```

```
39        dth[x] < dth[y] && (x ^= y ^= x ^= y);
40        ans = (1ll * ans + query(1, n, id[y], id[x], 1)) % mod;
41        return ans;
42    }
```

## 线段树合并

搞个动态开点线段树出来

```
1    #define mval(x) tree[x].mval
2    #define mpos(x) tree[x].mpos
3    #define lson(x) tree[x].lson
4    #define rson(x) tree[x].rson
5    struct node {
6        int mpos, mval, lson, rson;
7    } tree[N * 50];
8    void update(int rt) {
9        if(mval(lson(rt)) >= mval(rson(rt))) {
10           mval(rt) = mval(lson(rt));
11           mpos(rt) = mpos(lson(rt));
12       } else {
13           mval(rt) = mval(rson(rt));
14           mpos(rt) = mpos(rson(rt));
15
16       }
17   }
18   void modify(int l, int r, int x, int v, int &rt) {
19       if(!rt) rt = ++idtot;
20       if(l == r) {
21           mval(rt) += v;
22           mpos(rt) = l;
23           return ;
24       }
25       if(x <= Mid) modify(l, Mid, x, v, lson(rt));
26       else modify(Mid + 1, r, x, v, rson(rt));
27       update(rt);
28   }
29   int merge(int l, int r, int rt1, int rt2) {
30       if(!rt1 || !rt2) return rt1 + rt2;
31       if(l == r) {
32           mval(rt1) += mval(rt2);
33           mpos(rt1) = l;
34           return rt1;
35       }
36       lson(rt1) = merge(l, Mid, lson(rt1), lson(rt2));
37       rson(rt1) = merge(Mid + 1, r, rson(rt1), rson(rt2));
38       update(rt1);
39       return rt1;
40   }
```

## 二维树状数组

- 矩阵修改, 矩阵查询

  查询前缀和公式:

  令 $d[i][j]$ 为差分数组, 定义 $d[i][j] = a[i][j] - (a[i-1][j] - a[i][j-1] - a[i-1][j])$

  $\sum_{i=1}^{x} \sum_{j-1}^{y} a[i][j] = (x+1) * (y+1) * d[i][j] - (y+1) * i * d[i][j] + d[i][j] * i * j$

```
1    void modify(int x, int y, int v) {
2        for(int rx = x; rx <= n; rx += rx & -rx) {
3            for(int ry = y; ry <= m; ry += ry & -ry) {
4                tree[rx][ry][0] += v;
5                tree[rx][ry][1] += v * x;
6                tree[rx][ry][2] += v * y;
7                tree[rx][ry][3] += v * x * y;
8            }
9        }
10   }
```

```
11    void range_modify(int x, int y, int xx, int yy, int v) {
12        modify(xx + 1, yy + 1, v);
13        modify(x, yy + 1, -v);
14        modify(xx + 1, y, -v);
15        modify(x, y, v);
16    }
17    int query(int x, int y) {
18        int ans = 0;
19        for(int rx = x; rx; rx -= rx & -rx) {
20            for(int ry = y; ry; ry -= ry & -ry) {
21                ans += (x + 1) * (y + 1) * tree[rx][ry][0]
22                - tree[rx][ry][1] * (y + 1) - tree[rx][ry][2] * (x + 1)
23                + tree[rx][ry][3];
24            }
25        }
26        return ans;
27    }
28    int range_query(int x, int y, int xx, int yy) {
29        return query(xx, yy) + query(x - 1, y - 1)
30            - query(x - 1, yy) - query(xx, y - 1);
31    }
```

## 平衡树

- luogu P3369 【模板】普通平衡树

```
1    #define val(x) tree[x].val
2    #define cnt(x) tree[x].cnt
3    #define siz(x) tree[x].siz
4    #define fa(x) tree[x].fa
5    #define son(x, k) tree[x].ch[k]
6    struct Tree {
7        struct node {
8            int val, cnt, siz, fa, ch[2];
9        } tree[N];
10       int root, tot;
11       int chk(int x) {
12           return son(fa(x), 1) == x;
13       }
14       void update(int x) {
15           siz(x) = siz(son(x, 0)) + siz(son(x, 1)) + cnt(x);
16       }
17       void rotate(int x) {
18           int y = fa(x), z = fa(y), k = chk(x), w = son(x, k ^ 1);
19           son(y, k) = w; fa(w) = y;
20           son(z, chk(y)) = x; fa(x) = z;
21           son(x, k ^ 1) = y; fa(y) = x;
22           update(y); update(x);
23       }
24       void splay(int x, int goal = 0) {
25           while(fa(x) != goal) {
26               int y = fa(x), z = fa(y);
27               if(z != goal) {
28                   //双旋
29                   if(chk(y) == chk(x)) rotate(y);
30                   else rotate(x);
31               }
32               rotate(x);
33           }
34           if(!goal) root = x;
35       }
36       int New(int x, int pre) {
37           tot++;
38           if(pre) son(pre, x > val(pre)) = tot;
39           val(tot) = x; fa(tot) = pre;
40           siz(tot) = cnt(tot) = 1;
41           son(tot, 0) = son(tot, 1) = 0;
42           return tot;
43       }
```

```
44      void Insert(int x) {
45          int cur = root, p = 0;
46          while(cur && val(cur) != x) {
47              p = cur;
48              cur = son(cur, x > val(cur));
49          }
50          if(cur) cnt(cur)++;
51          else cur = New(x, p);
52          splay(cur);
53      }
54      void Find(int x) {
55          if(!root) return ;
56          int cur = root;
57          while(val(cur) != x && son(cur, x > val(cur)))
58              cur = son(cur, x > val(cur));
59          splay(cur);
60      }
61      int Pre(int x) {
62          Find(x);
63          if(val(root) < x) return root;
64          int cur = son(root, 0);
65          while(son(cur, 1))
66              cur = son(cur, 1);
67          return cur;
68      }
69      int Succ(int x) {
70          Find(x);
71          if(val(root) > x) return root;
72          int cur = son(root, 1);
73          while(son(cur, 0))
74              cur = son(cur, 0);
75          return cur;
76      }
77      void Del(int x) {
78          int lst = Pre(x), nxt = Succ(x);
79          splay(lst); splay(nxt, lst);
80          int cur = son(nxt, 0);
81          if(cnt(cur) > 1) cnt(cur)--, splay(cur);
82          else son(nxt, 0) = 0, splay(nxt);
83      }
84      int Kth(int k) {
85          int cur = root;
86          while(1) {
87              if(son(cur, 0) && siz(son(cur, 0)) >= k) cur = son(cur, 0);
88              else if(siz(son(cur, 0)) + cnt(cur) >= k) return cur;
89              else k -= siz(son(cur, 0)) + cnt(cur), cur = son(cur, 1);
90          }
91      }
92  } T;
```

## K-D Tree

用方差最大的那一维坐标作为当前的划分点集，然后选取该维度的中位数点划分成左右两个点集。

```
1   #include <bits/stdc++.h>
2   #define pt(x) cout << x << endl;
3   #define Mid ((l + r) / 2)
4   #define low(x, k) tree[x].low[k]
5   #define high(x, k) tree[x].high[k]
6   #define lson(x) tree[x].lson
7   #define rson(x) tree[x].rson
8   using namespace std;
9   int read() {
10      char c; int num, f = 1;
11      while(c = getchar(),!isdigit(c)) if(c == '-') f = -1; num = c - '0';
12      while(c = getchar(), isdigit(c)) num = num * 10 + c - '0';
13      return f * num;
14  }
15  const int N = 5e5 + 1009;
16
17  namespace KD_Tree{
```

```
18
19     const int dimension = 2;
20     struct node {
21         int lson, rson;
22         int low[dimension], high[dimension];
23     } tree[N];
24     struct Point {
25         int id;
26         int v[dimension];
27     } p[N];
28     void update(int rt) {
29         for(int i = 0; i < dimension; i++) {
30             low(rt, i) = high(rt, i) = p[rt].v[i];
31             if(lson(rt)) {
32                 low(rt, i) = min(low(rt, i), low(lson(rt), i));
33                 high(rt, i) = max(high(rt, i), high(lson(rt), i));
34             }
35             if(rson(rt)) {
36                 low(rt, i) = min(low(rt, i), low(rson(rt), i));
37                 high(rt, i) = max(high(rt, i), high(rson(rt), i));
38             }
39
40         }
41     }
42     int build(int l, int r) {
43         if(l > r) return 0;
44         double av[dimension] = {0};
45         double va[dimension] = {0};
46         for(int i = 0; i < dimension; i++)
47             low(Mid, i) = high(Mid, i) = p[Mid].v[i];
48         for(int i = l; i <= r; i++)
49             for(int j = 0; j < dimension; j++)
50                 av[j] += p[i].v[j];
51         for(int i = 0; i < dimension; i++)
52             av[i] /= (double) (r - l + 1);
53         for(int i = l; i <= r; i++)
54             for(int j = 0; j < dimension; j++)
55                 va[j] += (p[i].v[j] - av[j]) * (p[i].v[j] - av[j]);
56         int maxdi = 0;
57         for(int i = 1; i < dimension; i++)
58             if(va[i] > va[maxdi])
59                 maxdi = i;
60         nth_element(p + l, p + Mid, p + 1 + r, [maxdi](const Point &a, const Point &b) -> int{return a.v[maxdi] <
   ↪ b.v[maxdi];});
61         lson(Mid) = build(l, Mid - 1);
62         rson(Mid) = build(Mid + 1, r);
63         update(Mid);
64         return Mid;
65     }
66     int isIn(const Point &a, const Point &ld, const Point &ru) {
67         for(int i = 0; i < dimension; i++)
68             if(a.v[i] < ld.v[i] || a.v[i] > ru.v[i])
69                 return false;
70         return true;
71     }
72     void debug(int rt, int l, int r) {
73         if(l > r) return ;
74         printf("%d\n", p[rt].id);
75         debug(lson(rt), l, Mid - 1);
76         debug(rson(rt), Mid + 1, r);
77
78     }
79     //只能处理二维
80     void getNodeset(int rt, int l, int r, vector<int> &v, const Point &ld, const Point &ru) {
81         if(l > r) return ;
82         for(int i = 0; i < dimension; i++) {
83             if(low(rt, i) > ru.v[i] || high(rt, i) < ld.v[i]) {
84                 return ;
85             }
86         }
87         if(isIn(p[Mid], ld, ru))
```

```
88              v.push_back(p[Mid].id);
89          getNodeset(lson(rt), l, Mid - 1, v, ld, ru);
90          getNodeset(rson(rt), Mid + 1, r, v, ld, ru);
91      }
92  }
93  using namespace KD_Tree;
94  int n, q, root;
95  signed main()
96  {
97      n = read();
98      for(int i = 1; i <= n; i++) {
99          p[i].v[0] = read();
100         p[i].v[1] = read();
101         p[i].id = i - 1;
102     }
103     root = build(1, n);
104     q = read();
105     for(int i = 1; i <= q; i++) {
106         int x = read(), xx = read();
107         int y = read(), yy = read();
108         Point ld, ru;
109         ld.v[0] = x; ld.v[1] = y;
110         ru.v[0] = xx; ru.v[1] = yy;
111         vector<int> v;
112         v.clear();
113         getNodeset(root, 1, n, v, ld, ru);
114         sort(v.begin(), v.end());
115         for(auto x : v)
116             printf("%d\n", x);
117         printf("\n");
118     }
119     return 0;
120 }
```

## 可持久化数据结构

### 可持久化 Trie

```
1   namespace Trie {
2       struct node {
3           int ch[2], ed, siz;
4       } tree[N * 40];
5       int tot = 0;
6       int _new() {
7           tot++;
8           tree[tot].ch[0] = 0;
9           tree[tot].ch[1] = 0;
10          tree[tot].ed = tree[tot].siz = 0;
11          return tot;
12      }
13      void init() {
14          tot = 0;
15          rt[0] = _new();
16      }
17      int Insert(int x, int t, int i = 15) {
18          int u = _new(), f = (x >> i) & 1;
19          tree[u] = tree[t];
20          if(i == -1) {
21              ed(u)++;
22              siz(u)++;
23              return u;
24          }
25          son(u, f) = Insert(x, son(t, f), i - 1);
26          siz(u) = siz(son(u, 0)) + siz(son(u, 1));
27          return u;
28      }
29      void print(int u, int now) {
30          if(u == 0) return ;
31          for(int i = 1; i <= ed(u); i++) printf("%d ", now);
32          if(son(u, 0)) print(son(u, 0), now * 2);
33          if(son(u, 1)) print(son(u, 1), now * 2 + 1);
```

```
34         }
35     int query(int u1, int u2, int x, int i = 15, int now = 0) {
36         if(i == -1) return now;
37         int f = (x >> i) & 1;
38         if(siz(son(u1, f ^ 1)) - siz(son(u2, f ^ 1)) > 0)
39             return query(son(u1, f ^ 1), son(u2, f ^ 1), x, i - 1, now * 2 + (f ^ 1));
40         else return query(son(u1, f), son(u2, f), x, i - 1, now * 2 + (f));
41     }
42 }
```

## 主席树（静态第 k 小）

建立权值树，那么 $[l, r]$ 的区间权值树就是第 $r$ 个版本减去第 $l - 1$ 个版本的树。

```
1  #include <iostream>
2  #include <cstdio>
3  #include <algorithm>
4  #include <cmath>
5  #include <assert.h>
6  #define Mid ((l + r) / 2)
7  #define lson (rt << 1)
8  #define rson (rt << 1 | 1)
9  using namespace std;
10 int read() {
11     char c; int num, f = 1;
12     while(c = getchar(),!isdigit(c)) if(c == '-') f = -1; num = c - '0';
13     while(c = getchar(), isdigit(c)) num = num * 10 + c - '0';
14     return f * num;
15 }
16 const int N = 1e7 + 1009;
17 const int M = 2e5 + 1009;
18 struct node {
19     int ls, rs, v;
20 } tree[N];
21 int tb;
22 int n, m, tot, a[M], b[M], rt[M];
23 int _new(int ls, int rs, int v) {
24     tree[++tot].ls = ls;
25     tree[tot].rs = rs;
26     tree[tot].v = v;
27     return tot;
28 }
29 void update(int rt) {
30     tree[rt].v = tree[tree[rt].ls].v + tree[tree[rt].rs].v;
31 }
32 int build(int l, int r) {
33     if(l == r) return _new(0, 0, 0);
34     int x = _new(build(l, Mid), build(Mid + 1, r), 0);
35     update(x);
36     return x;
37 }
38 int add(int l, int r, int p, int rt, int v) {
39     int x = ++tot;
40     tree[x] = tree[rt];
41     if(l == r) {
42         tree[x].v += v;
43         return x;
44     }
45     if(p <= Mid) tree[x].ls = add(l, Mid, p, tree[x].ls, v);
46     else tree[x].rs = add(Mid + 1, r, p, tree[x].rs, v);
47     update(x);
48     return x;
49 }
50 int query(int l, int r, int rt1, int rt2, int k) {
51     if(l == r) return l;
52     if(k <= tree[tree[rt1].ls].v - tree[tree[rt2].ls].v) return query(l, Mid, tree[rt1].ls, tree[rt2].ls, k);
53     else return query(Mid + 1, r, tree[rt1].rs, tree[rt2].rs, k - (tree[tree[rt1].ls].v - tree[tree[rt2].ls].v));
54 }
55 void Debug(int l, int r, int rt) {
56     printf("%d %d %d\n", l, r, tree[rt].v);
57     if(l == r) return ;
```

```
58        Debug(l, Mid, tree[rt].ls);
59        Debug(Mid + 1, r, tree[rt].rs);
60    }
61    signed main()
62    {
63        n = read(); m = read();
64        for(int i = 1; i <= n; i++) a[i] = b[i] = read();
65        sort(b + 1, b + 1 + n);
66        tb = unique(b + 1, b + 1 + n) - b - 1;
67        rt[0] = build(1, tb);
68        for(int i = 1; i <= n; i++) {
69            rt[i] = add(1, tb, lower_bound(b + 1, b + 1 + tb, a[i]) - b, rt[i - 1], 1);
70        }
71        for(int i = 1; i <= m; i++) {
72            int l, r, k;
73            l = read(); r = read(); k = read();
74            assert(r - l + 1 >= k);
75            printf("%d\n", b[query(1, tb, rt[r], rt[l - 1], k)]);
76        }
77        return 0;
78    }
```

## cdq 分治三维偏序

先按照第一维排序, 然后对第二维归并, 归并时计算左对右的贡献, 先双指针, 满足当前统计出的第二维都有序

```
1    const int N = 1e6 + 1009;
2    struct node{
3        int x, y, z, id, cnt;
4    }a[N], tmp[N];
5    bool operator ==(const node &a, const node &b) {
6        return a.x == b.x && a.y == b.y && a.z == b.z;
7    }
8    int n, m, tot, ans[N], tt[N], tree[N];
9    int ttt[N];
10   bool cmp(node a, node b) {
11       if(a.x == b.x && a.y == b.y) return a.z < b.z;
12       if(a.x == b.x) return a.y < b.y;
13       return a.x < b.x;
14   }
15   void add(int x, int y) {
16       for( ; x <= m; x += x & -x)
17           tree[x] += y;
18   }
19   int query(int x) {
20       int ans = 0;
21       for( ; x; x -= x & -x)
22           ans += tree[x];
23       return ans;
24   }
25   void cdq(int l, int r) {
26       if(l == r) return ;
27       cdq(l, Mid); cdq(Mid + 1, r);
28       int i = l, j = Mid + 1, now = l - 1;
29       while(i <= Mid && j <= r) {
30           if(a[i].y <= a[j].y) {
31               tmp[++now] = a[i];
32               add(a[i].z, a[i].cnt);
33               i++;
34           } else {
35               tmp[++now] = a[j];
36               ans[a[j].id] += query(a[j].z);
37               j++;
38           }
39       }
40       while(i <= Mid) {
41           tmp[++now] = a[i];
42           add(a[i].z, a[i].cnt);
43           i++;
44       }
45       while(j <= r) {
```

```
46          tmp[++now] = a[j];
47          ans[a[j].id] += query(a[j].z);
48          j++;
49       }
50       for(int i = l; i <= Mid; i++) add(a[i].z, -a[i].cnt);
51       for(int i = l; i <= r; i++) a[i] = tmp[i];
52  }
53  main()
54  {
55      n = read(); m = read();
56      for(int i = 1; i <= n; i++) {
57          a[i].x = read();
58          a[i].y = read();
59          a[i].z = read();
60          a[i].cnt = 1;
61      }
62      sort(a + 1, a + 1 + n, cmp);
63      for(int i = 1; i <= n; i++) {
64          if(i == 1 || !(a[i] == a[i - 1])){
65              a[++tot] = a[i];
66          }else a[tot].cnt += a[i].cnt;
67      }
68      for(int i = 1; i <= tot; i++) a[i].id = i, ttt[i] = a[i].cnt;
69      cdq(1, tot);
70      for(int i = 1; i <= tot; i++) tt[ans[i] + ttt[i] - 1] += ttt[i];
71      for(int i = 0; i < n; i++) printf("%d\n", tt[i]);
72      return 0;
73  }
```

# 数学

## 数论

### 欧拉函数

**性质**
1 和任何数互质。
$+\phi(1) = 1 + \phi(p) = p - 1(\text{p 为质数}) + \phi(x \times p) = \phi(x) \times p(p \mid x), \phi(x \times p) = \phi(x) \times p(p \nmid x)$

线性欧拉函数筛

```
1   int phi[N], f[N], pri[N], tot;
2   void getphi() {
3       int k;
4       phi[1] = 1;
5       for(int i = 2; i < N; i++) {
6           if(!f[i]) phi[pri[++tot] = i] = i - 1;
7           for(int j = 1; j <= tot && (k = i * pri[j]) < N; j++) {
8               f[k] = 1;
9               if(i % pri[j]) phi[k] = phi[i] * (pri[j] - 1);
10              else {
11                  phi[k] = phi[i] * pri[j];
12                  break;
13              }
14          }
15      }
16  }
```

$O(\sqrt{n})$ 求欧拉函数

```
1   int getphi(int x) {
2       int phi = 1;
3       for(int i = 2; i * i <= x; i++) if(x % i == 0) {
4           while(x % i == 0) {
5               if(x / i % i == 0) phi = phi * i;
6               else phi = phi * (i - 1);
7               x /= i;
8           }
9       }
10      if(x > 1) phi *= x - 1;
```

```
11        return phi;
12    }
```

**排列组合**

**斯特林近似求组合 ($\geq 15$ 时收敛)**

精度容易不够，推荐使用 python Demical 类

$$\ln n! \simeq n \ln n - n + \frac{1}{6} \ln \left(8n^3 + 4n^2 + n + \frac{1}{30}\right) + \frac{1}{2} \ln \pi$$

```
1    double lnfac(int n) {
2        return n * log(n) - n + 1.0 / 6 * log(8 * n * n * n + 4 * n * n + n + 1.0 / 30) + 0.5 * log(acos(-1.0));
3    }
4    double C(int n, int m) {
5        return exp(lnfac(n) - lnfac(n - m) - lnfac(m));
6    }
```

**Lucas 定理**

$$\binom{n}{m} = \binom{n \mod p}{m \mod p} \times \binom{n/p}{m/p}$$

```
1    int C(int n, int m) {
2        if(m > n) return 0;
3        if(n < mod) return 1ll * fac[n] * inv[n - m] % mod * inv[m] % mod;
4        else return 1ll * C(n / mod, m / mod) * C(n % mod, m % mod) % mod;
5    }
```

**Min-Max 容斥**

$$max(S) = \sum_{T \subseteq S} (-1)^{|T|-1} min(T)$$

**逆元**

线性推

```
1    inv[1] = inv[0] = 1;
2    for(int i = 2; i < N; i++)  inv[i] = (1ll * mod - mod / i) * inv[mod % i] % mod;
```

费马小定理 (模数为质数)

```
1    int inv(int x) {
2        return Pow(x % mod, mod - 2);
3    }
```

exgcd(ap 互质)

```
1    int inv(int x) {
2        int x, y;
3        exgcd(x, y, a, p);
4        return (x % p + p) % p;
5    }
```

**拓展欧几里得**

求解的是类似 $ax + by = gcd(a, b)$ 的一组解。

```
1    void exgcd(int &x, int &y, int a, int b) {
2        if(b == 0) return (void)(x = 1, y = 0);
3        exgcd(y, x, b, a % b);
4        y = y - a / b * x;
5    }
```

**拓展中国剩余定理**

拓展中国剩余定理用于解决同余方程组。

$$x \equiv a_i(mod\, b_i)$$

构造 $M_k = lcm_{i=1}^{k-1}b_i$

假设前面的解为 $p$ 显然新解 $p + M_k \times y$ 仍然是前面方程的解。

exgcd 求出 $M_k \times x + b_i \times y = gcd(M_k, b_i)$ 的解。

于是 $p' = p + x \times M_k \times (a_i - p)/gcd(M_k, b_i)$。

实际处理的时候可以直接让 $b_i = b_i/gcd(b_i, M_k)$ 防止溢出。

```cpp
#define long long ll
ll gcd(ll a, ll b) {
    return b == 0 ? a : gcd(b, a % b);
}
ll lcm(ll a, ll b) {
    return a / gcd(a, b) * b;
}
ll exgcd(ll &x, ll &y, ll a, ll b) {
    if(b == 0) return x = 1, y = 0, a;
    ll t = exgcd(y, x, b, a % b);
    y -= a / b * x;
    return t;
}
inline ll mul(ll x, ll y, ll mod){
    return (x * y - (ll)((long double)x / mod * y) * mod + mod) % mod;
}

ll excrt(ll n, ll *a, ll *b) {
    ll ans = a[1], M = b[1];
    for(ll i = 2; i <= n; i++) {
        ll c = ((a[i] - ans) % b[i] + b[i]) % b[i], x, y;
        ll t = exgcd(x, y, M, b[i]), pb = b[i] / t;
        if(c % t != 0) return -1;
        x = mul(x, c / t, pb);
        ans = ans + x * M;
        M = M *pb;
        ans = (ans % M + M) % M;
    }
    return ans;
}
```

## Miller_rabbin 素数测试

```cpp
namespace Isprime{
    ll mul(ll x, ll y, ll mod){
        return (x * y - (ll)((long double)x / mod * y) * mod + mod) % mod;
    }
    ll Pow(ll a, ll p, ll mod) {
        ll ans = 1;
        for( ; p; p >>= 1, a = mul(a, a, mod))
            if(p & 1)
                ans = mul(ans, a, mod);
        return ans % mod;
    }
    int check(ll P){
        const ll test[11] = {0, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
        if(P == 1) return false;
        if(P > 6 && P % 6 != 1 && P % 6 != 5) return false;
        ll k = 0, t = P - 1;
        while(!(t & 1)) k++, t >>= 1;
        for(int i = 1; i <= 10 && test[i] <= P; i++) {
            if(P == test[i]) return true;
            ll nxt, a = Pow(test[i], t, P);
            for(int j = 1; j <= k; j++) {
                nxt = mul(a, a, P);
                if(nxt == 1 && a != 1 && a != P - 1) return false;
                a = nxt;
```

```
25              }
26              if(a != 1)return false;
27          }
28          return true;
29      }
30  }
```

## 多项式

### 结论

1. 自然数幂之和 $s(n) = \sum_{i=0}^{n} i^k$ 是关于 $n$ 的 $k+1$ 次多项式

### 拉格朗日插值法

令拉格朗日函数

$$l_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

注意到这个函数有一些性质:
1. 次数为 $n$
2. 在 $x = x_i$ 位置值为 $1$, $x = x_j (j \neq i)$ 位置值为 $0$
于是可以凑出唯一的多项式表达式为:

$$f(x) = \sum_{i=0}^{n} y_i \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

如果要取模的话得求逆元，逆元先求好分母再一起求即可。

```
1   int interpolation(int *x, int *y, int n) {
2       int f = 0;
3       for(int i = 1; i <= n; i++) {
4           int s1 = 1, s2 = 1;
5           for(int j = 1; j <= n; j++) {
6               if(i != j) {
7                   s1 = 1ll * s1 * (k - x[j] + mod) % mod;
8                   s2 = 1ll * s2 * (x[i] - x[j] + mod) % mod;
9               }
10          }
11          f = (f + 1ll * y[i] * s1 % mod * inv(s2) % mod) % mod;
12      }
13      return f;
14  }
```

### FFT 快速傅里叶变换

FFT 的想法是把第 $k$ 号位置变成 $f(\omega_n^k)$，注意到 $\omega_n^k = -\omega_n^{k+n/2}$，于是可以进行变换。
**几条公式:**
$$\omega_n^n = 1$$
$$\omega_n^k = \omega_{2n}^{2k}$$

$$\omega_{2n}^{k+n} = -\omega_{2n}^k$$

蝴蝶变换: 相邻的位置为二进制的 reverse
DFT 变换公式 ($DFT(f)$ 为矩阵):
令

$$G(x) = a_0 + a_2 x + a_4 x^2 + ...$$
$$H(x) = a_1 + a_3 x + a_5 x^3 + ...$$

则有

$$f(x) = G(x^2) + x \times H(x^2)$$

$$DFT(f(\omega_n^k)) = DFT(G(\omega_{n/2}^k) + \omega_n^k \times DFT(H(\omega_{n/2}^k)))$$

$$DFT(f(\omega_n^{k+n/2})) = DFT(G(\omega_{n/2}^k) - \omega_n^k \times DFT(H(\omega_{n/2}^k)))$$

$DFT(G(\omega_{n/2}^k), DFT(H(\omega_{n/2}^k)))$ 可递归计算

### NTT 快速数论变换

NTT 使用原根代替复数进行运算。

原根 $g$ 的重要性质: $g^t \equiv k \mod n, t \in [0, n-2]$, $k$ 遍取 $1 \sim n-1$

原根存在的充要条件是: 模数 $n = 2, 4, p^\alpha, 2p^\alpha (p$ 为奇质数$)$。

对于一个质数 $p = qn + 1 (n = 2^m)$, 原根满足性质 $g^{qn} \equiv 1 \mod p$。

它满足和复数近似的性质, 我们把 $q$ 看成复数中的 $2\pi$, 就可以套用 FFT 实现 NTT 了。

$g_n^n \equiv 1, g_n^n \equiv -1$

通常取

$$p = 1004535809 = 7 \times 479 \times 2^{21} + 1, g = 3$$

$$p = 998244353 = 7 \times 17 \times 2^{23} + 1, g = 3$$

```cpp
const int P = 998244353, G = 3, Gi = 332748118;
struct Complex {double x, y;};
Complex operator+(const Complex &a, const Complex &b) {return (Complex) {a.x + b.x, a.y + b.y};}
Complex operator-(const Complex &a, const Complex &b) {return (Complex) {a.x - b.x, a.y - b.y};}
Complex operator*(const Complex &a, const Complex &b) {return (Complex) {a.x * b.x - a.y * b.y, a.x * b.y + a.y *
    b.x};}
namespace Polynomial {
    const double Pi = acos(-1.0);
    int rev[N];
    template <typename T>
    void change(T *y, int n) {
        for(int i = 0; i < n; i++)
            rev[i] = (rev[i >> 1] >> 1) | ((i & 1) ? (n >> 1) : 0);
        for(int i = 0; i < n; i++)
            if(i < rev[i])
                swap(y[i], y[rev[i]]);
    }
    void FFT(Complex *A, int n, int type) {
        //type = 1 DFT
        //type = -1 IDFT
        //确保 n 是 2 的幂次
        change(A, n);
        for(int m = 1; m < n; m <<= 1) {
            Complex Wn = (Complex) {cos(Pi / m), type * sin(Pi / m)};
            for(int i = 0; i < n; i += 2 * m) {
                Complex w = (Complex) {1.0, 0};
                for(int j = 0; j < m; j++, w = w * Wn) {
                    Complex x = A[i + j], y = w * A[i + j + m];
                    A[i + j] = x + y;
                    A[i + j + m] = x - y;
                }
            }
        }
        if(type == -1) {
            for(int i = 0; i < n; i++)
                A[i].x = A[i].x / n;
        }
    }
    void NTT(int *A, int n, int type) {
        //type = 1 DFT
        //type = -1 IDFT
        change(A, n);
```

```
42          for(int m = 1; m < n; m <<= 1) {
43              int Wn = Pow(type == 1 ? G : Gi, (P - 1) / (m << 1));
44              for(int i = 0; i < n; i += 2 * m) {
45                  int w = 1;
46                  for(int j = 0; j < m; j++, w = 1ll * w * Wn % P) {
47                      int x = A[i + j], y = 1ll * w * A[i + j + m] % P;
48                      A[i + j] = (x + y) % P;
49                      A[i + j + m] = (x - y + P) % P;
50                  }
51              }
52          }
53          if(type == -1) {
54              int inv = Pow(n, P - 2);
55              for(int i = 0; i < n; i++)
56                  A[i] = 1ll * A[i] * inv % P;
57          }
58      }
59
60  }
61  //以下代码加在主函数内
62  limit = 1;
63  while(limit <= n + m) limit <<= 1;
64  Polynomial :: FFT(A, limit, 1);
65  Polynomial :: FFT(B, limit, 1);
66  for(int i = 0; i < limit; i++) A[i] = A[i] * B[i];
67  Polynomial :: FFT(A, limit, -1);
```

**FWT 快速沃尔什变换**

FWT 用于计算下列多项式

$$C[k] = \sum_{i \oplus j = k} A[i] \times B[j]$$

先通过 FWT 将 $A, B$ 变为 $FWT(A), FWT(B)$，这样有 $FWT(C) = FWT(A) \times FWT(B)$。
当然位运算符不同的时候对应的变换形式也需要改变。

$a \in S, b \in S$ 可以表示为 $a|b \in S$
FWT 为线性变换 $\sum FWT(F) = FWT(\sum F)$

**与卷积**
当 $\oplus = and$ 的时候
$FWT(A) = (FWT(A_0) + FWT(A_1), FWT(A_1))$
$FWT(A) = A$(长度为 1)
$IFWT(A) = (IFWT(A_0) - IFWT(A_1), IFWT(A_1))$

**或卷积**
当 $\oplus = or$ 的时候
$FWT(A) = (FWT(A_0), FWT(A_0) + FWT(A_1))$
$FWT(A) = A$(长度为 1)
$IFWT(A) = (IFWT(A_0), IFWT(A_1) - IFWT(A_0))$

**异或卷积**
当 $\oplus = xor$ 的时候
$FWT(A) = (FWT(A_0) + FWT(A_1), FWT(A_0) - FWT(A_1))$
$FWT(A) = A$(长度为 1)
$IFWT(A) = (\frac{IFWT(A_0) + IFWT(A_1)}{2}, \frac{IFWT(A_0) - IFWT(A_1)}{2})$

```
1  namespace Polynomial {
2      void FWT_or(int *A, int n, int type) {
3          for(int m = 1; m < n; m <<= 1) {
4              for(int i = 0; i < n; i += 2 * m) {
5                  for(int j = 0; j < m; j++) {
6                      A[i + j + m] = (1ll * A[i + j + m] + A[i + j] * type + mod) % mod;
7                  }
8              }
```

```
9          }
10      }
11      void FWT_and(int *A, int n, int type) {
12          for(int m = 1; m < n; m <<= 1) {
13              for(int i = 0; i < n; i += 2 * m) {
14                  for(int j = 0; j < m; j++) {
15                      A[i + j] = (1ll * A[i + j + m] * type + A[i + j] + mod) % mod;
16                  }
17              }
18          }
19      }
20      void FWT_xor(int *A, int n, int type) {
21          int inv_2 = Pow(2, mod - 2);
22          for(int m = 1; m < n; m <<= 1) {
23              for(int i = 0; i < n; i += 2 * m) {
24                  for(int j = 0; j < m; j++) {
25                      int x = A[i + j], y = A[i + j + m];
26                      A[i + j] = (1ll * x + y) * (type == 1 ? 1 : inv_2) % mod;
27                      A[i + j + m] = (1ll * x - y + mod) * (type == 1 ? 1 : inv_2) % mod;
28                  }
29              }
30          }
31      }
32  }
```

**子集卷积**

子集卷积求的是下面一个式子:

$$c_k = \sum_{i|j=k,\,i\&j=0} a_i \times b_j$$

就是把集合 k 划分成两个集合。

后面那个与的条件通过 $|k| = |i| + |j|$ 干掉，加一维集合元素个数，就变成了

$$c[i+j][mask_k] = \sum_{i|j=k} a[i][mask_i] \times b[j][mask_j]$$

这个可以用 FWT 算。

```
1   namespace ssc{
2       int f[21][1 << 21], g[21][1 << 21], ans[21][1 << 21];
3       void subset_convolution(int *A, int *B, int *C, int n, int lim) {
4   //      memset(f, 0, sizeof(f));
5   //      memset(g, 0, sizeof(g));
6           for(int i = 0; i < lim; i++) f[__builtin_popcount(i)][i] = A[i];
7           for(int i = 0; i < lim; i++) g[__builtin_popcount(i)][i] = B[i];
8           for(int i = 0; i <= n; i++) FWT_or(f[i], lim, 1), FWT_or(g[i], lim, 1);
9           for(int i = 0; i <= n; i++)
10              for(int j = 0; j <= i; j++)
11                  for(int k = 0; k < lim; k++)
12                      ans[i][k] = (ans[i][k] + 1ll * f[j][k] * g[i - j][k] % mod) % mod;
13          for(int i = 0; i <= n; i++) FWT_or(ans[i], lim, -1);
14          for(int i = 0; i < lim; i++) C[i] = ans[__builtin_popcount(i)][i];
15      }
16  }
```

# 群论

## 结论

1. **子群检验法**: 群 $G$ 是群 $H$ 的子群的充分必要条件: 对于所有元素 $h, g$, 只需检查 $g^{-1} \cdot h \in H$。

# 线性代数

## 矩阵运算全家桶

```cpp
struct mat {
    int g[5][5], n, m;
};
void operator+=(mat &a, const mat &b) {
    if(a.n != b.n || a.m != b.m) cerr << "+= size error" << endl, exit(0);
    for(int i = 1; i <= a.n; i++)
        for(int j = 1; j <= a.m; j++) {
            a.g[i][j] = (a.g[i][j] + b.g[i][j]);
            if(a.g[i][j] >= mod) a.g[i][j] -= mod;
        }
}
void operator-=(mat &a, const mat &b) {
    if(a.n != b.n || a.m != b.m) cerr << "-= size error" << endl, exit(0);
    for(int i = 1; i <= a.n; i++)
        for(int j = 1; j <= a.m; j++) {
            a.g[i][j] -= b.g[i][j];
            if(a.g[i][j] < 0) a.g[i][j] += mod;
        }
}
mat operator+(const mat &a, const mat &b) {
    if(a.n != b.n || a.m != b.m) cerr << "+ size error" << endl, exit(0);
    mat c;
    c.n = a.n; c.m = a.m;
    for(int i = 1; i <= a.n; i++)
        for(int j = 1; j <= a.m; j++) {
            c.g[i][j] = (a.g[i][j] + b.g[i][j]);
            if(c.g[i][j] >= mod) c.g[i][j] -= mod;
        }
    return c;
}
mat operator-(const mat &a, const mat &b) {
    if(a.n != b.n || a.m != b.m) cerr << "- size error" << endl, exit(0);
    mat c;
    c.n = a.n; c.m = a.m;
    for(int i = 1; i <= a.n; i++)
        for(int j = 1; j <= a.m; j++) {
            c.g[i][j] = (a.g[i][j] - b.g[i][j]);
            if(c.g[i][j] < 0) c.g[i][j] += mod;
        }
    return c;
}
mat operator*(const mat &a, const mat &b) {
    if(a.m != b.n) cerr << "* size error" << endl, exit(0);
    mat c;
    c.n = a.n; c.m = b.m;
    for(int i = 1; i <= a.n; i++) {
        for(int j = 1; j <= b.m; j++) {
            c.g[i][j] = 0;
            for(int k = 1; k <= a.m; k++) {
                c.g[i][j] = c.g[i][j] + 1ll * a.g[i][k] * b.g[k][j] % mod;
                if(c.g[i][j] >= mod) c.g[i][j] -= mod;
            }
        }
    }
    return c;
}
mat Pow(mat a, int p) {
    if(a.n != a.m) cerr << "* size error" << endl, exit(0);
    mat ans;
    ans.n = ans.m = a.n;
    memset(ans.g, 0, sizeof(ans.g));
    for(int i = 1; i <= ans.n; i++) ans.g[i][i] = 1;
    for(; p; p >>= 1, a = a * a)
        if(p & 1)
            ans = ans * a;
    return ans;
}
```

**高斯消元**

```cpp
namespace Gauss {
    int n, m;
    double g[N][N];
    int iszero(double x) {return fabs(x) < eps;}
    void exchange(int i, int j) {
        for(int k = 1; k <= m; k++)
            swap(g[i][k], g[j][k]);
    }
    void minus(int i, int j, double t) {
        for(int k = 1; k <= m; k++)
            g[j][k] -= g[i][k] * t;
    }
    void div(int i, double d) {
        for(int k = 1; k <= m; k++)
            g[i][k] /= d;
    }
    void solve() {
        for(int i = 1; i <= n; i++) {
            if(iszero(g[i][i])) {
                for(int j = i + 1; j <= n; j++) {
                    if(!iszero(g[j][i])) {
                        exchange(i, j);
                        break;
                    }
                }
                if(iszero(g[i][i])) continue;
            }
            div(i, g[i][i]);
            for(int j = 1; j <= n; j++) if(i != j && !iszero(g[j][i])){
                minus(i, j, g[j][i]);
            }
        }
    }
}
```

# 图论

## 树论

### 树的直径

模板：POJ - 1985

- 两遍 DFS

```cpp
void dfs(int x, int fa) {
    for(int i = 0; i < E[x].size(); i++) {
        int y = E[x][i].ver;
        int w = E[x][i].val;
        if(y == fa) continue;
        d[y] = d[x] + w;
        if(d[y] > d[c]) c = y;
        dfs(y, x);
    }
}
signed main()
{
    n = read();
    for(int i = 1; i < n; i++) {
        int x = read(), y = read(), w = read();
        E[x].push_back((Edge) {y, w});
        E[y].push_back((Edge) {x, w});
    }
    dfs(1, 0);
    d[c] = 0;
    dfs(c, 0);
    printf("%d\n", d[c]);
```

```
23        return 0;
24    }
```

● 树形 DP

```
1   void dfs(int x, int fa) {
2       d1[x] = d2[x] = 0;
3       for(int i = 0; i < E[x].size(); i++) {
4           int y = E[x][i].ver;
5           int w = E[x][i].val;
6           if(y == fa) continue;
7           dfs(y, x);
8           int t = d1[y] + w;
9           if(t > d1[x]) {
10              d2[x] = d1[x];
11              d1[x] = t;
12          } else if(t > d2[x]) {
13              d2[x] = t;
14          }
15      }
16      d = max(d, d1[x] + d2[x]);
17  }
18  signed main()
19  {
20      n = read();
21      for(int i = 1; i < n; i++) {
22          int x = read(), y = read(), w = read();
23          E[x].push_back((Edge) {y, w});
24          E[y].push_back((Edge) {x, w});
25      }
26      dfs(1, 0);
27      printf("%d\n", d);
28      return 0;
29  }
```

### 求 LCA

● 树链剖分

```
1   namespace Tree {
2       int siz[N], mson[N], ltp[N], fa[N], dth[N];
3       vector<int> son[N];
4       void dfs1(int x, int pre) {
5           siz[x] = 1;
6           mson[x] = 0;
7           fa[x] = pre;
8           dth[x] = dth[pre] + 1;
9           for(auto y : son[x]) if(y != pre) {
10              dfs1(y, x);
11              if(mson[x] == 0 || siz[y] > siz[mson[x]]) mson[x] = y;
12          }
13      }
14      void dfs2(int x, int pre, int tp) {
15          ltp[x] = tp;
16          if(mson[x]) dfs2(mson[x], x, tp);
17          for(auto y : son[x]) if(y != pre && y != mson[x]) {
18              dfs2(y, x, y);
19          }
20      }
21      void init() {
22          dfs1(1, 0);
23          dfs2(1, 0, 1);
24      }
25      int LCA(int x, int y) {
26          while(ltp[x] != ltp[y]) {
27              if(dth[ltp[x]] > dth[ltp[y]]) x = fa[ltp[x]];
28              else y = fa[ltp[y]];
29          }
30          return dth[y] > dth[x] ? x : y;
```

```
31            }
32      }
```

- 倍增

```
1   namespace Tree {
2       vector<int> son[N];
3       int root, fa[N][31], dth[N];
4       void dfs(int x, int pre) {
5           fa[x][0] = pre;
6           dth[x] = dth[pre] + 1;
7           for(int i = 1; i <= 30; i++)
8               fa[x][i] = fa[fa[x][i - 1]][i - 1];
9           for(auto y : son[x]) if(y != pre)
10              dfs(y, x);
11      }
12      void init() {
13          dfs(root, 0);
14      }
15      int LCA(int x, int y) {
16          if(dth[x] > dth[y]) swap(x, y);
17          for(int i = 30; ~i; i--)
18              if(dth[fa[y][i]] >= dth[x])
19                  y = fa[y][i];
20          if(x == y) return x;
21          for(int i = 30; ~i; i--)
22              if(fa[y][i] != fa[x][i]) {
23                  x = fa[x][i];
24                  y = fa[y][i];
25              }
26          return fa[x][0];
27      }
28  }
```

## 树上启发式合并

长春站的痛.jpg

- 先递归计算轻儿子的答案
- 计算重儿子的答案，并且保留重儿子的状态数组
- 把其他所有轻儿子的答案加到状态数组中，更新当前点的答案

```
1   void dfs1(int x, int pre) {
2       siz[x] = 1;
3       mson[x] = 0;
4       for(auto y : son[x]) if(y != pre) {
5           dfs1(y, x);
6           siz[x] += siz[y];
7           if(!mson[x] || siz[y] > siz[mson[x]]) mson[x] = y;
8       }
9   }
10  void add(int x, int pre, int v) {
11      cnt[col[x]] += v;
12      if(cnt[col[x]] > Mx) Mx = cnt[col[x]], sum = col[x];
13      else if(cnt[col[x]] == Mx) sum += col[x];
14      for(auto y : son[x]) {
15          if(y == pre || y == Son) continue;
16          add(y, x, v);
17      }
18  }
19  void dfs2(int x, int pre, int keep) {
20      for(auto y : son[x]) {
21          if(y == pre || y == mson[x]) continue;
22          dfs2(y, x, 0);
23      }
24      if(mson[x]) dfs2(mson[x], x, 1), Son = mson[x];
25      add(x, pre, 1); Son = 0;
26      ans[x] = sum;
27      if(!keep) add(x, pre, -1), sum = 0, Mx = 0;
28
29  }
```

# 图论

## 第 k 短路

模板：HDU-6351

估值函数：$h(x) = f(x) + g(x)$，其中 $f(x)$ 为从起点到现在的距离，$g(x)$ 为起点到当前点的最短路。

```cpp
bool operator<(const node &a, const node &b) {
    return a.f + a.g > b.f + b.g;
}
priority_queue<node> q;
signed main()
{
    n = read(); m = read();
    for(int i = 1; i <= m; i++) {
        int x, y, w;
        x = read(); y = read(); w = read();
        E[x].push_back((Edge) {y, w});
        re[y].push_back((Edge) {x, w});
    }
    s = read(); t = read(); k = read();
    memset(dis, 0x3f, sizeof(dis)); dis[t] = 0;
    q.push((node) {t, 0, 0});
    while(q.size()) {
        int x = q.top().x, d = q.top().f;
        q.pop();
        if(dis[x] < d) continue;
        for(int i = 0; i < re[x].size(); i++) {
            int y = re[x][i].y, w = re[x][i].w;
            if(dis[y] > dis[x] + w) {
                dis[y] = dis[x] + w;
                q.push((node) {y, dis[y], 0});
            }
        }
    }
    for(int i = 1; i <= n; i++) cnt[i] = k;
    cnt[s]++;
    q.push((node) {s, 0, dis[s]});
    while(q.size()) {
        int x = q.top().x, f = q.top().f, g = q.top().g;
        q.pop();
        if(cnt[x] == 0) continue;
        cnt[x]--;
        if(x == t && cnt[x] == 0) {
            printf("%lld\n", f);
            return 0;
        }
        for(int i = 0; i < E[x].size(); i++) {
            int y = E[x][i].y, w = E[x][i].w;
            q.push((node) {y, f + w, dis[y]});
        }
    }
    printf("-1\n");
    return 0;
}
```

## 二分图匹配

### 结论

**最大匹配数**：最大匹配的匹配边的数目

**最小点/边覆盖数**：选取最少的点/边，使任意一条边至少有一个点被选择 / 点至少连有一条边。

**最大独立数**：选取最多的点，使任意所选两点均不相连

**最小路径覆盖数**：对于一个 DAG（有向无环图），选取最少条路径，使得每个顶点属于且仅属于一条路径。路径长可以为 0（即单个点）。

1. 最大匹配数 = 最小点覆盖数（这是 Konig 定理）
2. 最大匹配数 = 最大独立数

3. 最小路径覆盖数 = 顶点数 - 最大匹配数
4. 原图的最大团 = 补图的最大独立集 原图的最大独立集 = 补图的最大团
5. 最小边覆盖 = 顶点数 - 最大匹配数

在一般图中：

**最小不相交路径覆盖**：每个点拆点为 $2x - 1, 2x$，那么一条边 $(x, y)$，则连边 $(2x - 1, 2y)$，答案是 $n - maxmatch$

**最小可相交路径覆盖**：跑一遍传递闭包，按传递闭包上的边建边之后转化为最小不相交路径覆盖。

**二分图最大匹配的必须边**：

在完备匹配中：

匹配边从左到右方向，非匹配边从右到左方向，则一条边为必须边当且仅当边在最大匹配中，并且边所连的两个点**不在**同一个强连通分量中。

在非完备匹配中：

### 匈牙利算法

```
int dfs(int x) {
    for(int i = head[x]; i; i = nxt[i]) {
        int y = ver[i];
        if(vis[y]) continue;
        vis[y] = 1;
        if(!match[y] || dfs(match[y])) {
            match[y] = x;
            return true;
        }
    }
    return false;
}
for(int i = 1; i <= n; i++) {
    memset(vis, 0, sizeof(vis));
    if(dfs(i)) ans++;
}
```

### KM 算法二分图最大权匹配

KM 算法只支持二分图最大权完美匹配，若图不一定存在完美匹配，注意补 0 边和补点。

KM 算法引入了顶标的概念，用 $la[x]$ 和 $lb[x]$ 分别保存两侧点的顶标，顶标必须满足大于所有边。
每次对每个点进行循环匹配，匹配中统计一个 $delta$ 表示最小的权值使得一条边可以加入。
然后修改顶标再继续匹配。

```
int la[N], lb[N], va[N], vb[N], delta, match[N], g[N][N], n;
int dfs(int x) {
    va[x] = 1;
    for(int y = 1; y <= n; y++) {
        if(!vb[y]) {
            if(la[x] + lb[y] - g[x][y] == 0) {
                vb[y] = 1;
                if(!match[y] || dfs(match[y])) {
                    match[y] = x;
                    return true;
                }
            } else delta = min(delta, la[x] + lb[y] - g[x][y]);
        }
    }
    return false;
}
void work() {
    for(int i = 1; i <= n; i++)
        for(int j = 1; j <= n; j++)
            g[i][j] = read();
    memset(match, 0, sizeof(match));
    for(int i = 1; i <= n; i++) {
        la[i] = g[i][1];
        lb[i] = 0;
```

```
25          for(int j = 2; j <= n; j++)
26              la[i] = max(la[i], g[i][j]);
27      }
28      for(int i = 1; i <= n; i++) {
29          while(true) {
30              memset(va, 0, sizeof(va));
31              memset(vb, 0, sizeof(vb));
32              delta = 0x3f3f3f3f;
33              if(dfs(i)) break;
34              for(int j = 1; j <= n; j++) {
35                  if(va[j]) la[j] -= delta;
36                  if(vb[j]) lb[j] += delta;
37              }
38          }
39      }
40      long long ans = 0;
41      for(int i = 1; i <= n; i++)
42          ans += g[match[i]][i];
43      printf("%lld\n", ans);
44  }
```

## 网络流

### Dinic 算法

```
1   const int inf = 0x3f3f3f3f;
2   queue<int> q;
3   int d[N];
4   int bfs() {
5       memset(d, 0, sizeof(int) * (t + 10)); d[s] = 1;
6       while(q.size()) q.pop(); q.push(s);
7       while(q.size()) {
8           int x = q.front(); q.pop();
9           for(int i = head[x]; i; i = nxt[i]) {
10              if(d[ver[i]]) continue;
11              if(edge[i] <= 0) continue;
12              d[ver[i]] = d[x] + 1;
13              q.push(ver[i]);
14          }
15      }
16      return d[t];
17  }
18  int dinic(int x, int flow) {
19      if(x == t) return flow;
20      int k, res = flow;
21      for(int i = head[x]; i && res; i = nxt[i]) {
22          if(d[ver[i]] != d[x] + 1 || edge[i] <= 0) continue;
23          k = dinic(ver[i], min(res, edge[i]));
24          if(k == 0) d[ver[i]] = 0;
25          edge[i] -= k;
26          edge[i ^ 1] += k;
27          res -= k;
28      }
29      return flow - res;
30  }
```

### EK 算法费用流

```
1   //反向边 cost 为负数, 容量为 0
2   int SPFA() {
3       queue<int> q; q.push(s);
4       memset(dis, 0x3f, sizeof(dis)); dis[s] = 0;
5       memset(vis, 0, sizeof(vis)); vis[s] = 1;
6       q.push(s); flow[s] = 0x3f3f3f3f;
7       while(q.size()) {
8           int x = q.front();
9           vis[x] = 0; q.pop();
10          for(int i = head[x]; i; i = nxt[i]) {
11              if(edge[i] <= 0) continue;
12              if(dis[ver[i]] > dis[x] + cost[i]) {
```

```
13              dis[ver[i]] = dis[x] + cost[i];
14              pre[ver[i]] = i;
15              flow[ver[i]] = min(flow[x], edge[i]);
16              if(!vis[ver[i]]) {
17                  q.push(ver[i]);
18                  vis[ver[i]] = 1;
19              }
20          }
21      }
22  }
23      return dis[t] != 0x3f3f3f3f;
24  }
25  void update() {
26      int x = t;
27      while(x != s) {
28          int i = pre[x];
29          edge[i] -= flow[t];
30          edge[i ^ 1] += flow[t];
31          x = ver[i ^ 1];
32      }
33      maxflow += flow[t];
34      minncost += dis[t] * flow[t];
35  }
```

### Dinic 算法费用流

```
1   int SPFA() {
2       while(q.size()) q.pop(); q.push(s);
3       memset(d, 0x3f, sizeof(int) * (n + 10)); d[s] = 0;
4       memset(vis, 0, sizeof(int) * (n + 10)); vis[s] = 1;
5       while(q.size()) {
6           int x = q.front(); q.pop();
7           vis[x] = 0;
8           for(int i = head[x]; i; i = nxt[i]) {
9               if(edge[i] <= 0) continue;
10              if(d[ver[i]] > d[x] + cost[i]) {
11                  d[ver[i]] = d[x] + cost[i];
12                  if(!vis[ver[i]]) {
13                      vis[ver[i]] = 1;
14                      q.push(ver[i]);
15                  }
16              }
17          }
18      }
19      return d[t] != 0x3f3f3f3f;
20  }
21  int dinic(int x, int flow) {
22      if(x == t) return flow;
23      vis[x] = 1;
24      int k, res = flow;
25      for(int i = head[x]; i && res; i = nxt[i]) {
26          if(vis[ver[i]]) continue;
27          if(d[ver[i]] != d[x] + cost[i] || edge[i] <= 0) continue;
28          k = dinic(ver[i], min(edge[i], res));
29          if(!k) d[ver[i]] = -1;
30          edge[i] -= k;
31          edge[i ^ 1] += k;
32          res -= k;
33          mincost += cost[i] * k;
34      }
35      vis[x] = 0;
36      return flow - res;
37  }
```

### 无源汇上下界可行流

$x->y$, 则 $s$ 向 $y$, $s$ 向 $x$ 连 $l$, $x$ 向 $y$ 连 $r-l$, 有可行流的条件是 $s$ 出边全满流, 解通过残量网络构造出。

```
1   for(int i = 1; i <= m; i++) {
2       int x = read(), y = read();
3       int l = read(), r = read();
```

```
4        low[i] = l;
5        add(x, y, r - l); add(y, x, 0);
6        id[i] = tot;
7        add(s, y, l); add(y, s, 0);
8        add(x, t, l); add(t, x, 0);
9    }
10   while(bfs())
11       dinic(s, inf);
12   int f = 1;
13   for(int i = head[s]; i; i = nxt[i]) {
14       f &= (edge[i] == 0);
15   }
16   printf("%s\n", f ? "YES" : "NO");
17   if(!f) return 0;
18   for(int i = 1; i <= m; i++) {
19       printf("%d\n", edge[id[i]] + low[i]);
20   }
```

## 连通性算法

### Tarjan 强连通分量

$dfn[x]$: $dfs$ 序。

$low[x]$: 追溯值, 指 $x$ 的子树内部, 通过一条非树边能到达的最小的 $dfn$ 值。

如果 $dfn[x] == low[x]$, 当前栈中, $x$ 以后的元素为一个强连通。

```
1    void tarjan(int x) {
2        low[x] = dfn[x] = ++dfncnt;
3        s[++t] = x; vis[x] = 1;
4        for(int i = head[x]; i; i = nxt[i]) {
5            if(!dfn[ver[i]]) {
6                tarjan(ver[i]);
7                low[x] = min(low[x], low[ver[i]]);
8            } else if(vis[ver[i]]) {
9                low[x] = min(low[x], dfn[ver[i]]);
10           }
11       }
12       if(dfn[x] == low[x]) {
13           int z = -1;
14           ++sc;
15           while(z != x) {
16               scc[s[t]] = sc;
17               siz[sc]++;
18               vis[s[t]] = 0;
19               z = s[t];
20               t--;
21           }
22       }
23   }
24   //从任意点开始跑, 但是注意如果图不连通, 需要每个点跑一次
25   for(int i = 1; i <= n; i++)
26       if(!dfn[i])
27           tarjan(i);
```

### 点双连通

### Tarjan 割点判定

```
1    int cut[N];
2    namespace v_dcc {
3        int root, low[N], dfn[N], dfntot;
4        void tarjan(int x) {
5            low[x] = dfn[x] = ++dfntot;
6            int flag = 0;
7            for(int i = head[x]; i; i = nxt[i]) {
8                int y = ver[i];
9                if(!dfn[y]) {
10                   tarjan(y);
11                   low[x] = min(low[x], low[y]);
```

```
12              if(low[y] >= dfn[x]) {
13                  flag++;
14                  if(x != root || flag > 1) cut[x] = 1;
15
16              }
17          } else low[x] = min(low[x], dfn[y]);
18      }
19  }
20  void getcut() {
21      for(int i = 1; i <= n; i++)
22          if(!dfn[i])
23              tarjan(root = i);
24  }
25 }
```

### 求点双连通分量

点双连通分量比较复杂，一个点可能存在于多个点双连通分量当中，一个点删除与搜索树中的儿子节点断开时，不能在栈中弹掉父亲点，但是父亲点属于儿子的 v-dcc。

```
1  int cut[N];
2  vector<int> dcc[N];
3  namespace v_dcc {
4      int s[N], t, root;
5      int es[N], et;
6      void tarjan(int x) {
7          dfn[x] = low[x] = ++dfntot;
8          s[++t] = x;
9          if(x == root && head[x] == 0) {
10             dcc[++dc].clear();
11             dcc[dc].push_back(x);
12             return ;
13         }
14         int flag = 0;
15         for(int i = head[x]; i; i = nxt[i]) {
16             int y = ver[i];
17             if(!dfn[y]) {
18                 tarjan(y);
19                 low[x] = min(low[x], low[y]);
20                 if(low[y] >= dfn[x]) {
21                     flag++;
22                     if(x != root || flag > 1) cut[x] = true;
23                     dcc[++dc].clear();
24                     int z = -1;
25                     while(z != y) {
26                         z = s[t--];
27                         dcc[dc].push_back(z);
28                     }
29                     dcc[dc].push_back(x);
30                 }
31             } else low[x] = min(low[x], dfn[y]);
32         }
33     }
34     void get_cut() {
35         for(int i = 1; i <= n; i++)
36             if(!dfn[i])
37                 tarjan(root = i);
38     }
39 }
```

### 边双连通

搜索树上的点 x，若它的一个儿子 y，满足严格大于号 $low[y] > dfn[x]$，那么这条边就是桥。

注意由于会有重边，不能仅仅考虑他的父亲编号，而应该记录入边编号。

```
1  namespace e_dcc {
2      int low[N], dfn[N], dfntot;
3      vector<int> E[N];
4      void tarjan(int x, int in_edge) {
5          low[x] = dfn[x] = ++dfntot;
```

```
6          for(int i = head[x]; i; i = nxt[i]) {
7              int y = ver[i];
8              if(!dfn[y]) {
9                  tarjan(y, i);
10                 low[x] = min(low[x], low[y]);
11                 if(low[y] > dfn[x])
12                     bridge[i] = bridge[i ^ 1] = true;
13             } else if(i != (in_edge ^ 1))
14             //注意运算优先级
15                 low[x] = min(low[x], dfn[y]);
16         }
17     }
18     void getbridge() {
19         for(int i = 1; i <= n; i++)
20             if(!dfn[i])
21                 tarjan(i, 0);
22     }
23     void dfs(int x) {
24         dcc[x] = dc;
25         for(int i = head[x]; i; i = nxt[i]) {
26             if(!dcc[ver[i]] && !bridge[i]) {
27                 dfs(ver[i]);
28             }
29         }
30     }
31     void getdcc() {
32         for(int i = 1; i <= n; i++) {
33             if(!dcc[i]) {
34                 ++dc;
35                 dfs(i);
36             }
37         }
38     }
39     void getgraphic() {
40         for(int x = 1; x <= n; x++) {
41             for(int i = head[x]; i; i = nxt[i]) {
42                 if(dcc[ver[i]] != dcc[x]) {
43                     E[dcc[x]].push_back(dcc[ver[i]]);
44                     E[dcc[ver[i]]].push_back(dcc[x]);
45                 }
46             }
47         }
48     }
49 }
```

## 2-SAT

2-SAT 用于解决每个变量的 01 取值问题，用于判断是否存在一种不冲突取值方法。

建边方法：假如选了 $A$ 之后，$B$ 的取值**确定**，那么就 $A$ 的这个取值向 $B$ 的这个取值建边，否则不要建边。

判定方法：如果，$\exists A$，使得 $A$ 和 $\neg A$ 在同一个强连通分量里面，说明不存在一种合法取值，否则存在。

输出方案：自底向上确定每个变量的取值，由于 tarjan 求解强连通分量是自底向上，所以编号比较小的强连通是位于 DAG 底部的。

基于 tarjan 的方案输出就变得十分简单了，只要判断一个点和对立节点哪个 scc 的编号小就行了。

例如：$A->B->C$，那么 C 的编号最小。

```
1  for(int i = 1; i <= m; i++) {
2      int x = read() + 1, y = read() + 1;
3      int w = read();
4      char c[10];
5      scanf("%s", c + 1);
6      if(c[1] == 'A') {
7          if(w) {
8              add(2 * x - 0, 2 * x - 1);
9              add(2 * y - 0, 2 * y - 1);
10         } else {
11             add(2 * x - 1, 2 * y - 0);
12             add(2 * y - 1, 2 * x - 0);
```

```
13              }
14          }
15          if(c[1] == 'O') {
16              if(w) {
17                  add(2 * x - 0, 2 * y - 1);
18                  add(2 * y - 0, 2 * x - 1);
19              } else {
20                  add(2 * x - 1, 2 * x - 0);
21                  add(2 * y - 1, 2 * y - 0);
22              }
23          }
24          if(c[1] == 'X') {
25              if(w) {
26                  add(2 * x - 0, 2 * y - 1);
27                  add(2 * x - 1, 2 * y - 0);
28                  add(2 * y - 0, 2 * x - 1);
29                  add(2 * y - 1, 2 * x - 0);
30              } else {
31                  add(2 * x - 0, 2 * y - 0);
32                  add(2 * x - 1, 2 * y - 1);
33                  add(2 * y - 0, 2 * x - 0);
34                  add(2 * y - 1, 2 * x - 1);
35              }
36          }
37      }
38      for(int i = 1; i <= 2 * n; i++)
39          if(!dfn[i])
40              tarjan(i);
41      for(int i = 1; i <= n; i++) {
42          if(scc[2 * i - 0] == scc[2 * i - 1]) {
43              printf("NO\n");
44              return 0;
45          }
46      }
47      printf("YES\n");
48      //2 * x - a -> 2 * y - b 的边表示, 假如 x 取值为 a, 那么 y 的取值必须为 b

49
50      //输出方案
51      for(int i = 2; i <= 2 * n; i += 2) {
52          if(scc[i - 0] == scc[i - 1]) {
53              printf("NO\n");
54              return 0;
55          } else ans[(i + 1) / 2] = scc[i - 1] < scc[i - 0];
56      }
```

# 计算几何

## 公式

三角形内心和重心公式（点为 A,B,C, 对边为 a,b,c）:
+ 内心: $\frac{aA+bB+cC}{a+b+c}$
+ 重心: $\frac{A+B+C}{3}$
+ 外心，垂心: 用两直线交点计算

### 结构体定义

```
1  const double Pi = acos(-1.0);
2  const double eps = 1e-11;
3  // 三态函数
4  int sgn(double x) {
5      if(fabs(x) < eps) return 0;
6      else return x < 0 ? -1 : 1;
7  }
8  struct line;
9  struct Point;
10 struct Point {
11     double x, y;
12     Point() : x(0), y(0) {}
```

```
13        Point(double x, double y) : x(x), y(y) {}
14        Point(const line &l);
15    };
16    struct line{
17        Point s, t;
18        line() {}
19        line(const Point &s, const Point &t) : s(s), t(t) {}
20    };
21
22    struct circle{
23        Point c;
24        double r;
25        circle() : c(Point(0,0)), r(0) {}
26        circle(const Point &c, double r) : c(c), r(r) {}
27        Point point(double a) {
28            return Point(c.x + cos(a)*r, c.y + sin(a)*r);
29        }
30    };
31    typedef Point Vector;
32    Point operator+(const Point &a, const Point &b) { return Point(a.x + b.x, a.y + b.y); }
33    Point operator-(const Point &a, const Point &b) { return Point(a.x - b.x, a.y - b.y); }
34    Point operator*(const Point &a, const double &c) { return Point(c * a.x, c * a.y); }
35    Point operator/(const Point &a, const double &c) { return Point(a.x / c, a.y / c); }
36    inline bool operator < (const Point &a, const Point &b) {
37        return sgn(a.x - b.x) < 0 || (sgn(a.x - b.x) == 0 && sgn(a.y - b.y) < 0);
38    }
39    Point :: Point(const line &l) { *this = l.t - l.s; }
40    bool operator == (const Point& a, const Point& b) { return !sgn(a.x - b.x) && !sgn(a.y - b.y); }
41    // 点积
42    double dot(const Vector& a, const Vector& b) { return a.x * b.x + a.y * b.y; }
43    // 叉积
44    double det(const Vector& a, const Vector& b) { return a.x * b.y - a.y * b.x; }
45    double cross(const Point& s, const Point& t, const Point& o = Point()) { return det(s - o, t - o); }
```

## 基本操作

```
1    // 点到原点距离
2    double abs(const Point &a){ return sqrt(a.x * a.x + a.y * a.y); }
3    // 点旋转 theta 角度
4    Point rot(const Point &a, double theta){ return Point(a.x * cos(theta) - a.y * sin(theta), a.x * sin(theta) + a.y *
     ↪ cos(theta)); }
5    // 逆时针旋转 90 度
6    Point rotCCW90(const Point &a) { return Point(-a.y, a.x); }
7    // 顺时针旋转 90 度
8    Point rotCW90(const Point &a) { return Point(a.y, -a.x); }
9    // 点的幅角
10   double arg(const Point &a){
11       double t = atan2(a.y, a.x);
12       return t < 0 ? t + 2 * Pi:t;
13   }
```

## 线

```
1    // 是否平行
2    bool parallel(const line &a, const line &b) {
3        return !sgn(det(a.t - a.s, b.t - b.s));
4    }
5    // 直线是否相等
6    bool l_eq(const line& a, const line& b) {
7        return parallel(a, b) && parallel(line(a.s, b.t), line(b.s, a.t));
8    }
```

## 点与线

```
1    // 点是否在线段上, <= 包含端点
2    bool p_on_seg(const Point &p, const line &seg) {
3        return !sgn(det(p - seg.s, p - seg.t)) && sgn(dot(p - seg.s, p - seg.t)) <= 0;
4    }
5    // 点到直线距离
```

```
6   double dist_to_line(const Point &p, const line &l) {
7       return fabs(cross(l.s, l.t, p)) / abs(Point(l));
8   }
9   // 点到线段距离
10  double dist_to_seg(const Point &p, const line &l) {
11      if (l.s == l.t) return abs(p - l.s);
12      Vector vs = p - l.s, vt = p - l.t;
13      if (sgn(dot(Point(l), vs)) < 0) return abs(vs);
14      else if (sgn(dot(Point(l), vt)) > 0) return abs(vt);
15      else return dist_to_line(p, l);
16  }
```

## 线与线

```
1   // 直线交点，需保证存在
2   Point l_intersection(const line& a, const line& b) {
3       double s1 = det(Point(a), b.s - a.s), s2 = det(Point(a), b.t - a.s);
4       return (b.s * s2 - b.t * s1) / (s2 - s1);
5   }
6   // 线段和直线是否有交    1 = 规范, 2 = 不规范
7   int s_l_cross(const line &seg, const line &line) {
8       int d1 = sgn(cross(line.s, line.t, seg.s));
9       int d2 = sgn(cross(line.s, line.t, seg.t));
10      if ((d1 ^ d2) == -2) return 1; // proper
11      if (d1 == 0 || d2 == 0) return 2;
12      return 0;
13  }
14  // 线段的交    1 = 规范, 2 = 不规范
15  // 如果是不规范相交, p_on_seg 函数要改成 <=
16  int s_cross(const line &a, const line &b, Point &p) {
17      int d1 = sgn(cross(a.t, b.s, a.s)), d2 = sgn(cross(a.t, b.t, a.s));
18      int d3 = sgn(cross(b.t, a.s, b.s)), d4 = sgn(cross(b.t, a.t, b.s));
19      if ((d1 ^ d2) == -2 && (d3 ^ d4) == -2) { p = l_intersection(a, b); return 1; }
20      if (!d1 && p_on_seg(b.s, a)) { p = b.s; return 2; }
21      if (!d2 && p_on_seg(b.t, a)) { p = b.t; return 2; }
22      if (!d3 && p_on_seg(a.s, b)) { p = a.s; return 2; }
23      if (!d4 && p_on_seg(a.t, b)) { p = a.t; return 2; }
24      return 0;
25  }
```

## 多边形

```
1   #define nxt(i) ((i + 1) % s.size())
2   typedef vector<Point> Polygon
3   // 多边形面积
4   double poly_area(const Polygon &s){
5       double area = 0;
6       for(int i = 1; i < s.size() - 1; i++)
7           area += cross(s[i], s[i + 1], s[0]);
8       return area / 2;
9   }
10  // 多边形是否为凸多边形
11  int is_convex(const Polygon &s) {
12      int x = 1;
13      int sg = 0;
14      for(int i = 1; i < s.size(); i++) {
15          if(!sg && sgn(det(s[i] - s[i - 1], s[nxt(i)] - s[i])) != 0) {
16              sg = sgn(det(s[i] - s[i - 1], s[nxt(i)] - s[i]));
17          }
18          x &= (sgn(det(s[i] - s[i - 1], s[nxt(i)] - s[i])) == sg) ||
19          (sgn(det(s[i] - s[i - 1], s[nxt(i)] - s[i])) == 0);
20      }
21      return x;
22  }
23  // 点是否在多边形中 0 = 在外部 1 = 在内部 -1 = 在边界上
24  int p_in_poly(Point p, const Polygon &s){
25      int cnt = 0;
26      for(int i = 0; i < s.size(); i++) {
27          Point a = s[i], b = s[nxt(i)];
28          if (p_on_seg(p, line(a, b))) return -1;
```

```
29        //p 在多边形边上
30        if (sgn(a.y - b.y) <= 0) swap(a, b);
31        if (sgn(p.y - a.y) > 0) continue;
32        if (sgn(p.y - b.y) <= 0) continue;
33        //一条边包含它较高的点, 不包含较低的点
34        cnt += sgn(cross(b, a, p)) > 0;
35        //如果 p 在这条线段左边
36    }
37    return bool(cnt & 1);
38 }
```

## 凸包

andrew 算法,

```
1  Polygon Convex_hull(Polygon &s) {
2      sort(s.begin(), s.end());
3      Polygon ret(s.size() * 2);
4      int sz = 0;
5      for(int i = 0; i < s.size(); i++) {
6          while(sz > 1 && sgn(cross(ret[sz - 1], s[i], ret[sz - 2])) < 0) sz--;
7          ret[sz++] = s[i];
8      }
9      int k = sz;
10     for(int i = s.size() - 2; i >= 0; i--) {
11         while(sz > 1 && sgn(cross(ret[sz - 1], s[i], ret[sz - 2])) < 0) sz--;
12         ret[sz++] = s[i];
13     }
14     ret.resize(sz - (s.size() > 1));
15     return ret;
16 }
```

## 旋转卡壳

用平行线夹多边形, 根据两个向量的叉积判断支点变化

```
1  double rotatingCalipers(const Polygon &s) {
2      if(s.size() == 2) return abs(s[1] - s[0]);
3      int i = 0, j = 0;
4      for(int k = 0; k < s.size(); k++) {
5          if(!(s[i] < s[k])) i = k;
6          if(s[j] < s[k]) j = k;
7      }
8      double ans = 0;
9      int si = i, sj = j;
10     do{
11         ans = max(ans, abs(s[i] - s[j]));
12         if(sgn(det(s[nxt(i)] - s[i], s[nxt(j)] - s[j])) < 0)
13             i = nxt(i);
14         else j = nxt(j);
15     } while(i != si || j != sj);
16     return ans;
17 }
```

## 圆

### 直线和圆

```
1  // 直线与圆交点
2  vector<Point> l_c_intersection(const line &l, const circle &o) {
3      vector<Point> ret;
4      Point b = l.t - l.s, a = l.s - o.c;
5      double x = dot(b, b), y = dot(a, b), z = dot(a, a) - o.r * o.r;
6      double D = y * y - x * z;
7      if (sgn(D) < 0) return ret;
8      ret.push_back(o.c + a + b * (-y + sqrt(D + eps)) / x);
9      if (sgn(D) > 0) ret.push_back(o.c + a + b * (-y - sqrt(D)) / x);
10     return ret;
11 }
12 // 点到圆的切点
```

```cpp
vector<Point> p_c_tangent(const Point &p, const circle &o) {
    vector<Point> ret;
    double d = abs(p - o.c), x = dot(p - o.c, p - o.c) - o.r * o.r;
    if(sgn(x) < 0) ;
    else if(sgn(x) == 0) ret.push_back(p);
    else {
        Vector base = p + (o.c - p) * x / dot(p - o.c, p - o.c);
        Vector e = rotCW90(o.c - p) / d;
        ret.push_back(base + e * sqrt(x) * o.r / d);
        ret.push_back(base - e * sqrt(x) * o.r / d);
    }
    return ret;
}
// 圆与圆的交点
vector<Point> c_c_intersection(const circle &a, const circle &b) {
    vector<Point> ret;
    double d = abs(b.c - a.c);
    if(sgn(d) == 0 || sgn(d - a.r - b.r) > 0 || sgn(d + min(a.r, b.r) - max(a.r, b.r)) < 0)
        return ret;
    double x = (a.r * a.r + dot(b.c - a.c, b.c - a.c) - b.r * b.r) / (2 * d);
    double y = sqrt(a.r * a.r - x * x);
    Point v = (b.c - a.c) / d;
    ret.push_back(a.c + v * x + rotCW90(v) * y);
    if(sgn(y) > 0) ret.push_back(a.c + v * x - rotCW90(v) * y);
    return ret;
}
```

在 res 中存放的线上的两点分别是在 c1,c2 上的切点。

```cpp
int tangent(const circle &C1, const circle &C2, vector<line> &res){
    double d = abs(C1.c - C2.c);
    if(d < eps) return 0;

    int c=0;
    // 内公切线
    if(C1.r + C2.r < d - eps){
        double t = acos((C1.r + C2.r) / d);
        res.push_back(line(C1.c + rot(C1.r / d * (C2.c - C1.c), t), C2.c + rot(C2.r / d * (C1.c - C2.c), t)));
        res.push_back(line(C1.c + rot(C1.r / d * (C2.c - C1.c),-t), C2.c + rot(C2.r / d * (C1.c - C2.c),-t)));
        c += 2;
    } else if(C1.r + C2.r < d + eps){
        Point p = C1.c + C1.r / d * (C2.c - C1.c);
        res.push_back(line(p, p + rot(C2.c - C1.c, Pi / 2)));
        c++;
    }

    // 外公切线
    if(abs(C1.r - C2.r) < d - eps){
        double t1 = acos((C1.r - C2.r) / d), t2 = Pi - t1;
        res.push_back(line(C1.c + rot(C1.r / d * (C2.c - C1.c), t1), C2.c + rot(C2.r / d * (C1.c - C2.c),-t2)));
        res.push_back(line(C1.c + rot(C1.r / d * (C2.c - C1.c),-t1), C2.c + rot(C2.r / d * (C1.c - C2.c), t2)));
        c+=2;
    } else if(abs(C1.r - C2.r) < d + eps){
        Point p = C1.c + C1.r / d * (C2.c - C1.c);
        res.push_back(line(p, p + rot(C2.c - C1.c, Pi / 2)));
        c++;
    }

    return c;
}
```

## tips

- atan2(y, x) 函数: 点 $(x, y)$ 到原点的方位角，值域在 $(-\pi, \pi)$ 在一二象限为正，三四象限为负。

# 字符串

## 字串哈希

```cpp
namespace String {
    const int x = 135;
    const int p1 = 1e9 + 7, p2 = 1e9 + 9;
    ull xp1[N], xp2[N], xp[N];
    void init_xp() {
        xp1[0] = xp2[0] = xp[0] = 1;
        for(int i = 1; i < N; i++) {
            xp1[i] = xp1[i - 1] * x % p1;
            xp2[i] = xp2[i - 1] * x % p2;
            xp[i] = xp[i - 1] * x;
        }
    }
    struct HashString {
        char s[N];
        int length, subsize;
        bool sorted;
        ull h[N], hl[N];
        ull init(const char *t) {
            if(xp[0] != 1) init_xp();
            length = strlen(t);
            strcpy(s, t);
            ull res1 = 0, res2 = 0;
            h[length] = 0;
            for(int j = length - 1; j >= 0; j--) {
                #ifdef ENABLE_DOUBLE_HASH
                res1 = (res1 * x + s[j]) % p1;
                res2 = (res2 * x + s[j]) % p2;
                h[j] = (res1 << 32) | res2;
                #else
                res1 = res1 * x + s[j];
                h[j] = res1;
                #endif
            }
            return h[0];
        }
        //获取子串哈希，左闭右开
        ull get_substring_hash(int left, int right)  {
            int len = right - left;
            #ifdef ENABLE_DOUBLE_HASH
            unsigned int mask32 = ~(0u);
            ull left1 = h[left] >> 32, right1 = h[right] >> 32;
            ull left2 = h[left] & mask32, right2 = h[right] & mask32;
            return (((left1 - right1 * xp1[len] % p1 + p1) % p1) << 32) |
                    (((left2 - right2 * xp2[len] % p2 + p2) % p2));
            #else
            return h[left] - h[right] * xp[len];
            #endif
        }
        void get_all_subs_hash(int sublen) {
            subsize = length - sublen + 1;
            for (int i = 0; i < subsize; ++i)
                hl[i] = get_substring_hash(i, i + sublen);
            sorted = 0;
        }

        void sort_substring_hash() {
            sort(hl, hl + subsize);
            sorted = 1;
        }

        bool match(ull key) const {
            if (!sorted) assert (0);
            if (!subsize) return false;
            return binary_search(hl, hl + subsize, key);
        }
    };
}
```

## Trie

```
1   namespace trie {
2       int t[N][26], sz, ed[N];
3       int _new() {
4           sz++;
5           memset(t[sz], 0, sizeof(t[sz]));
6           return sz;
7       }
8       void init() {
9           sz = 0;
10          _new();
11          memset(ed, 0, sizeof(ed));
12      }
13      void Insert(char *s, int n) {
14          int u = 1;
15          for(int i = 0; i < n; i++) {
16              int c = s[i] - 'a';
17              if(!t[u][c]) t[u][c] = _new();
18              u = t[u][c];
19          }
20          ed[u]++;
21      }
22      int find(char *s, int n) {
23          int u = 1;
24          for(int i = 0; i < n; i++) {
25              int c = s[i] - 'a';
26              if(!t[u][c]) return -1;
27              u = t[u][c];
28          }
29          return u;
30      }
31  }
```

## KMP 算法

```
1   namespace KMP {
2       void get_next(char *t, int m, int *nxt) {
3           int j = nxt[0] = 0;
4           for(int i = 1; i < m; i++) {
5               while(j && t[i] != t[j]) j = nxt[j - 1];
6               nxt[i] = j += (t[i] == t[j]);
7           }
8       }
9       vector<int> find(char *t, int m, int *nxt, char *s, int n) {
10          vector<int> ans;
11          int j = 0;
12          for(int i = 0; i < n; i++) {
13              while(j && s[i] != t[j]) j = nxt[j - 1];
14              j += s[i] == t[j];
15              if(j == m) {
16                  ans.push_back(i - m + 1);
17                  j = nxt[j - 1];
18              }
19          }
20          return ans;
21      }
22  }
```

## manacher 算法

```
1   namespace manacher {
2       char s[N];
3       int p[N], len;
4       void getp(string tmp) {
5           len = 0;
6           for(auto x : tmp) {
7               s[len++] = '#';
8               s[len++] = x;
9           }
```

```
10          s[len++] = '#';
11          memset(p, 0, sizeof(int) * (len + 10));
12          int c = 0, r = 0;
13          for(int i = 0; i < len; i++) {
14              if(i <= r) p[i] = min(p[2 * c - i], r - i);
15              else p[i] = 1;
16              while(i - p[i] >= 0 && i + p[i] < len && s[i - p[i]] == s[i + p[i]])
17                  p[i]++;
18              if(i + p[i] - 1 > r) {
19                  r = i + p[i] - 1;
20                  c = i;
21              }
22          }
23          for(int i = 0; i < len; i++) p[i]--;
24      }
25      void getp(char *tmp, int n) {
26          len = 0;
27          for(int i = 0; i < n; i++) {
28              s[len++] = '#';
29              s[len++] = tmp[i];
30          }
31          s[len++] = '#';
32          memset(p, 0, sizeof(int) * (len + 10));
33          int c = 0, r = 0;
34          for(int i = 0; i < len; i++) {
35              if(i <= r) p[i] = min(p[2 * c - i], r - i);
36              else p[i] = 1;
37              while(i - p[i] >= 0 && i + p[i] < len && s[i - p[i]] == s[i + p[i]])
38                  p[i]++;
39              if(i + p[i] - 1 > r) {
40                  r = i + p[i] - 1;
41                  c = i;
42              }
43          }
44          for(int i = 0; i < len; i++) p[i]--;
45      }
46      int getlen() {
47          return *max_element(p, p + len);
48      }
49      int getlen(string s) {
50          getp(s);
51          return getlen();
52      }
53  }
```

## AC 自动机

```
1  struct ac_automaton {
2      int t[N][26], danger[N], tot, fail[N];
3      int dp[N][N];
4      void init() {
5          tot = -1;
6          _new();
7      }
8      int _new() {
9          tot++;
10          memset(t[tot], 0, sizeof(t[tot]));
11          danger[tot] = 0;
12          fail[tot] = 0;
13          return tot;
14      }
15      void Insert(const char *s) {
16          int u = 0;
17          for(int i = 0; s[i]; i++) {
18              if(!t[u][mp[s[i]]]) t[u][s[i] - 'a'] = _new();
19              u = t[u][mp[s[i]]];
20          }
21          danger[u] = 1;
22      }
23      void build() {
24          queue<int> q;
```

```cpp
25              for(int i = 0; i < 26; i++) {
26                  if(t[0][i]) {
27                      fail[i] = 0;
28                      q.push(t[0][i]);
29                  }
30              }
31              while(q.size()) {
32                  int u = q.front(); q.pop();
33                  danger[u] |= danger[fail[u]];
34                  for(int i = 0; i < 26; i++) {
35                      if(t[u][i]) {
36                          fail[t[u][i]] = t[fail[u]][i];
37                          q.push(t[u][i]);
38                      } else t[u][i] = t[fail[u]][i];
39                  }
40              }
41          }
42          int query(const char *s) {
43              memset(dp, 0x3f, sizeof(dp));
44              int n = strlen(s);
45              dp[0][0] = 0;
46              for(int i = 0; i < n; i++) {
47                  for(int j = 0; j <= tot; j++) if(!danger[j]) {
48                      for(int k = 0; k < 26; k++) if(!danger[t[j][k]]) {
49                          dp[i + 1][t[j][k]] = min(dp[i + 1][t[j][k]], dp[i][j] + (s[i] - 'a' != k));
50                      }
51                  }
52              }
53              int ans = 0x3f3f3f3f;
54              for(int i = 0; i <= tot; i++) if(!danger[i]) {
55                  ans = min(ans, dp[n][i]);
56              }
57              return ans == 0x3f3f3f3f ? -1 : ans;
58          }
59      };
```

# 杂项

## int128

```cpp
1   typedef __uint128_t u128;
2   inline u128 read() {
3       static char buf[100];
4       scanf("%s", buf);
5       // std::cin >> buf;
6       u128 res = 0;
7       for(int i = 0;buf[i];++i) {
8           res = res << 4 | (buf[i] <= '9' ? buf[i] - '0' : buf[i] - 'a' + 10);
9       }
10      return res;
11  }
12  inline void output(u128 res) {
13      if(res >= 16)
14          output(res / 16);
15      putchar(res % 16 >= 10 ? 'a' + res % 16 - 10 : '0' + res % 16);
16      //std::cout.put(res % 16 >= 10 ? 'a' + res % 16 - 10 : '0' + res % 16);
17  }
```

## Java,BigInteger

```java
1   public BigInteger add(BigInteger val)      返回当前大整数对象与参数指定的大整数对象的和
2   public BigInteger subtract(BigInteger val)   返回当前大整数对象与参数指定的大整数对象的差
3   public BigInteger multiply(BigInteger val)    返回当前大整数对象与参数指定的大整数对象的积
4   public BigInteger devide(BigInteger val)     返回当前大整数对象与参数指定的大整数对象的商
5   public BigInteger remainder(BigInteger val)    返回当前大整数对象与参数指定的大整数对象的余
6   public int compareTo(BigInteger val)      返回当前大整数对象与参数指定的大整数对象的比较结果，返回值是 1、−1、0，分别表示当前大整数对象大
    ↪   于、小于或等于参数指定的大整数。
7   public BigInteger abs()       返回当前大整数对象的绝对值
8   public BigInteger pow(int exponent)     返回当前大整数对象的 exponent 次幂。
```

```
9   public String toString()      返回当前当前大整数对象十进制的字符串表示。
10  public String toString(int p)    返回当前大整数对象 p 进制的字符串表示。
11  public BigInteger negate()    返回当前大整数的相反数。
```

## 奇技淫巧

**_builtin_ 内建函数 **

- ~~__builtin_popcount(unsigned int n) 该函数是判断 n 的二进制中有多少个 1~~

- __builtin_parity(unsigned int n) 该函数是判断 n 的二进制中 1 的个数的奇偶性

- __builtin_ffs(unsigned int n) 该函数判断 n 的二进制末尾最后一个 1 的位置，从一开始

- __builtin_ctz(unsigned int n) 该函数判断 n 的二进制末尾后面 0 的个数，当 n 为 0 时，和 n 的类型有关

- __builtin_clz (unsigned int x) 返回前导的 0 的个数

真·popcount

```
1   int _popcount(int x) {
2       return __builtin_popcount(x & (0ull - 1)) + __builtin_popcount(x >> 32);
3   }
```

### 随机数种子

```
1   srand(std :: chrono :: system_clock :: now().time_since_epoch().count());
```

### T(5) 求任意 int log2

```
1   inline int LOG2_1(unsigned x){
2       static const int tb[32]={0,9,1,10,13,21,2,29,11,14,16,18,22,25,3,30,8,12,20,28,15,17,24,7,19,27,23,6,26,5,4,31};
3       x|=x>>1; x|=x>>2; x|=x>>4; x|=x>>8; x|=x>>16;
4       return tb[x*0x07C4ACDDu>>27];
5   }
```

### O(1) 求 2 的整幂次 log2

```
1   inline int LOG2(unsigned x){  //x=2^k
2       static const int tb[32]={31,0,27,1,28,18,23,2,29,21,19,12,24,9,14,3,30,26,17,22,20,11,8,13,25,16,10,7,15,6,5,4};
3       return tb[x*263572066>>27];
4   }
```

### 开启编译优化

```
1   作者：qwqwqer
2   链接：https://www.zhihu.com/question/264251178/answer/2155420801
3   来源：知乎
4   著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
5
6   #pragma GCC optimize(2)
7   #pragma GCC optimize(3)
8   #pragma GCC optimize("Ofast")
9   #pragma GCC optimize("inline")
10  #pragma GCC optimize("-fgcse")
11  #pragma GCC optimize("-fgcse-lm")
12  #pragma GCC optimize("-fipa-sra")
13  #pragma GCC optimize("-ftree-pre")
14  #pragma GCC optimize("-ftree-vrp")
15  #pragma GCC optimize("-fpeephole2")
16  #pragma GCC optimize("-ffast-math")
17  #pragma GCC optimize("-fsched-spec")
18  #pragma GCC optimize("unroll-loops")
19  #pragma GCC optimize("-falign-jumps")
20  #pragma GCC optimize("-falign-loops")
21  #pragma GCC optimize("-falign-labels")
22  #pragma GCC optimize("-fdevirtualize")
23  #pragma GCC optimize("-fcaller-saves")
24  #pragma GCC optimize("-fcrossjumping")
25  #pragma GCC optimize("-fthread-jumps")
26  #pragma GCC optimize("-funroll-loops")
```

```
27    #pragma GCC optimize("-fwhole-program")
28    #pragma GCC optimize("-freorder-blocks")
29    #pragma GCC optimize("-fschedule-insns")
30    #pragma GCC optimize("inline-functions")
31    #pragma GCC optimize("-ftree-tail-merge")
32    #pragma GCC optimize("-fschedule-insns2")
33    #pragma GCC optimize("-fstrict-aliasing")
34    #pragma GCC optimize("-fstrict-overflow")
35    #pragma GCC optimize("-falign-functions")
36    #pragma GCC optimize("-fcse-skip-blocks")
37    #pragma GCC optimize("-fcse-follow-jumps")
38    #pragma GCC optimize("-fsched-interblock")
39    #pragma GCC optimize("-fpartial-inlining")
40    #pragma GCC optimize("no-stack-protector")
41    #pragma GCC optimize("-freorder-functions")
42    #pragma GCC optimize("-findirect-inlining")
43    #pragma GCC optimize("-fhoist-adjacent-loads")
44    #pragma GCC optimize("-frerun-cse-after-loop")
45    #pragma GCC optimize("inline-small-functions")
46    #pragma GCC optimize("-finline-small-functions")
47    #pragma GCC optimize("-ftree-switch-conversion")
48    #pragma GCC optimize("-foptimize-sibling-calls")
49    #pragma GCC optimize("-fexpensive-optimizations")
50    #pragma GCC optimize("-funsafe-loop-optimizations")
51    #pragma GCC optimize("inline-functions-called-once")
52    #pragma GCC optimize("-fdelete-null-pointer-checks")
```

## 快速乘

```
1    ll mul(ll x, ll y, ll mod){
2        return (x * y - (ll)((long double)x / mod * y) * mod + mod) % mod;
3    }
4    ll mul(ll a, ll b, ll MOD) {
5        __int128 x = a, y = b, m = MOD;
6        return (ll)(x * y % m);
7    }
```

## 子集枚举

枚举 s 的子集

```
1    for(int i = s; i; i = (i - 1) & s))
```

枚举所有大小为 r 的集合

```
1    for(int s = (1 << r) - 1; s < (1 << n); ) {
2        int x = s & -s;
3        int y = s + x;
4        s = ((y ^ s) >> __builtin_ctz(x) + 2) | y;
5    }
```

## mt19937_64 随机数生成器

```
1    std::mt19937_64 rng(std::chrono::steady_clock::now().time_since_epoch().count());
2    template <typename T>
3    T rd(T l, T r) {
4        std::uniform_int_distribution<T> u(l, r);
5        return u(rng);
6    }
7    template <>
8    double rd<double>(double l, double r) {
9        std::uniform_real_distribution<double> u(l, r);
10       return u(rng);
11   }
```

## tips:

- 如果使用 sort 比较两个函数，不能出现 $a < b$ 和 $a > b$ 同时为真的情况，否则会运行错误。

- 多组数据清空线段树的时候，不要忘记清空全部数组（比如说 lazytag 数组）。

- 注意树的深度和节点到根的距离是两个不同的东西，深度是点数，距离是边长，如果求 LCA 时用距离算会出错。

- 连通性专题：注意判断 $dfn[x]$ 和 $low[y]$ 的关系时是否不小心两个都达成 $low$ 了

- 推不等式确定范围的时候，仅需要考虑所有不等式限定的范围，然后判断左端点是否大于右端点，不要加额外的臆想条件。

- 矩阵快速幂如果常数十分大的时候，可以考虑 unordered_map 保存结果，可以明显加速。

- **__builtin_popcount** 只支持 unsigned int 型，不支持 long long!!!!!!!!