Standard Code Library

ONGLU

North Eastern University

August 2021

Contents

初始化

Clion 环境配置

program : g++ argument :

-std=c++14 "$FileDir$\$FileName$" -o "$ProjectFileDir$\a.exe" -D MYLOCAL -Wl,--stack,524288000

数据结构

树分治

点分治，思想是枚举所有路径，但是路径要支持可合并。
递归切割整棵树，无根树选定重心作为根。
枚举每棵子树向上到达根的路径，与其他子树的答案合并，组成经过当前根节点的所有路径。
计算完经过当前点的路径后，递归计算每棵子树，同样也是选取重心作为分割点。

```
const int M = 1e7 + 1009;
const int N = 2e5 + 1009;
struct Edge {
    int to, w;
};
int sum;
int n, m, q[N], ans[N], rt;
int maxSize[N], siz[N], vis[N];
vector<Edge> ver[N];
void findRoot(int x, int pre) {
    maxSize[x] = 0;
    siz[x] = 1;
    for(auto e : ver[x]) {
        int y = e.to, w = e.w;
        if(y == pre || vis[y]) continue;
        findRoot(y, x);
        maxSize[x] = max(maxSize[x], siz[y]);
        siz[x] += siz[y];
    }
    // 这里一定要是 sum 而不能是 n，因为要切割子树
    maxSize[x] = max(maxSize[x], sum - siz[x]);
    if(rt == -1 || maxSize[x] < maxSize[rt]) rt = x;
}
int dis[N], distot, bul[M];
void getDist(int x, int pre, int dd) {
    dis[++distot] = dd;
    for(auto e : ver[x]) {
        int y = e.to, w = e.w;
        if(y != pre && !vis[y]) {
            getDist(y, x, dd + w);
        }
    }
}
queue<int> clr;
void dfz(int x) {
    vis[x] = 1;
    bul[0] = 1;
    clr.push(0);
    for(auto e : ver[x]) {
        // 一定要每棵子树单独计算链，否则会出现两条同一子树内的链合并的情况，造成多算
        int y = e.to, w = e.w;
        if(vis[y]) continue;
        distot = 0;
        getDist(y, x, w);
        for(int i = 1; i <= distot; i++) {
            for(int j = 1; j <= m; j++) {
                if(q[j] >= dis[i] && bul[q[j] - dis[i]]) {
                    ans[j] = 1;
                }
```

```
50                }
51            }
52            // 合并完子树和其他树的链后，再将链加入桶
53            for(int i = 1; i <= distot; i++) if(dis[i] < M) bul[dis[i]] = 1, clr.push(dis[i]);
54        }
55        while(!clr.empty()) bul[clr.front()] = 0, clr.pop();
56        for(auto e : ver[x]) {
57            int y = e.to, w = e.w;
58            if(vis[y]) continue;
59            // 子树大小一定要设置成 siz[y]，否则复杂度出错
60            sum = siz[y];
61            rt = -1;
62            findRoot(y, x);
63            findRoot(rt, rt);
64            dfz(rt);
65        }
66    }
67    void work() {
68        cin >> n >> m;
69        for(int i = 1; i < n; i++) {
70            int x, y, w;
71            cin >> x >> y >> w;
72            ver[x].push_back({y, w});
73            ver[y].push_back({x, w});
74        }
75        for(int i = 1; i <= m; i++) cin >> q[i];
76        rt = -1;
77        sum = n;
78        findRoot(1, 1);
79        findRoot(rt, rt);
80        dfz(rt);
81        for(int i = 1; i <= m; i++) {
82            if(ans[i]) cout << "AYE" << endl;
83            else cout << "NAY" << endl;
84        }
85    }
```

线段树上二分

要先 check 是否存在第 k 大线段树上权值 <10，判断 L,R 右边权值都等于 0 或者 9 的最长长度。

```
1    int countval(int l, int r, int rt, int L, int R, int val) {
2        if(L == l && R == r && sum[rt] == val * (r - l + 1)) {
3            return (r - l + 1);
4        }
5        if(l == r) return sum[rt] == val;
6        pushdown(l, r, rt);
7        if(L > Mid) return countval(Mid + 1, r, rson, L, R, val);
8        if(R <= Mid) return countval(l, Mid, lson, L, R, val);
9        int ans = countval(Mid + 1, r, rson, Mid + 1, R, val);
10        if(ans < (R - Mid)) return ans;
11        ans += countval(l, Mid, lson, L, Mid, val);
12        return ans;
13    }
```

ST 表

```
1    int query(int l, int r) {
2        int p = log2(r - l + 1);
3        return max(st[p][l], st[p][r - (1 << p) + 1]);
4    }
5    void init() {
6        for(int i = 2; i <= n; i++) LOG2[i] = LOG2[i / 2] + 1;
7        for(int i = 1; i <= n; i++) st[0][i] = h[i];
8        for(int i = 1; i < 25; i++) {
9            for(int j = 1; j + (1 << i - 1) <= n; j++) {
10                st[i][j] = max(st[i - 1][j], st[i - 1][j + (1 << i - 1)]);
11            }
12        }
13    }
```

二维哈希

```cpp
ull hs[109][109], pw1[10009], pw2[100009];
ull gethash(int lx, int ly, int rx, int ry) {
    ull hs1 = hs[lx][ly] - pw2[ry - ly + 1] * hs[lx][ry + 1];
    ull hs2 = hs[rx + 1][ly] - pw2[ry - ly + 1] * hs[rx + 1][ry + 1];
    return hs1 - pw1[rx - lx + 1] * hs2;
}
pw1[0] = pw2[0] = 1;
for(int i = 1; i <= 1000; i++) pw1[i] = pw1[i - 1] * 19260817;
for(int i = 1; i <= 1000; i++) pw2[i] = pw2[i - 1] * 135;
for(int i = n; i >= 1; i--) {
    for(int j = 1; j <= m; j++) {
        if(i == n) hs[i][j] = sum[i][j] + 2;
        else hs[i][j] = hs[i + 1][j] * 19260817 + sum[i][j] + 2;
    }
}
for(int i = 1; i <= n; i++) {
    for(int j = m - 1; j; j--) {
        hs[i][j] = hs[i][j + 1] * 135 + hs[i][j];
    }
}
```

线性基

最大异或和

```cpp
struct Basis {
    long long base[64];
    void insert(long long x) {
        for(int i = 62; i >= 0; i--) {
            if(x == 0) return;
            if(x >> i & 1) {
                if(base[i] != 0) x ^= base[i];
                else {
                    base[i] = x;
                    break;
                }
            }
        }
    }
    int contains(long long x) {
        for(int i = 62; i >= 0; i--) if(x >> i & 1) {
            if(!base[i]) return 0;
            x ^= base[i];
        }
        return 1;
    }
    int maxContains() {
        long long ans = 0;
        for(int i = 62; i >= 0; i--) {
            if(ans >> i & 1) continue;
            ans ^= base[i];
        }
        return ans;
    }
};
```

轻重链剖分

```cpp
void dfs1(int x, int pre) {
    siz[x] = 1; mson[x] = 0;
    dth[x] = dth[pre] + 1;
    fa[x] = pre;
    for(auto y : son[x]) if(y != pre) {
        dfs1(y, x);
        siz[x] += siz[y];
        if(!mson[x] || siz[y] > siz[mson[x]])
            mson[x] = y;
    }
```

```
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。

```cpp
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;
```

```cpp
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，第二维，第三维的顺序排序，再去重。目的是为了保证右边元素不会对左边元素产生贡献。

然后对第二维归并，归并时计算左边第三维小于等于右边第三维的数量。

注意清空树状数组的时候要与值域相关。

```cpp
1   //
2   // Created by onglu on 2022/8/3.
3   //
4
5   #include <bits/stdc++.h>
6
7   #define all(a) a.begin(),a.end()
8   #define rall(a) a.rbegin(),a.rend()
9
10  #define endl '\n'
11  #define lson (rt << 1)
12  #define rson (rt << 1 | 1)
13  #define Mid ((l + r) / 2)
14  //#define int long long
15  using namespace std;
```

```
16    const int N = 2e6 + 1009;
17    //const int N = 2e5 + 1009;
18    //const int N = 5009;
19    //const int N = 309;
20    int n, m;
21    struct Point {
22        int id, x, y, z, cnt, ans;
23    } a[N], b[N];
24    int tree[N];
25    void add(int x, int y) {
26        for(; x <= m; x += x & -x)
27            tree[x] += y;
28    }
29    void clear(int x) {
30        for( ; x <= m; x += x & -x)
31            tree[x] = 0;
32    }
33    int query(int x) {
34        int ans = 0;
35        for( ; x; x -= x & -x)
36            ans += tree[x];
37        return ans;
38    }
39    void solve(int l, int r) {
40        if(l == r) return ;
41        solve(l, Mid); solve(Mid + 1, r);
42        int i = l, tot = i;
43        for(int j = Mid + 1; j <= r; j++) {
44            while(i <= Mid && a[i].y <= a[j].y) {
45                add(a[i].z, a[i].cnt);
46                b[tot++] = a[i++];
47            }
48            a[j].ans += query(a[j].z);
49            b[tot++] = a[j];
50        }
51
52        while(i <= Mid) b[tot++] = a[i++];
53        for(int j = l; j <= r; j++) {
54            clear(b[j].z);
55            a[j] = b[j];
56        }
57    }
58    void work() {
59        cin >> n >> m;
60        for(int i = 1; i <= n; i++) {
61            cin >> a[i].x >> a[i].y >> a[i].z;
62            a[i].id = i;
63        }
64        sort(a + 1, a + 1 + n, [](const Point &a, const Point &b) {
65            if(a.x != b.x) return a.x < b.x;
66            if(a.y != b.y) return a.y < b.y;
67            return a.z < b.z;
68        });
69        int tot = 0;
70        for(int i = 1; i <= n; i++) {
71            if(i == 0 || a[i].x != a[i - 1].x || a[i].y != a[i - 1].y || a[i].z != a[i - 1].z) {
72                a[++tot] = a[i];
73                a[tot].cnt = 1;
74            } else {
75                a[tot].cnt++;
76            }
77        }
78        solve(1, n);
79        vector<int> ans_cnt(n + 1);
80        for(int i = 1; i <= n; i++) {
81            ans_cnt[a[i].ans + a[i].cnt - 1] += a[i].cnt;
82        }
83        for(int i = 0; i < n; i++) cout << ans_cnt[i] << endl;
84    }
```

整体二分求区间静态 k 小

二分时用 vector 存在答案在 [l,r] 之间的询问。统计答案 [l, Mid] 对所有询问的影响，影响足够的放到 [l, Mid] 中继续处理。影响不够的**消除影响后**放到 [Mid + 1, r] 中继续处理。

```cpp
//
// Created by onglu on 2022/8/3.
//

#define Mid ((l + r) / 2)
using namespace std;
const int N = 2e6 + 1009;
int n, m, a[N], ans[N];
struct query {
    int id, l, r, k;
};
vector<int> b;
vector<int> pos[N];
void solve(int l, int r, vector<query> v) {
    vector<int> nums;
    for(int i = l; i <= Mid; i++) {
        for(auto x : pos[i]) {
            nums.push_back(x);
        }
    }
    std::sort(nums.begin(), nums.end());
    vector<query> vl, vr;
    for(auto x : v) {
        int lpos = std::lower_bound(nums.begin(), nums.end(),x.l) - nums.begin();
        int rpos = std::upper_bound(nums.begin(), nums.end(),x.r) - nums.begin();
        if(rpos - lpos >= x.k) {
            vl.push_back(x);
        } else {
            vr.push_back(x);
            vr.back().k -= (rpos - lpos);
        }
    }
    if(l == r) {
        for(auto x : vl) ans[x.id] = l;
        for(auto x : vr) ans[x.id] = -1;
        return ;
    }
    if(vl.size()) solve(l, Mid, vl);
    if(vr.size()) solve(Mid + 1, r, vr);
}
void work() {
    cin >> n >> m;
    for(int i = 1; i <= n; i++) {
        cin >> a[i];
        b.push_back(a[i]);
    }
    std::sort(b.begin(), b.end());
    b.resize(std::unique(b.begin(), b.end()) - b.begin());
    for(int i = 1; i <= n; i++) {
        a[i] = std::lower_bound(b.begin(), b.end(), a[i]) - b.begin() + 1;
        pos[a[i]].push_back(i);
    }
    for(int i = 1; i <= b.size(); i++) {
        std::sort(pos[i].begin(), pos[i].end());
    }
    vector<query> v(m);
    for(int i = 0; i < m; i++) {
        v[i].id = i;
        cin >> v[i].l >> v[i].r >> v[i].k;
    }
    solve(1, b.size(), v);
    for(int i = 0; i < m; i++) {
        if(ans[i] == -1) cout << "No Answer" << endl;
        else cout << b[ans[i] - 1] << endl;
    }
}
```

Link Cut Tree

access(x)：将 x 与 x 所在树的路径构造到同一个 splay 里面。makeroot(x)：将 x 置为树的根。

```
1   struct LCT {
2       int fa[N], ch[N][2], val[N], sum[N], tmp[N];
3       bool rev[N];
4       bool isroot(int x) {return !fa[x] || ch[fa[x]][0] != x && ch[fa[x]][1] != x;}
5       void pushrev(int x) {if(!x)return; swap(ch[x][0], ch[x][1]); rev[x] ^= 1;}
6       void pushdown(int x) {if(rev[x]) {pushrev(ch[x][0]); pushrev(ch[x][1]); rev[x] = 0;}}
7       void update(int x) {
8           sum[x] = val[x];
9           if(ch[x][0]) sum[x] += sum[ch[x][0]];
10          if(ch[x][1]) sum[x] += sum[ch[x][1]];
11      }
12      void rotate(int x) {
13          int y = fa[x], w = ch[y][1] == x;
14          ch[y][w] = ch[x][w ^ 1];
15          if(ch[x][w ^ 1]) fa[ch[x][w ^ 1]] = y;
16          if(fa[y]) {
17              int z = fa[y];
18              if(ch[z][0] == y) ch[z][0] = x; else if(ch[z][1] == y) ch[z][1] = x;
19          }
20          fa[x] = fa[y]; fa[y] = x; ch[x][w ^ 1] = y; update(y);
21      }
22      void splay(int x) {
23          int s = 1, i = x, y; tmp[1] = i;
24          while(!isroot(i)) tmp[++s] = i = fa[i];
25          while(s) pushdown(tmp[s--]);
26          while(!isroot(x)) {
27              y = fa[x];
28              if(!isroot(y)) {
29                  if((ch[fa[y]][0] == y) ^ (ch[y][0] == x)) rotate(x);
30                  else rotate(y);
31              }
32              rotate(x);
33          }
34          update(x);
35      }
36      int access(int x) { int y; for(y = 0; x; y = x, x = fa[x]) {splay(x); ch[x][1] = y; update(x);}return y;}
37      int root(int x) { access(x); splay(x); while(ch[x][0]) pushdown(x), x = ch[x][0]; return x; }
38      void makeroot(int x) { access(x); splay(x); pushrev(x); }
39      void link(int x, int y) { makeroot(x); if(root(y) != x) fa[x] = y;}
40      void cut(int x, int y) {
41          makeroot(x);
42          if(root(y) == x && fa[x] == y && ch[y][0] == x) {
43              fa[x] = ch[y][0] = 0;
44              update(x);
45          }
46      }
47      int ask(int x, int y) { makeroot(x); access(y); splay(y); return sum[y]; }
48      int lca(int root, int x, int y) {return makeroot(root), access(x), access(y);}
49      void fix(int x, int y) { splay(x); val[x] = y; update(x);}
50  };
```

带子树查询的 LCT（[BJOI2014] 大融合）

```
1   struct LCT {
2       int fa[N], ch[N][2], siz[N], siz2[N], tmp[N];
3       bool rev[N];
4       bool isroot(int x) {return !fa[x] || ch[fa[x]][0] != x && ch[fa[x]][1] != x;}
5       void pushrev(int x) {if(!x)return; swap(ch[x][0], ch[x][1]); rev[x] ^= 1;}
6       void pushdown(int x) {if(rev[x]) {pushrev(ch[x][0]); pushrev(ch[x][1]); rev[x] = 0;}}
7       void update(int x) {
8           siz[x] = 1 + siz2[x];
9           if(ch[x][0]) siz[x] += siz[ch[x][0]];
10          if(ch[x][1]) siz[x] += siz[ch[x][1]];
11      }
12      void rotate(int x) {
13          int y = fa[x], w = ch[y][1] == x;
14          ch[y][w] = ch[x][w ^ 1];
15          if(ch[x][w ^ 1]) fa[ch[x][w ^ 1]] = y;
```

```
16          if(fa[y]) {
17              int z = fa[y];
18              if(ch[z][0] == y) ch[z][0] = x; else if(ch[z][1] == y) ch[z][1] = x;
19          }
20          fa[x] = fa[y]; fa[y] = x; ch[x][w ^ 1] = y; update(y);
21      }
22      void splay(int x) {
23          int s = 1, i = x, y; tmp[1] = i;
24          while(!isroot(i)) tmp[++s] = i = fa[i];
25          while(s) pushdown(tmp[s--]);
26          while(!isroot(x)) {
27              y = fa[x];
28              if(!isroot(y)) {
29                  if((ch[fa[y]][0] == y) ^ (ch[y][0] == x)) rotate(x);
30                  else rotate(y);
31              }
32              rotate(x);
33          }
34          update(x);
35      }
36      void access(int x) {
37          for(int y = 0; x; y = x, x = fa[x]) {
38              splay(x);
39              if(ch[x][1]) siz2[x] += siz[ch[x][1]];
40              if(y) siz2[x] -= siz[y];
41              ch[x][1] = y;
42              update(x);
43          }
44      }
45      int root(int x) { access(x); splay(x); while(ch[x][0]) pushdown(x), x = ch[x][0]; return x; }
46      void makeroot(int x) { access(x); splay(x); pushrev(x); }
47      void link(int x, int y) {
48          makeroot(x);
49          if(root(y) != x) {
50              fa[x] = y;
51              siz2[y] += siz[x];
52          }
53      }
54      void cut(int x, int y) {
55          makeroot(x);
56          if(root(y) == x && fa[x] == y && ch[y][0] == x) {
57              fa[x] = ch[y][0] = 0;
58              update(x);
59              update(y);
60          }
61      }
62  } lct;
63  int n, m;
64  void work() {
65      cin >> n >> m;
66      for(int i = 1; i <= m; i++) {
67          char c;
68          int x, y;
69          cin >> c >> x >> y;
70          if(c == 'A') {
71              lct.link(x, y);
72          } else {
73              lct.makeroot(x);
74              lct.access(y);
75              lct.splay(y);
76              cout << 1ll * (lct.siz[y] - lct.siz[x]) * lct.siz[x] << endl;
77          }
78      }
79  }
```

16

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

```
int phi[N], f[N], pri[N], tot;
void getphi() {
    int k;
    phi[1] = 1;
    for(int i = 2; i < N; i++) {
        if(!f[i]) phi[pri[++tot] = i] = i - 1;
        for(int j = 1; j <= tot && (k = i * pri[j]) < N; j++) {
            f[k] = 1;
            if(i % pri[j]) phi[k] = phi[i] * (pri[j] - 1);
            else {
                phi[k] = phi[i] * pri[j];
                break;
            }
        }
    }
}
```

$O(\sqrt{n})$ 求欧拉函数

```
int getphi(int x) {
    int phi = 1;
    for(int i = 2; i * i <= x; i++) if(x % i == 0) {
        phi *= (i - 1); x /= i;
        while(x % i == 0) {
            phi = phi * i;
            x /= i;
        }
    }
    if(x > 1) phi *= x - 1;
    return phi;
}
```

排列组合

斯特林近似求组合 (≥ 15 时收敛)

精度容易不够，推荐使用 python Demical 类

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

```
double lnfac(int n) {
    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));
}
double C(int n, int m) {
    return exp(lnfac(n) - lnfac(n - m) - lnfac(m));
}
```

Lucas 定理

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

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

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。

```
1  #define long long ll
2  ll gcd(ll a, ll b) {
3      return b == 0 ? a : gcd(b, a % b);
4  }
5  ll lcm(ll a, ll b) {
6      return a / gcd(a, b) * b;
7  }
8  ll exgcd(ll &x, ll &y, ll a, ll b) {
9      if(b == 0) return x = 1, y = 0, a;
10     ll t = exgcd(y, x, b, a % b);
11     y -= a / b * x;
12     return t;
13 }
14 inline ll mul(ll x, ll y, ll mod){
15     return (x * y - (ll)((long double)x / mod * y) * mod + mod) % mod;
16 }
17
18 ll excrt(ll n, ll *a, ll *b) {
19     ll ans = a[1], M = b[1];
```

```
20      for(ll i = 2; i <= n; i++) {
21          ll c = ((a[i] - ans) % b[i] + b[i]) % b[i], x, y;
22          ll t = exgcd(x, y, M, b[i]), pb = b[i] / t;
23          if(c % t != 0) return -1;
24          x = mul(x, c / t, pb);
25          ans = ans + x * M;
26          M = M *pb;
27          ans = (ans % M + M) % M;
28      }
29      return ans;
30  }
```

Miller_rabbin 素数测试

```
1   namespace Isprime{
2       ll mul(ll x, ll y, ll mod){
3           return (x * y - (ll)((long double)x / mod * y) * mod + mod) % mod;
4       }
5       ll Pow(ll a, ll p, ll mod) {
6           ll ans = 1;
7           for( ; p; p >>= 1, a = mul(a, a, mod))
8               if(p & 1)
9                   ans = mul(ans, a, mod);
10          return ans % mod;
11      }
12      int check(ll P){
13          const ll test[11] = {0, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
14          if(P == 1) return false;
15          if(P > 6 && P % 6 != 1 && P % 6 != 5) return false;
16          ll k = 0, t = P - 1;
17          while(!(t & 1)) k++, t >>= 1;
18          for(int i = 1; i <= 10 && test[i] <= P; i++) {
19              if(P == test[i]) return true;
20              ll nxt, a = Pow(test[i], t, P);
21              for(int j = 1; j <= k; j++) {
22                  nxt = mul(a, a, P);
23                  if(nxt == 1 && a != 1 && a != P - 1) return false;
24                  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~$n-1$

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 2π，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);
        for(int m = 1; m < n; m <<= 1) {
            int Wn = Pow(type == 1 ? G : Gi, (P - 1) / (m << 1));
            for(int i = 0; i < n; i += 2 * m) {
                int w = 1;
                for(int j = 0; j < m; j++, w = 1ll * w * Wn % P) {
                    int x = A[i + j], y = 1ll * w * A[i + j + m] % P;
                    A[i + j] = (x + y) % P;
                    A[i + j + m] = (x - y + P) % P;
                }
            }
        }
        if(type == -1) {
            int inv = Pow(n, P - 2);
            for(int i = 0; i < n; i++)
                A[i] = 1ll * A[i] * inv % P;
        }
    }

}
//以下代码加在主函数内
limit = 1;
while(limit <= n + m) limit <<= 1;
Polynomial :: FFT(A, limit, 1);
Polynomial :: FFT(B, limit, 1);
for(int i = 0; i < limit; i++) A[i] = A[i] * B[i];
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)
$IFWT(A) = (\frac{IFWT(A_0) + IFWT(A_1)}{2}, \frac{IFWT(A_0) - IFWT(A_1)}{2})$

```cpp
namespace Polynomial {
    void FWT_or(int *A, int n, int type) {
        for(int m = 1; m < n; m <<= 1) {
            for(int i = 0; i < n; i += 2 * m) {
                for(int j = 0; j < m; j++) {
                    A[i + j + m] = (1ll * A[i + j + m] + A[i + j] * type + mod) % mod;
                }
            }
        }
    }
    void FWT_and(int *A, int n, int type) {
        for(int m = 1; m < n; m <<= 1) {
            for(int i = 0; i < n; i += 2 * m) {
                for(int j = 0; j < m; j++) {
                    A[i + j] = (1ll * A[i + j + m] * type + A[i + j] + mod) % mod;
                }
            }
        }
    }
    void FWT_xor(int *A, int n, int type) {
        int inv_2 = Pow(2, mod - 2);
        for(int m = 1; m < n; m <<= 1) {
            for(int i = 0; i < n; i += 2 * m) {
                for(int j = 0; j < m; j++) {
                    int x = A[i + j], y = A[i + j + m];
                    A[i + j] = (1ll * x + y) * (type == 1 ? 1 : inv_2) % mod;
                    A[i + j + m] = (1ll * x - y + mod) * (type == 1 ? 1 : inv_2) % mod;
                }
            }
        }
    }
}
```

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。

```cpp
namespace ssc{
    int f[21][1 << 21], g[21][1 << 21], ans[21][1 << 21];
    void subset_convolution(int *A, int *B, int *C, int n, int lim) {
    //  memset(f, 0, sizeof(f));
    //  memset(g, 0, sizeof(g));
        for(int i = 0; i < lim; i++) f[__builtin_popcount(i)][i] = A[i];
        for(int i = 0; i < lim; i++) g[__builtin_popcount(i)][i] = B[i];
        for(int i = 0; i <= n; i++) FWT_or(f[i], lim, 1), FWT_or(g[i], lim, 1);
        for(int i = 0; i <= n; i++)
            for(int j = 0; j <= i; j++)
                for(int k = 0; k < lim; k++)
                    ans[i][k] = (ans[i][k] + 1ll * f[j][k] * g[i - j][k] % mod) % mod;
        for(int i = 0; i <= n; i++) FWT_or(ans[i], lim, -1);
        for(int i = 0; i < lim; i++) C[i] = ans[__builtin_popcount(i)][i];
    }
}
```

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 $g^{-1} \cdot h \in H$。

BurnSide 引理

定义 AB 同构为在群 G 中存在一个运算 f 使得 $f(A) = B$，则本质不同的元素个数为

$$\frac{\sum_{f \in G} c(f)}{|G|}$$

$c(f)$ 为 $\sum[f(A) == A]$，也就是 f 的不动点数量。

Polya 定理

在 BurnSide 的基础上，染色数为 m，则本质不同的染色方案数为

$$\frac{\sum_{f \in G} m^{cnt_f}}{|G|}$$

cnt_f 为置换 f 的循环节个数
> 在 Burnside 的计算不动点过程中，如果两个状态置换后相同，那么同一个子循环置换中颜色一定相同，不同子循环置换中颜色选取独立。

项链计数问题:
一个 n 元环，m 染色，旋转同构，方案数为:

$$\frac{\sum_{i=1}^{n} m^{gcd(n,i)}}{n}$$

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

28

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  }
```

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);
```

kosaraju 算法

先正着跑一边 dfs，在出栈的时候把点加入栈。
再倒着跑，此时栈顶是 DAG 的末端，每次跑到的点在同一个 scc 里面。

```
1   pair<int, vector<int> > kosaraju(vector<vector<int> > ver) {
2       int n = ver.size() - 1;
3       vector<int> scc(n + 1);
4       int cnt = 0;
5       vector<vector<int> > rver(n + 1);
6       for(int i = 1; i <= n; i++) {
7           for(auto y : ver[i]) {
8               rver[y].push_back(i);
9           }
10      }
11      vector<int> vis(n + 1);
12      stack<int> q;
13      auto dfs1 = [&](auto &&me, int x) -> void {
14          vis[x] = 1;
15          for(auto y : ver[x]) if(!vis[y]) {
16              me(me, y);
17          }
18          q.push(x);
19      };
20      auto dfs2 = [&](auto &&me, int x) -> void {
21          vis[x] = 0;
22          scc[x] = cnt;
23          for(auto y : rver[x]) if(vis[y]) {
24              me(me, y);
25          }
26      };
27      for(int i = 1; i <= n; i++) if(!vis[i]) dfs1(dfs1, i);
28      while(q.size()) {
29          int x = q.top();
30          q.pop();
31          if(vis[x]) {
32              cnt++;
33              dfs2(dfs2, x);
34          }
35      }
36      return {cnt, scc};
37  }
38  vector<vector<int> > reduction(vector<vector<int> > ver, vector<int> scc, int scnt) {
39      int n = ver.size() - 1;
40      vector<vector<int> > sver(scnt + 1);
41      map<pair<int, int>, int> M;
42      for(auto x = 1; x <= n; x++) {
43          for(auto y : ver[x]) {
44              if(scc[x] != scc[y] && !M.count({scc[x], scc[y]})) {
45                  M[{scc[x], scc[y]}] = 1;
46                  sver[scc[x]].push_back(scc[y]);
47              }
48          }
49      }
50      return sver;
```

```
51    }
```

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。

```c
for(int i = 1; i <= m; i++) {
    int x = read() + 1, y = read() + 1;
    int w = read();
    char c[10];
    scanf("%s", c + 1);
    if(c[1] == 'A') {
        if(w) {
            add(2 * x - 0, 2 * x - 1);
            add(2 * y - 0, 2 * y - 1);
        } else {
            add(2 * x - 1, 2 * y - 0);
            add(2 * y - 1, 2 * x - 0);
        }
    }
    if(c[1] == 'O') {
        if(w) {
            add(2 * x - 0, 2 * y - 1);
            add(2 * y - 0, 2 * x - 1);
        } else {
            add(2 * x - 1, 2 * x - 0);
            add(2 * y - 1, 2 * y - 0);
        }
    }
    if(c[1] == 'X') {
        if(w) {
            add(2 * x - 0, 2 * y - 1);
            add(2 * x - 1, 2 * y - 0);
            add(2 * y - 0, 2 * x - 1);
            add(2 * y - 1, 2 * x - 0);
        } else {
            add(2 * x - 0, 2 * y - 0);
            add(2 * x - 1, 2 * y - 1);
            add(2 * y - 0, 2 * x - 0);
            add(2 * y - 1, 2 * x - 1);
        }
    }
}
for(int i = 1; i <= 2 * n; i++)
    if(!dfn[i])
        tarjan(i);
for(int i = 1; i <= n; i++) {
    if(scc[2 * i - 0] == scc[2 * i - 1]) {
        printf("NO\n");
        return 0;
    }
}
printf("YES\n");
//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

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

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

```
const double Pi = acos(-1.0);
const double eps = 1e-11;
// 三态函数
int sgn(double x) {
    if(fabs(x) < eps) return 0;
    else return x < 0 ? -1 : 1;
}
struct line;
struct Point;
struct Point {
    double x, y;
    Point() : x(0), y(0) {}
    Point(double x, double y) : x(x), y(y) {}
};
struct line{
    Point s, t;
    line() {}
    line(const Point &s, const Point &t) : s(s), t(t) {}
};

struct circle{
    Point c;
    double r;
    circle() : c(Point(0,0)), r(0) {}
    circle(const Point &c, double r) : c(c), r(r) {}
    Point point(double a) {
        return Point(c.x + cos(a)*r, c.y + sin(a)*r);
    }
};
typedef Point Vector;
Point operator+(const Point &a, const Point &b) { return Point(a.x + b.x, a.y + b.y); }
Point operator-(const Point &a, const Point &b) { return Point(a.x - b.x, a.y - b.y); }
Point operator*(const Point &a, const double &c) { return Point(c * a.x, c * a.y); }
Point operator/(const Point &a, const double &c) { return Point(a.x / c, a.y / c); }
inline bool operator < (const Point &a, const Point &b) {
    return sgn(a.x - b.x) < 0 || (sgn(a.x - b.x) == 0 && sgn(a.y - b.y) < 0);
}
bool operator == (const Point& a, const Point& b) { return !sgn(a.x - b.x) && !sgn(a.y - b.y); }
// 点积
double dot(const Vector& a, const Vector& b) { return a.x * b.x + a.y * b.y; }
// 叉积
double det(const Vector& a, const Vector& b) { return a.x * b.y - a.y * b.x; }
double cross(const Point& s, const Point& t, const Point& o = Point()) { return det(s - o, t - o); }
```

基本操作

```
// 点到原点距离
double abs(const Point &a){ return sqrt(a.x * a.x + a.y * a.y); }
// 点旋转 theta 角度
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)); }
// 逆时针旋转 90 度
Point rotCCW90(const Point &a) { return Point(-a.y, a.x); }
// 顺时针旋转 90 度
Point rotCW90(const Point &a) { return Point(a.y, -a.x); }
// 点的幅角`
double arg(const Point &a){
    double t = atan2(a.y, a.x);
    return t < 0 ? t + 2 * Pi:t;
}
//极角排序
// 1 浮点数坐标排序
// 顺序（象限）: 3 -> 4 -> 1 -> 2
int cmp(const node &a, const node &b) {
    if(atan2(a.y, a.x) != atan2(b.y, b.x)) {
```

```
19        return atan2(a.y, a.x) < atan2(b.y, b.x);
20    }
21    return a.x < b.x;
22 }
23 // 2 整数坐标排序
24 // 顺序（象限）: 1 -> 2 -> 3 -> 4
25 int up(const node &a) {
26    return a.y > 0 || (a.y == 0 && a.x >= 0);
27 }
28 int cmp(const node &a, const node &b) {
29    if(up(a) != up(b)) return up(a) > up(b);
30    return det(a, b) > 0;
31 }
```

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(det(l.s - p, l.t - p)) / (l.s - l.t).len();
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(a.t - a.s, b.s - a.s), s2 = det(a.t - a.s, 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 >= 1 表示相交
7  int s_l_cross(const Line &seg, const Line &line) {
8     int d1 = sgn(det(line.s - seg.s, line.t - seg.s));
9     int d2 = sgn(det(line.s - seg.t, 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   >= 1 表示相交
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(det(a.t - a.s, b.s - a.s)), d2 = sgn(det(a.t - a.s, b.t - a.s));
18    int d3 = sgn(det(b.t - b.s, a.s - b.s)), d4 = sgn(det(b.t - b.s, 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   // 点是否在多边形中 0 = 在外部 1 = 在内部 -1 = 在边界上
11   int p_in_poly(Point p, const vector<Point> &s){
12       int cnt = 0;
13       for(int i = 0; i < s.size(); i++) {
14           Point a = s[i], b = s[nxt(i)];
15           if (p_on_seg(p, line(a, b))) return -1;
16           //p 在多边形边上
17           if (sgn(a.y - b.y) <= 0) swap(a, b);
18           if (sgn(p.y - a.y) > 0) continue;
19           if (sgn(p.y - b.y) <= 0) continue;
20           //一条边包含它较高的点, 不包含较低的点
21           cnt += sgn(det(b - p, a - p)) > 0;
22           //如果 p 在这条线段左边
23       }
24       return bool(cnt & 1);
25   }
```

凸包

andrew 算法,

```
1    // 构建凸包 点不可以重复 < 0 边上可以有点, <= 0 则不能
2    // 会改变输入点的顺序
3    vector<Point> Convex_hull(vector<Point> &s) {
4        sort(s.begin(), s.end());
5        vector<Point> ret(s.size() * 2);
6        int sz = 0;
7        for(int i = 0; i < s.size(); i++) {
8            while(sz > 1 && sgn(det(ret[sz - 1] - ret[sz - 2], s[i] - ret[sz - 2])) <= 0) sz--;
9            ret[sz++] = s[i];
10       }
11       int k = sz;
12       for(int i = s.size() - 2; i >= 0; i--) {
13           while(sz > k && sgn(det(ret[sz - 1] - ret[sz - 2], s[i] - ret[sz - 2])) <= 0) sz--;
14           ret[sz++] = s[i];
15       }
16       ret.resize(sz - (s.size() > 1));
17       return ret;
18   }
```

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    int cmp(const Point &a, const Point &b) {
2        return sgn(a.x - b.x) < 0 || (sgn(a.x - b.x) == 0 && sgn(a.y - b.y) < 0);
3    }
4    double rotatingCalipers(const vector<Point> &s) {
5        if(s.size() == 2) return abs(s[1] - s[0]);
6        int i = 0, j = 0;
7        for(int k = 0; k < s.size(); k++) {
8            if( cmp(s[i], s[k])) i = k;
9            if(!cmp(s[j], s[k])) j = k;
10       }
11       double ans = 0;
12       int si = i, sj = j;
```

```
13        do{
14            ans = max(ans, abs(s[i] - s[j]));
15            if(sgn(det(s[(i + 1) % s.size()] - s[i], s[(j + 1) % s.size()] - s[j])) < 0)
16                i = (i + 1) % s.size();
17            else j = (j + 1) % s.size();
18        } while(i != si || j != sj);
19        return ans;
20    }
```

半平面交

（多边形面积交）

```
1    #include <bits/stdc++.h>
2    using namespace std;
3    const int N = 2e6 + 1009;
4    const double eps = 1e-9;
5    int sgn(double x) {
6        if(-eps < x && x < eps) return 0;
7        else return x > 0 ? 1 : -1;
8    }
9    struct Point {
10        double x, y;
11        int quad() const {return sgn(y) == 1 || (sgn(y) == 0 && sgn(x) >= 0);}
12        Point(double a, double b) : x(a), y(b) {}
13    };
14    Point operator-(const Point &a, const Point &b) {return {a.x - b.x, a.y - b.y};}
15    Point operator+(const Point &a, const Point &b) {return {a.x + b.x, a.y + b.y};}
16    Point operator*(const Point &a, double b) {return {a.x * b, a.y * b};}
17    Point operator*(double b, const Point &a) {return {a.x * b, a.y * b};}
18    Point operator/(double b, const Point &a) {return {a.x / b, a.y / b};}
19    Point operator/(const Point &a, double b) {return {a.x / b, a.y / b};}
20    double dot(const Point &a, const Point &b) {
21        return a.x * b.x + a.y * b.y;
22    }
23    double det(const Point &a, const Point &b) {
24        return a.x * b.y - a.y * b.x;
25    }
26    struct Line {
27        Point s, t;
28        bool include(const Point &p) const { return sgn(det(t - s, p - s)) > 0;}
29        Line(Point a, Point b) : s(a), t(b) {}
30    };
31    Point l_intersection(const Line& a, const Line& b) {
32        double s1 = det(a.t - a.s, b.s - a.s), s2 = det(a.t - a.s, b.t - a.s);
33        return (b.s * s2 - b.t * s1) / (s2 - s1);
34    }
35    bool operator<(const Point &a, const Point &b) {
36        if(a.quad() != b.quad()) {
37            return a.quad() < b.quad();
38        }
39        return sgn(det(a, b)) > 0;
40    }
41    bool parallel(const Line &a, const Line &b) {
42        return !sgn(det(a.t - a.s, b.t - b.s));
43    }
44    bool sameDir(const Line &a, const Line &b) {
45        return parallel(a, b) && (sgn(dot(a.t - a.s, b.t - b.s)) == 1);
46    }
47    bool operator<(const Line &a, const Line &b) {
48        if(sameDir(a, b)) {
49            return b.include(a.s);
50        } else {
51            return (a.t - a.s) < (b.t - b.s);
52        }
53    }
54    bool check(const Line &u, const Line &v, const Line &w) {
55        return w.include(l_intersection(u, v));
56    }
57    vector<Point> half_intersection(vector<Line> &l) {
58        sort(l.begin(), l.end());
```

```
59      deque<Line> q;
60      for(int i = 0; i < l.size(); i++) {
61          if(i && sameDir(l[i], l[i - 1])) {
62              continue;
63          }
64          while(q.size() > 1 && !check(q[q.size() - 2], q[q.size() - 1], l[i])) q.pop_back();
65          while(q.size() > 1 && !check(q[1], q[0], l[i])) q.pop_front();
66          q.push_back(l[i]);
67      }
68      while(q.size() > 2 && !check(q[q.size() - 2], q[q.size() - 1], q[0])) q.pop_back();
69      while(q.size() > 2 && !check(q[1], q[0], q[q.size() - 1])) q.pop_front();
70      vector<Point> ret;
71      for (int i = 0; i < q.size(); i++) ret.push_back(l_intersection(q[i], q[(i + 1) % q.size()]));
72      return ret;
73  }
74  int n, m;
75  void work() {
76      scanf("%d", &n);
77      vector<Line> v;
78      for(int i = 1; i <= n; i++) {
79          scanf("%d", &m);
80          vector<Point> p;
81          for(int j = 1; j <= m; j++) {
82              double x, y;
83              scanf("%lf%lf", &x, &y);
84              p.push_back({x, y});
85          }
86          for(int j = 0; j < m; j++) {
87              v.push_back({p[j], p[(j + 1) % m]});
88          }
89      }
90      vector<Point> p = half_intersection(v);
91      double area = 0;
92      for(int i = 0; i < p.size(); i++) {
93          area += det(p[i], p[(i + 1) % p.size()]);
94      }
95      printf("%.3f\n", fabs(area) / 2);
96
97  }
```

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
13  vector<Point> p_c_tangent(const Point &p, const circle &o) {
14      vector<Point> ret;
15      double d = abs(p - o.c), x = dot(p - o.c, p - o.c) - o.r * o.r;
16      if(sgn(x) < 0) ;
17      else if(sgn(x) == 0) ret.push_back(p);
18      else {
19          Vector base = p + (o.c - p) * x / dot(p - o.c, p - o.c);
20          Vector e = rotCW90(o.c - p) / d;
21          ret.push_back(base + e * sqrt(x) * o.r / d);
22          ret.push_back(base - e * sqrt(x) * o.r / d);
23      }
24      return ret;
25  }
26  // 圆与圆的交点
27  vector<Point> c_c_intersection(const circle &a, const circle &b) {
```

```
28        vector<Point> ret;
29        double d = abs(b.c - a.c);
30        if(sgn(d) == 0 || sgn(d - a.r - b.r) > 0 || sgn(d + min(a.r, b.r) - max(a.r, b.r)) < 0)
31            return ret;
32        double x = (a.r * a.r + dot(b.c - a.c, b.c - a.c) - b.r * b.r) / (2 * d);
33        double y = sqrt(a.r * a.r - x * x);
34        Point v = (b.c - a.c) / d;
35        ret.push_back(a.c + v * x + rotCW90(v) * y);
36        if(sgn(y) > 0) ret.push_back(a.c + v * x - rotCW90(v) * y);
37        return ret;
38    }
```

在 res 中存放的线上的两点分别是在 c1,c2 上的切点。

```
1     int tangent(const circle &C1, const circle &C2, vector<line> &res){
2         double d = abs(C1.c - C2.c);
3         if(d < eps) return 0;
4
5         int c=0;
6         // 内公切线
7         if(C1.r + C2.r < d - eps){
8             double t = acos((C1.r + C2.r) / d);
9             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)));
10            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)));
11            c += 2;
12        } else if(C1.r + C2.r < d + eps){
13            Point p = C1.c + C1.r / d * (C2.c - C1.c);
14            res.push_back(line(p, p + rot(C2.c - C1.c, Pi / 2)));
15            c++;
16        }
17
18        // 外公切线
19        if(abs(C1.r - C2.r) < d - eps){
20            double t1 = acos((C1.r - C2.r) / d), t2 = Pi - t1;
21            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)));
22            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)));
23            c+=2;
24        } else if(abs(C1.r - C2.r) < d + eps){
25            Point p = C1.c + C1.r / d * (C2.c - C1.c);
26            res.push_back(line(p, p + rot(C2.c - C1.c, Pi / 2)));
27            c++;
28        }
29
30        return c;
31    }
```

最小圆覆盖

```
1     Line norm(const Line &a) {
2         Point mid = (a.t + a.s) / 2;
3         Point dir = a.t - a.s;
4         dir = {dir.y, -dir.x};
5         return Line(mid, mid + dir);
6     }
7     double abs(const Point &a, const Point &b) {
8         return sqrt(dot(a - b, a - b));
9     }
10    struct Circle {
11        Point o;
12        double r;
13        Circle(const Point &o, const double &r) : o(o), r(r) {}
14        Circle(const Point &a, const Point &b, const Point &c) : o(0, 0) {
15            Line ab = norm(Line(b, a));
16            Line bc = norm(Line(b, c));
17            o = l_intersection(ab, bc);
18            r = abs(o, a);
19        }
20    };
21
22    int n, m, a[N];
23    vector<Point> v;
```

```
24   void work() {
25       cin >> n;
26       srand(12391278);
27       for(int i = 1; i <= n; i++) {
28           double x, y;
29           cin >> x >> y;
30           v.push_back({x, y});
31       }
32       random_shuffle(v.begin(), v.end());
33       Circle ans = {v[0], 0};
34       for(int i = 1; i < n; i++) {
35           if(abs(ans.o, v[i]) > ans.r + eps) {
36               ans = {v[i], 0};
37               for(int j = 0; j < i; j++) {
38                   if(abs(ans.o, v[j]) > ans.r + eps) {
39                       ans = {(v[i] + v[j]) / 2, abs(v[i], v[j]) / 2};
40                       for(int k = 0; k < j; k++) {
41                           if(abs(ans.o, v[k]) > ans.r + eps) {
42                               ans = Circle(v[i], v[j], v[k]);
43                           }
44                       }
45                   }
46               }
47           }
48       }
49       cout << fixed << setprecision(10) << ans.o.x << " " << ans.o.y << " " << ans.r << endl;
50   }
```

三维几何

三维凸包（最小面积）

```
1    const double eps = 1e-10;
2    double rand_eps() {
3        return ((double)rand() / RAND_MAX - 0.5) * eps;
4    }
5    struct Point {
6        double x, y, z;
7        Point(double x, double y, double z) : x(x), y(y), z(z) {}
8        double len() {
9            return sqrt(x * x + y * y + z * z);
10       }
11       void shake() {
12           x += rand_eps();
13           y += rand_eps();
14           z += rand_eps();
15       }
16   };
17   Point operator+(const Point &a, const Point &b) { return {a.x + b.x, a.y + b.y, a.z + b.z}; }
18   Point operator-(const Point &a, const Point &b) { return {a.x - b.x, a.y - b.y, a.z - b.z}; }
19   Point operator*(const Point &a, const double &b) { return {a.x * b, a.y * b, a.z * b}; }
20   Point operator/(const Point &a, const double &b) { return {a.x / b, a.y / b, a.z / b}; }
21   double dot(const Point &a, const Point &b) {
22       return a.x * b.x + a.y * b.y + a.z * b.z;
23   }
24   Point det(const Point &a, const Point &b) {
25       return {a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z, a.x * b.y - a.y * b.x};
26   }
27   vector<Point> p;
28   struct Plane {
29       int v[3];
30       Plane(const int &a, const int &b, const int &c) : v{a, b, c} {}
31       Point norm() {
32           return det(p[v[0]] - p[v[1]], p[v[2]] - p[v[1]]);
33       }
34       int above(const Point &q) {
35           return dot((q - p[v[0]]), norm()) >= 0;
36       }
37       double area() {
38           return norm().len() / 2;
39       }
```

```
40    };
41    int n;
42    vector<Plane> convex_hull_3d() {
43        for(int i = 0; i < p.size(); i++) p[i].shake();
44        vector<Plane> plane, tmp;
45        vector<vector<int> > vis(n);
46        for(int i = 0; i < n; i++)
47            vis[i].resize(n);
48        plane.emplace_back(0, 1, 2);
49        plane.emplace_back(0, 2, 1);
50        for(int i = 3; i < n; i++) {
51            tmp = plane;
52            plane.clear();
53            for(auto t : tmp) {
54                bool f = t.above(p[i]);
55                if(!f) plane.push_back(t);
56                for(int k = 0; k < 3; k++) {
57                    vis[t.v[k]][t.v[(k + 1) % 3]] = f;
58                }
59            }
60            for(auto t : tmp) {
61                for(int k = 0; k < 3; k++) {
62                    int a = t.v[k], b = t.v[(k + 1) % 3];
63                    if(vis[a][b] != vis[b][a] && vis[a][b]) {
64                        plane.push_back(Plane{a, b, i});
65                    }
66                }
67            }
68        }
69        return plane;
70
71    }
72    void work() {
73        srand(1230213);
74        cin >> n;
75        for(int i = 0; i < n; i++) {
76            double x, y, z;
77            cin >> x >> y >> z;
78            p.push_back({x, y, z});
79        }
80        vector<Plane> convex = convex_hull_3d();
81        double area = 0;
82        for(auto x : convex) {
83            area += x.area();
84        }
85        cout << fixed << setprecision(6) << area << endl;
86    }
```

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

双哈希

```
1   struct Hash {
2       int a, b;
3       Hash() : a(0), b(0) {}
4       Hash(int x) : a(x), b(x) {}
5       Hash operator+(Hash x) {
6           Hash tmp;
7           tmp.a = (a + x.a >= p1) ? a + x.a - p1 : a + x.a;
8           tmp.b = (b + x.b >= p2) ? b + x.b - p2 : b + x.b;
9           return tmp;
10      }
11      Hash operator-(Hash x) {
12          Hash tmp;
13          tmp.a = (a - x.a < 0) ? a - x.a + p1 : a - x.a;
```

```
14              tmp.b = (b - x.b < 0) ? b - x.b + p2 : b - x.b;
15              return tmp;
16          }
17          Hash operator*(Hash x) {
18              Hash tmp;
19              tmp.a = (long long) a * x.a % p1;
20              tmp.b = (long long) b * x.b % p2;
21              return tmp;
22          }
23          Hash operator*(int x) {
24              Hash tmp;
25              tmp.a = (long long) a * x % p1;
26              tmp.b = (long long) b * x % p2;
27              return tmp;
28          }
29          bool operator==(Hash x) {
30              return a == x.a && b == x.b;
31          }
32      };
```

字串哈希

```
1   namespace String {
2       const int x = 133;
3       const int p1 = 1e9 + 7, p2 = 1e9 + 9;
4       ull xp1[N], xp2[N], xp[N];
5       void init_xp() {
6           xp1[0] = xp2[0] = xp[0] = 1;
7           for(int i = 1; i < N; i++) {
8               xp1[i] = xp1[i - 1] * x % p1;
9               xp2[i] = xp2[i - 1] * x % p2;
10              xp[i] = xp[i - 1] * x;
11          }
12      }
13      struct HashString {
14          char s[N];
15          int length, subsize;
16          bool sorted;
17          ull h[N], hl[N];
18          ull init(const char *t) {
19              if(xp[0] != 1) init_xp();
20              length = strlen(t);
21              strcpy(s, t);
22              ull res1 = 0, res2 = 0;
23              h[length] = 0;
24              for(int j = length - 1; j >= 0; j--) {
25          #ifdef ENABLE_DOUBLE_HASH
26                  res1 = (res1 * x + s[j]) % p1;
27                  res2 = (res2 * x + s[j]) % p2;
28                  h[j] = (res1 << 32) | res2;
29          #else
30                  res1 = res1 * x + s[j];
31                  h[j] = res1;
32          #endif
33              }
34              return h[0];
35          }
36          //获取子串哈希，左闭右开
37          ull get_substring_hash(int left, int right)  {
38              int len = right - left;
39          #ifdef ENABLE_DOUBLE_HASH
40              unsigned int mask32 = ~(0u);
41              ull left1 = h[left] >> 32, right1 = h[right] >> 32;
42              ull left2 = h[left] & mask32, right2 = h[right] & mask32;
43              return (((left1 - right1 * xp1[len] % p1 + p1) % p1) << 32) |
44                      (((left2 - right2 * xp2[len] % p2 + p2) % p2));
45          #else
46              return h[left] - h[right] * xp[len];
47          #endif
48          }
49          void get_all_subs_hash(int sublen) {
```

```
50          subsize = length - sublen + 1;
51          for (int i = 0; i < subsize; ++i)
52              hl[i] = get_substring_hash(i, i + sublen);
53          sorted = 0;
54      }
55
56      void sort_substring_hash() {
57          sort(hl, hl + subsize);
58          sorted = 1;
59      }
60
61      bool match(ull key) const {
62          if (!sorted) assert (0);
63          if (!subsize) return false;
64          return binary_search(hl, hl + subsize, key);
65      }
66  };
67 }
```

后缀数组

sa[i] 定义为字典序排名第 i 的后缀的起始位置。

rk[i] 定义为以 i 起始的后缀的排名。

height[i] 定义为 sa[i] 和 sa[i - 1] 的最长公共前缀的长度。

定义 h[i] 为排名第 i 的字符串和排名第 i - 1 的字符串的最长公共前缀长度。

有 height[i]>=height[i - 1] - 1。

$$lcp(sa[i], sa[j]) = min(lcp(sa[i], sa[k]), lcp(sa[k], sa[j])) \forall sa[i] \le sa[k] \le sa[j]$$

$$lcp(sa[i], sa[j]) = min\{height[i+1, \cdots, j]\}$$

```
1  int sa[N], id[N], rk[N], oldrk[N], cnt[N], height[N];
2  void get_sa(char *c) {
3      n = strlen(c + 1);
4      m = 300;
5      for(int i = 1; i <= n; i++) cnt[rk[i] = c[i]]++;
6      for(int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
7      for(int i = n; i >= 1; i--) sa[cnt[rk[i]]--] = i;
8      for(int w = 1; w < n; w <<= 1) {
9          int tt = 0;
10         for(int i = 1; i <= w; i++) id[++tt] = i + n - w;
11         for(int i = 1; i <= n; i++) if(sa[i] > w) {
12                 id[++tt] = sa[i] - w;
13             }
14         memset(cnt, 0, sizeof(int) * (m + 1));
15         for(int i = 1; i <= n; i++) cnt[rk[i]]++;
16         for(int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
17         for(int i = n; i >= 1; i--) sa[cnt[rk[id[i]]]--] = id[i];
18         memcpy(oldrk, rk, sizeof(int) * (n + 1));
19         int p = 1;
20         rk[sa[1]] = 1;
21         for(int i = 2; i <= n; i++) {
22             rk[sa[i]] = (oldrk[sa[i]] == oldrk[sa[i - 1]] && oldrk[sa[i] + w] == oldrk[sa[i - 1] + w]) ? p : ++p;
23         }
24         if(p == n) break;
25         m = p;
26     }
27 }
28 void get_height(char *c) {
29     for(int i = 1, k = 0; i <= n; i++) {
30         if(rk[i] == 1) continue;
31         if(k) k--;
32         int j = sa[rk[i] - 1];
33         while(i + k <= n && j + k <= n && c[i + k] == c[j + k]) k++;
34         height[rk[i]] = k;
35     }
36 }
```

SA-IS

会开两倍空间，如果传入的是整数数组，字符集请从 1 开始，不能出现 0。

```
1    template<size_t size>
2    struct suffix_array {
3        bool type[size << 1];
4        int ht[size], rk[size], sa[size], bk[size], bk1[size];
5
6        inline bool isLMS(int i, bool *type) {
7            return (i > 0 && type[i] && !type[i - 1]);
8        }
9
10       template<class T>
11       inline void induced_sort(T s, int *sa, int len, int bs, int sigma,
12                                bool *type, int *bk, int *cb, int *p) {
13           memset(bk, 0, sizeof(int) * sigma), memset(sa, -1, sizeof(int) * len);
14           for (int i = 0; i < len; ++i) bk[s[i]]++;
15           cb[0] = bk[0];
16           for (int i = 1; i < sigma; ++i) cb[i] = cb[i - 1] + bk[i];
17           for (int i = bs - 1; i >= 0; --i) sa[--cb[s[p[i]]]] = p[i];
18           for (int i = 1; i < sigma; ++i) cb[i] = cb[i - 1] + bk[i - 1];
19           for (int i = 0; i < len; ++i)
20               if (sa[i] > 0 && !type[sa[i] - 1])
21                   sa[cb[s[sa[i] - 1]]++] = sa[i] - 1;
22           cb[0] = bk[0];
23           for (int i = 1; i < sigma; ++i) cb[i] = cb[i - 1] + bk[i];
24           for (int i = len - 1; i >= 0; --i)
25               if (sa[i] > 0 && type[sa[i] - 1])
26                   sa[--cb[s[sa[i] - 1]]] = sa[i] - 1;
27       }
28
29       template<class T>
30       inline void sais(T s, int *sa, int len, bool *type, int *bk,
31                        int *bk1, int sigma) {
32           int p = -1, j, cnt = 0, bs = 0, *cb = bk + sigma, x;
33           type[len - 1] = true;
34           for (int i = len - 2; i >= 0; --i)
35               type[i] = (s[i] < s[i + 1] || (s[i] == s[i + 1] && type[i + 1]));
36           for (int i = 1; i < len; ++i)
37               if (type[i] && !type[i - 1]) bk1[bs++] = i;
38           induced_sort(s, sa, len, bs, sigma, type, bk, cb, bk1);
39           for (int i = bs = 0; i < len; ++i)
40               if (isLMS(sa[i], type)) sa[bs++] = sa[i];
41           for (int i = bs; i < len; ++i) sa[i] = -1;
42           for (int i = 0; i < bs; ++i) {
43               x = sa[i];
44               for (int j = 0; j < len; ++j) {
45                   if (p == -1 || s[x + j] != s[p + j] || type[x + j]
46                                                    != type[p + j]) { cnt++, p = x; break; }
47                   else if (j > 0 && (isLMS(p + j, type)
48                                    || isLMS(x + j, type))) break;
49               }
50               x >>= 1, sa[bs + x] = cnt - 1;
51           }
52           for (int i = j = len - 1; i >= bs; --i)
53               if (sa[i] >= 0) sa[j--] = sa[i];
54           int *s1 = sa + len - bs, *bk2 = bk1 + bs;
55           if (cnt < bs) sais(s1, sa, bs, type + len, bk, bk2, cnt);
56           else for (int i = 0; i < bs; ++i) sa[s1[i]] = i;
57           for (int i = 0; i < bs; ++i) bk2[i] = bk1[sa[i]];
58           induced_sort(s, sa, len, bs, sigma, type, bk, cb, bk2);
59       }
60
61       template<class T>
62       inline void get_height(T *s, int len, int *sa) {
63           for (int i = 0, k = 0; i < len; ++i) {
64               if (rk[i] == 0) k = 0;
65               else {
66                   if (k > 0) k--;
67                   int j = sa[rk[i] - 1];
68                   while (i + k < len && j + k < len && s[i + k] == s[j + k]) k++;
69               }
70               ht[rk[i]] = k;
71           }
```

```
72          }
73
74          template<class T>
75          inline void init(T s, int len, int sigma = 124) {
76              // len 是字符串长度， sigma 是字符集大小
77              // 计算完 sa[1...len]，存放 0...len-1，表示排名第 i 的后缀的开始位置
78              // rk[0...len-1] 表示从 i 开始的后缀的排名
79              // h[2...len] 表示排名为 i 的后缀，与排名为 i-1 的后缀的 lcp 是多少
80              len += 1;
81              sais(s, sa, len, type, bk, bk1, sigma);
82              for (int i = 1; i < len; ++i) rk[sa[i]] = i;
83              get_height(s, len, sa);
84          }
85      };
```

SAM

字符集要放缩到 0-sigma

```
1   template<size_t size, size_t sigma>
2   struct SAM{
3       int ch[size << 1][sigma], fail[size << 1], siz[size << 1], len[size << 1], endpos[size << 1];
4       int t[size << 1], A[size << 1];
5       //int jump[size << 1][19], endsta[size << 1];
6       long long cnt[size << 1];
7       int num = 1, last = 1;
8       void insert(int c, int pos) {
9           int now = ++ num, node = last;
10          len[now] = len[last] + 1;
11          siz[now] = 1; endpos[now] = pos;
12          //endsta[pos] = now;
13          while(node && !ch[node][c]) ch[node][c] = now, node = fail[node];
14          if(!node) fail[now] = 1;
15          else {
16              int cur = ch[node][c];
17              if(len[cur] == len[node] + 1) fail[now] = cur;
18              else {
19                  int clone = ++ num;
20                  for(int i = 0; i < sigma; i ++) ch[clone][i] = ch[cur][i];
21                  len[clone] = len[node] + 1;
22                  fail[clone] = fail[cur];
23                  while(node && ch[node][c] == cur)
24                      ch[node][c] = clone, node = fail[node];
25                  fail[now] = fail[cur] = clone;
26              }
27          }
28          last = now;
29      }
30      void prework() {
31          for(int i = 1; i <= num; i ++) t[len[i]] ++;
32          for(int i = 1; i <= num; i ++) t[i] += t[i - 1];
33          for(int i = 1; i <= num; i ++) A[t[len[i]] --] = i;
34          for(int i = num; i >= 1; i --)
35              siz[fail[A[i]]] += siz[A[i]],
36                  endpos[fail[A[i]]] = max(endpos[fail[A[i]]], endpos[A[i]]);
37          for(int i = 1; i <= num; i ++)
38              cnt[A[i]] = cnt[fail[A[i]]] + 1ll * (len[A[i]] - len[fail[A[i]]]) * siz[A[i]];
39          // for(int i = 1; i <= num; i++) jump[i][0] = fail[i];
40          // jump[1][0] = 1;
41          // for(int j = 1; j < 19; j++) {
42          //     for(int i = 1; i <= num; i++) {
43          //         jump[i][j] = jump[jump[i][j - 1]][j - 1];
44          //     }
45          // }
46      }
47      void clear() {
48          memset(len, 0, sizeof(int) * (num + 1));
49          memset(fail, 0, sizeof(int) * (num + 1));
50          memset(cnt, 0, sizeof(int) * (num + 1));
51          memset(endpos, 0, sizeof(int) * (num + 1));
52          memset(siz, 0, sizeof(int) * (num + 1));
```

```
53        memset(t, 0, sizeof(int) * (num + 1));
54        memset(A, 0, sizeof(int) * (num + 1));
55        memset(ch, 0, sizeof(int) * (num + 1) * sigma);
56        last = num = 1;
57    }
58    void build(const string &s) {
59        clear();
60        for(int i = 0; i < s.size(); i++) insert(s[i] - 'a', i);
61        prework();
62    }
63    // int fidNode(int l, int r) {
64    //    // 定位子串 [l, r] 对应的节点
65    //    int now = endsta[r];
66    //    for(int i = 18; i >= 0; i--) {
67    //        if(len[jump[now][i]] >= (r - l + 1)) {
68    //            now = jump[now][i];
69    //        }
70    //    }
71    //    return now;
72    // }
73 };
74 SAM<N, 26> sam;
```

后缀树

后缀树是将字符串的所有后缀插入到 Trie 树里面形成的数据结构，它等于字符串反串 SAM 的 parent 树。

```
1 // 放到 SAM 的 prework 中
2 for(int i = 2; i <= num; i++) {
3     ver[fail[i]].push_back({s[endpos[i] - len[fail[i]]] - 'a', i});
4 }
5 // 字典序
6 for(int i = 1; i <= num; i++) std::sort(ver[i].begin(), ver[i].end());
```

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

```cpp
namespace KMP {
    void get_next(char *t, int m, int *nxt) {
        int j = nxt[0] = 0;
        for(int i = 1; i < m; i++) {
            while(j && t[i] != t[j]) j = nxt[j - 1];
            nxt[i] = j += (t[i] == t[j]);
        }
    }
    vector<int> find(char *t, int m, int *nxt, char *s, int n) {
        vector<int> ans;
        int j = 0;
        for(int i = 0; i < n; i++) {
            while(j && s[i] != t[j]) j = nxt[j - 1];
            j += s[i] == t[j];
            if(j == m) {
                ans.push_back(i - m + 1);
                j = nxt[j - 1];
            }
        }
        return ans;
    }
}
```

manacher 算法

p 数组代表的是插入 '#' 的串的每个点开始的最长回文半径

比如 #a#b#a# 的 p[3] = 4，表示从 b 开始往外 3 个字母是最大的回文串

s 和 p 都是从 0 开始的。

```cpp
namespace manacher {
    char tmp[N];
    int p[N], len;
    void getP(char *s, int n) {
        len = 0;
        for(int i = 0; i < n; i++) {
            tmp[len++] = '#';
            tmp[len++] = s[i];
        }
        tmp[len++] = '#';
        memset(p, 0, sizeof(int) * (len + 10));
        int c = 0, r = 0;
        for(int i = 0; i < len; i++) {
            if(i <= r) p[i] = min(p[2 * c - i], r - i);
            else p[i] = 1;
            while(i - p[i] >= 0 && i + p[i] < len && tmp[i - p[i]] == tmp[i + p[i]])
                p[i]++;
            if(i + p[i] - 1 > r) {
                r = i + p[i] - 1;
                c = i;
            }
        }
    }
    int getlen() {
        return *max_element(p, p + len);
    }
    int getlen(string s) {
        getP(s);
        return getlen();
    }
}
```

回文自动机

回文自动机中，每个点表示一个本质不同的回文串。

有两个根，偶根 0，长度为 0，奇根 1，长度为-1.

偶根的 fail 指向奇根，奇根没有 fail，因为不可能失配 (匹配后长度为 1)。

fail 表示一个点的最长回文后缀

cnt 表示当前节点所能代表的回文串的数量

text 表示文本

son[x][y] 表示 x 点的回文串，前后各加一个 y，形成新的回文串。

```cpp
namespace PAM {
    int all, son[N][S], fail[N], cnt[N], len[N], text[N], last, tot;
    int newnode(int l) {
        memset(son[tot], 0, sizeof(int) * (S));
        cnt[tot] = 0; len[tot] = l;
        return tot++;
    }
    void init() {
        // 一定注意主函数是否运行了这一句
        last = tot = all = 0;
        newnode(0); newnode(-1);
        text[0] = -1; fail[0] = 1;
    }
    int getfail(int x) {
        while(text[all - len[x] - 1] != text[all]) x = fail[x];
        return x;
    }
    void add(int w) {
        text[++all] = w;
        int x = getfail(last);
        if(!son[x][w]) {
            int y = newnode(len[x] + 2);
            fail[y] = son[getfail(fail[x])][w];
            son[x][w] = y;
        }
        cnt[last = son[x][w]]++;
    }
    void count() {for(int i = tot - 1; ~i; i--) cnt[fail[i]] += cnt[i];}
}
```

AC 自动机

```cpp
struct ac_automaton {
    int t[N][26], danger[N], tot, fail[N];
    int dp[N][N];
    void init() {
        tot = -1;
        _new();
    }
    int _new() {
        tot++;
        memset(t[tot], 0, sizeof(t[tot]));
        danger[tot] = 0;
        fail[tot] = 0;
        return tot;
    }
    void Insert(const char *s) {
        int u = 0;
        for(int i = 0; s[i]; i++) {
            if(!t[u][mp[s[i]]]) t[u][s[i] - 'a'] = _new();
            u = t[u][mp[s[i]]];
        }
        danger[u] = 1;
    }
    void build() {
        queue<int> q;
        for(int i = 0; i < 26; i++) {
            if(t[0][i]) {
                fail[t[0][i]] = 0;
                q.push(t[0][i]);
            }
        }
        while(q.size()) {
            int u = q.front(); q.pop();
            danger[u] |= danger[fail[u]];
            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  };
```

区间本质不同子串个数

```
1   // 写一个线段树，写一个 SAM（记录每个字符加入时对应的节点位置），补全 LCT 的 splay, rotate
2   struct LCT {
3       void pushtag(int x, int y) {
4           if(x == 0) return;
5           val[x] = y;
6           tag[x] = y;
7       }
8       void pushdown(int x) {
9           if(tag[x]) {
10              if(ch[x][0]) pushtag(ch[x][0], tag[x]);
11              if(ch[x][1]) pushtag(ch[x][1], tag[x]);
12              tag[x] = 0;
13          }
14      }
15      void access(int x, int v) {
16          int y;
17          for(y = 0; x; y = x, x = fa[x]) {
18              splay(x);
19              ch[x][1] = y;
20              if(val[x]) {
21                  sgt.modify(val[x] - sam.len[x] + 1, val[x] - sam.len[fa[x]], -1);
22              }
23          }
24          pushtag(y, v); sgt.modify(1, v, 1);
25      }
26  } lct;
27  string s;
28  vector<pair<int, int> > v[N];
29  int n, ans[N];
30  void work() {
31      cin >> s;
32      cin >> n;
33      sgt.setLimit(s.size() + 3);
34      for(int i = 1; i <= n; i++) {
35          int l, r;
36          cin >> l >> r;
37          v[r].push_back({i, l});
38      }
39      for(int i = 0; i < s.size(); i++) sam.insert(s[i] - 'a', i + 1);
40      for(int i = 2; i <= sam.num; i++) lct.fa[i] = sam.fail[i];
41      for(int i = 1; i <= s.size(); i++) {
42          lct.access(sam.p[i], i);
```

```
43          for(auto item : v[i]) {
44              ans[item.first] = sgt.query(item.second, i);
45          }
46      }
47      for(int i = 1; i <= n; i++) cout << ans[i] << endl;
48  }
```

杂项

取模还原成分数

摘自 EI 的博客

```
1  pair<int, int> approx(int p, int q, int A) {
2      int x = q, y = p, a = 1, b = 0;
3      while (x > A) {
4          swap(x, y); swap(a, b);
5          a -= x / y * b;
6          x %= y;
7      }
8      return make_pair(x, a);
9  }
```

快读

```
1  #define gc()(is==it?it=(is=in)+fread(in,1,Q,stdin),(is==it?EOF:*is++):*is++)
2  const int Q=(1<<24)+1;
3  char in[Q],*is=in,*it=in,c;
4  void read(int &n){
5      for(n=0;(c=gc())<'0'||c>'9';);
6      for(;c<='9'&&c>='0';c=gc())n=n*10+c-48;
7  }
```

int128

```
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

```
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

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

随机数种子

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

T(5) 求任意 int log2

```
inline int LOG2_1(unsigned x){
    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};
    x|=x>>1; x|=x>>2; x|=x>>4; x|=x>>8; x|=x>>16;
    return tb[x*0x07C4ACDDu>>27];
}
```

O(1) 求 2 的整幂次 log2

```
inline int LOG2(unsigned x){  //x=2^k
    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};
    return tb[x*263572066>>27];
}
```

开启编译优化

```
#pragma GCC optimize(2)
#pragma GCC optimize(3)
#pragma GCC optimize("Ofast")
#pragma GCC optimize("inline")
#pragma GCC optimize("-fgcse")
#pragma GCC optimize("-fgcse-lm")
#pragma GCC optimize("-fipa-sra")
#pragma GCC optimize("-ftree-pre")
#pragma GCC optimize("-ftree-vrp")
#pragma GCC optimize("-fpeephole2")
#pragma GCC optimize("-ffast-math")
#pragma GCC optimize("-fsched-spec")
#pragma GCC optimize("unroll-loops")
#pragma GCC optimize("-falign-jumps")
#pragma GCC optimize("-falign-loops")
#pragma GCC optimize("-falign-labels")
#pragma GCC optimize("-fdevirtualize")
#pragma GCC optimize("-fcaller-saves")
#pragma GCC optimize("-fcrossjumping")
#pragma GCC optimize("-fthread-jumps")
#pragma GCC optimize("-funroll-loops")
#pragma GCC optimize("-fwhole-program")
#pragma GCC optimize("-freorder-blocks")
#pragma GCC optimize("-fschedule-insns")
#pragma GCC optimize("inline-functions")
#pragma GCC optimize("-ftree-tail-merge")
#pragma GCC optimize("-fschedule-insns2")
#pragma GCC optimize("-fstrict-aliasing")
#pragma GCC optimize("-fstrict-overflow")
#pragma GCC optimize("-falign-functions")
#pragma GCC optimize("-fcse-skip-blocks")
```

```
32    #pragma GCC optimize("-fcse-follow-jumps")
33    #pragma GCC optimize("-fsched-interblock")
34    #pragma GCC optimize("-fpartial-inlining")
35    #pragma GCC optimize("no-stack-protector")
36    #pragma GCC optimize("-freorder-functions")
37    #pragma GCC optimize("-findirect-inlining")
38    #pragma GCC optimize("-fhoist-adjacent-loads")
39    #pragma GCC optimize("-frerun-cse-after-loop")
40    #pragma GCC optimize("inline-small-functions")
41    #pragma GCC optimize("-finline-small-functions")
42    #pragma GCC optimize("-ftree-switch-conversion")
43    #pragma GCC optimize("-foptimize-sibling-calls")
44    #pragma GCC optimize("-fexpensive-optimizations")
45    #pragma GCC optimize("-funsafe-loop-optimizations")
46    #pragma GCC optimize("inline-functions-called-once")
47    #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!!!!!!!!

经典题

区间回文子串计数（非本质不同）

对于一个询问，考虑一个回文中心向两边拓展，在一个区间左半边的会被左端点挡住，右半边会被右端点挡住。所以我们左右半边分别计算答案。

```
input:
3 3
aba
1 3
1 2
2 2
output:
4
2
1
```

```cpp
const int N = 4e6 + 1009;
namespace manacher {
    char tmp[N];
    int p[N], len;
    void getP(char *s, int n) {
        len = 0;
        for(int i = 0; i < n; i++) {
            tmp[len++] = '#';
            tmp[len++] = s[i];
        }
        tmp[len++] = '#';
        memset(p, 0, sizeof(int) * (len + 10));
        int c = 0, r = 0;
        for(int i = 0; i < len; i++) {
            if(i <= r) p[i] = min(p[2 * c - i], r - i);
            else p[i] = 1;
            while(i - p[i] >= 0 && i + p[i] < len && tmp[i - p[i]] == tmp[i + p[i]]) p[i]++;
            if(i + p[i] - 1 > r) {
                r = i + p[i] - 1;
                c = i;
            }
        }
        for(int i = 0; i < len; i++)p[i]--;
    }
};
struct node {
    int l, r, id;
};
struct SGT {
    long long sum[N];
    int lzt[N];
    int n;
    void set(int x) {
        n = x;
        build(1, n, 1);
    }
    void update(int rt) {
        sum[rt] = sum[lson] + sum[rson];
    }
    void pushdown(int l, int r, int rt) {
        if(lzt[rt] == 0) return ;
        sum[lson] += (Mid - l + 1) * lzt[rt]; lzt[lson] += lzt[rt];
        sum[rson] += (r - Mid) * lzt[rt]; lzt[rson] += lzt[rt];
        lzt[rt] = 0;
    }
    void build(int l, int r, int rt) {
        lzt[rt] = 0;
        if(l == r) {
            sum[rt] = 0;
            return ;
```

```cpp
                }
                build(l, Mid, lson);
                build(Mid + 1, r, rson);
                update(rt);
            }
            void modify(int l, int r, int L, int R, int rt, int x) {
                if(L <= l && r <= R) {
                    sum[rt] += (r - l + 1) * x;
                    lzt[rt] += x;
                    return ;
                }
                pushdown(l, r, rt);
                if(L <= Mid) modify(l, Mid, L, R, lson, x);
                if(Mid <  R) modify(Mid + 1, r, L, R, rson, x);
                update(rt);
            }
            long long query(int l, int r, int L, int R, int rt) {
                if(L <= l && r <= R) return sum[rt];
                pushdown(l, r, rt);
                long long ans = 0;
                if(L <= Mid) ans += query(l, Mid, L, R, lson);
                if(Mid <  R) ans += query(Mid + 1, r, L, R, rson);
                return ans;
            }
            long long query(int l, int r) {
                if(l > r) return 0;
                return query(1, n,l, r, 1);
            }
            void modify(int l, int r, int x) {
                if(l > r) return ;
                modify(1, n, l, r, 1, x);
            }
    } T;
    vector<node> v;
    long long ans[N];
    int n, q;
    char s[N];
    void work() {
        cin >> n >> q;
        cin >> s;
        manacher::getP(s, n);
        for(int i = 1; i <= q; i++) {
            int l, r;
            cin >> l >> r;
            l = l * 2 - 1;
            r = r * 2 + 1;
            v.push_back({l, r, i});
        }
        int j = 1;
        std::sort(v.begin(), v.end(), [](const node &a, const node &b) {
            return (a.l + a.r) / 2 < (b.l + b.r) / 2;
        });
        T.set(2 * n + 1);
        for(int i = 0; i < q; i++) {
            int mid = (v[i].l + v[i].r) / 2;
            while(j <= mid) {
                j++;
            }
            ans[v[i].id] += T.query(v[i].l, mid);
        }

        std::sort(v.begin(), v.end(), [](const node &a, const node &b) {
            return (a.l + a.r) / 2 > (b.l + b.r) / 2;
        });
        j = 2 * n + 1;
        T.set(2 * n + 1);
        for(int i = 0; i < q; i++) {
            int mid = (v[i].l + v[i].r) / 2 + 1;
            while(j >= mid) {
                T.modify(j, j + manacher::p[j - 1], 1);
                j--;
```

```
122            }
123            ans[v[i].id] += T.query(mid, v[i].r);
124        }
125        std::sort(v.begin(), v.end(), [](const node &a, const node &b) {
126            return a.id < b.id;
127        });
128        for(int i = 1; i <= q; i++) {
129            ans[i] -= (v[i - 1].r - v[i - 1].l + 2) / 2;
130            cout << ans[i] / 2 << endl;
131        }
132    }
```