算法模板

刘阳 袁昊 邓宏亮

2019 年 8 月 11 日

目录

1 计算几何

1.1 自适应辛普森

```
typedef double db;
struct Simpson {
  /* 系数 */
  db F(db x) { return /* 表达式 */; }
  db Simpson(db l, db r) {
    db m = (l + r) / 2.0;
    return (F(l) + 4 * F(m) + F(r)) * (r - l) / 6.0;
  }
  db Asr(db l, db r, db ans, db eps) {
    db m = (l + r) / 2.0;
    db l_ans = Simpson(l, m), r_ans = Simpson(m, r);
    if (fabs(l_ans + r_ans - ans) <= 15.0 * eps) return
         l_ans + r_ans + (l_ans + r_ans - ans) / 15.0;
    return Asr(l, m, l_ans, eps / 2.0) + Asr(m, r, r_ans,
          eps / 2.0);
  }
};
```

1.2 皮克定理

```
polygon: S = in + (on / 2) - 1
```

1.3 动态凸包

```
// CodeForces 70D 动态凸包
#include <bits/stdc++.h>
typedef double db;
const int maxn = 1e5 + 5;
const db eps = 1e-9;
int Sgn(db k) { return fabs(k) < eps ? 0 : (k < 0 ? -1 :
     1); }
int Cmp(db k1, db k2) { return Sgn(k1 - k2); }
struct point { db x, y; };
point operator - (point k1, point k2) { return (point){k1
     .x - k2.x, k1.y - k2.y}; }
point operator + (point k1, point k2) { return (point){k1
     .x + k2.x, k1.y + k2.y}; }
db operator * (point k1, point k2) { return k1.x * k2.x +
      k1.y * k2.y; }
db operator ^ (point k1, point k2) { return k1.x * k2.y -
      k1.y * k2.x; }
db GetLen(point k) { return sqrt(k * k); }
int n;
point basic;
point p[maxn];
std::set<point> set;
bool operator < (point k1, point k2) {
  k1 = k1 - basic; k2 = k2 - basic;
  db Ang1 = atan2(k1.y, k1.x), Ang2 = atan2(k2.y, k2.x);
  db Len1 = GetLen(k1), Len2 = GetLen(k2);
  if (Cmp(Ang1, Ang2) != 0) return Cmp(Ang1, Ang2) < 0;
  return Cmp(Len1, Len2) < 0;
}
std::set<point>::iterator Prev(std::set<point>::iterator
     k) {
  if (k == set.begin()) k = set.end();
  return --k;
}
std::set<point>::iterator Next(std::set<point>::iterator
     k) {
  ++k;
  return k == set.end() ? set.begin() : k;
}
bool Query(point k) {
  std::set<point>::iterator it = set.lower_bound(k);
  if (it == set.end()) it = set.begin();
  return Sgn((k - *(Prev(it))) ^ (*(it) - *(Prev(it))))
       <= 0;
}
void Insert(point k) {
  if (Query(k)) return;
  set.insert(k);
  std::set<point>::iterator cur = Next(set.find(k));
  while (set.size() > 3 && Sgn((k - *(Next(cur))) ^ (*(
       cur) - *(Next(cur)))) <= 0) {
    set.erase(cur);
    cur = Next(set.find(k));
  }
  cur = Prev(set.find(k));
  while (set.size() > 3 && Sgn((k - *(cur)) ^ (*(cur) -
       *(Prev(cur)))) >= 0) {
    set.erase(cur);
    cur = Prev(set.find(k));
  }
}
int main() {
  scanf("%d", &n);
  basic.x = basic.y = 0.0;
  for (int i = 1, T; i <= 3; ++i) {
    scanf("%d%lf%lf", &T, &p[i].x, &p[i].y);
    basic.x += p[i].x; basic.y += p[i].y;
  }
  basic.x /= 3.0; basic.y /= 3.0;
  for (int i = 1; i <= 3; ++i) set.insert(p[i]);
  for (int i = 4, T; i <= n; ++i) {
    scanf("%d%lf%lf", &T, &p[i].x, &p[i].y);
    if (T == 1) Insert(p[i]);
```

```
    else {
      if (Query(p[i])) printf("YES\n");
      else printf("NO\n");
    }
  }
  return 0;
}
```

2 数论

2.1 快速幂

```
long long p;
long long mul(long long a, long long b) {
    long long ans = 0;
    while(b) {
        if(b & 1) ans = (ans + a ) % p;
        b >>= 1;
        a = (a + a) % p;
    }
    return ans;
}
long long pow(long long a, long long b) {
    long long res = 1;
    long long base = a;
    while(b) {
        if(b & 1) res = mul(res, base) % p;
        base = mul(base, base) % p;
        b >>= 1;
    }
    return res;
}
```

3 数据结构

3.1 线段树套伸展树

```
/* BZOJ 3196（线段树套伸展树）
1.查询k在区间内的排名
2.查询区间内排名为k的值
3.修改某一位值上的数值
4.查询k在区间内的前驱(前驱定义为小于x，且最大的数)
5.查询k在区间内的后继(后继定义为大于x，且最小的数) */
#include <bits/stdc++.h>
const int inf = 2147483647;
const int maxn = 5e4 + 5;
const int maxm = maxn * 25;
int n;
int arr[maxn];
namespace SplayTree {
  int rt[maxm], tot;
  int fa[maxm], son[maxm][2];
  int val[maxm], cnt[maxm];
  int sz[maxm];
  void Push(int o) {
    sz[o] = sz[son[o][0]] + sz[son[o][1]] + cnt[o];
  }
  bool Get(int o) {
    return o == son[fa[o]][1];
  }
  void Clear(int o) {
    son[o][0] = son[o][1] = fa[o] = val[o] = sz[o] = cnt[
        o] = 0;
  }
  void Rotate(int o) {
    int p = fa[o], q = fa[p], ck = Get(o);
    son[p][ck] = son[o][ck ^ 1];
    fa[son[o][ck ^ 1]] = p;
    son[o][ck ^ 1] = p;
    fa[p] = o; fa[o] = q;
    if (q) son[q][p == son[q][1]] = o;
    Push(p); Push(o);
  }
  void Splay(int &root, int o) {
    for (int f = fa[o]; (f = fa[o]); Rotate(o))
      if (fa[f]) Rotate(Get(o) == Get(f) ? f : o);
    root = o;
  }
  void Insert(int &root, int x) {
    if (!root) {
      val[++tot] = x;
      cnt[tot]++;
      root = tot;
      Push(root);
      return;
    }
    int cur = root, f = 0;
    while (true) {
      if (val[cur] == x) {
        cnt[cur]++;
        Push(cur); Push(f);
        Splay(root, cur);
        break;
      }
      f = cur;
      cur = son[cur][val[cur] < x];
      if (!cur) {
        val[++tot] = x;
        cnt[tot]++;
        fa[tot] = f;
        son[f][val[f] < x] = tot;
        Push(tot); Push(f);
        Splay(root, tot);
        break;
      }
    }
  }
  int GetRank(int &root, int x) {
    int ans = 0, cur = root;
    while (cur) {
      if (x < val[cur]) {
        cur = son[cur][0];
        continue;
      }
      ans += sz[son[cur][0]];
      if (x == val[cur]) {
        Splay(root, cur);
        return ans;
      }
      if (x > val[cur]) {
        ans += cnt[cur];
        cur = son[cur][1];
      }
    }
    return ans;
  }
  int GetKth(int &root, int k) {
    int cur = root;
    while (true) {
      if (son[cur][0] && k <= sz[son[cur][0]]) cur = son[
          cur][0];
      else {
        k -= cnt[cur] + sz[son[cur][0]];
        if (k <= 0) return cur;
        cur = son[cur][1];
      }
    }
  }
  int Find(int &root, int x) {
```

```
    int ans = 0, cur = root;
    while (cur) {
      if (x < val[cur]) {
        cur = son[cur][0];
        continue;
      }
      ans += sz[son[cur][0]];
      if (x == val[cur]) {
        Splay(root, cur);
        return ans + 1;
      }
      ans += cnt[cur];
      cur = son[cur][1];
    }
  }
  int GetPrev(int &root) {
    int cur = son[root][0];
    while (son[cur][1]) cur = son[cur][1];
    return cur;
  }
  int GetPrevVal(int &root, int x) {
    int ans = -inf, cur = root;
    while (cur) {
      if (x > val[cur]) {
        ans = std::max(ans, val[cur]);
        cur = son[cur][1];
        continue;
      }
      cur = son[cur][0];
    }
    return ans;
  }
  int GetNext(int &root) {
    int cur = son[root][1];
    while (son[cur][0]) cur = son[cur][0];
    return cur;
  }
  int GetNextVal(int &root, int x) {
    int ans = inf, cur = root;
    while (cur) {
      if (x < val[cur]) {
        ans = std::min(ans, val[cur]);
        cur = son[cur][0];
        continue;
      }
      cur = son[cur][1];
    }
    return ans;
  }
  void Delete(int &root, int x) {
    Find(root, x);
    if (cnt[root] > 1) {
      cnt[root]--;
      Push(root);
      return;
    }
    if (!son[root][0] && !son[root][1]) {
      Clear(root);
      root = 0;
      return;
    }
    if (!son[root][0]) {
      int cur = root;
      root = son[root][1];
      fa[root] = 0;
      Clear(cur);
      return;
    }
    if (!son[root][1]) {
      int cur = root;
      root = son[root][0];
      fa[root] = 0;
      Clear(cur);
      return;
    }
    int p = GetPrev(root), cur = root;
    Splay(root, p);
    fa[son[cur][1]] = p;
    son[p][1] = son[cur][1];
    Clear(cur);
    Push(root);
  }
};

namespace SegTree {
  int tree[maxn * 4];
  void Build(int o, int l, int r) {
    for (int i = l; i <= r; ++i) SplayTree::Insert(tree[o
        ], arr[i - 1]);
    if (l == r) return;
    int m = (l + r) / 2;
    Build(o * 2, l, m);
    Build(o * 2 + 1, m + 1, r);
  }
  void Modify(int o, int l, int r, int ll, int rr, int u,
      int v) {
    SplayTree::Delete(tree[o], u); SplayTree::Insert(tree
        [o], v);
    if (l == r) return;
    int m = (l + r) / 2;
    if (ll <= m) Modify(o * 2, l, m, ll, rr, u, v);
    if (rr > m) Modify(o * 2 + 1, m + 1, r, ll, rr, u, v)
        ;
  }
  int QueryRank(int o, int l, int r, int ll, int rr, int
      v) {
    if (ll <= l && rr >= r) return SplayTree::GetRank(
        tree[o], v);
```

```
    int m = (l + r) / 2, ans = 0;
    if (ll <= m) ans += QueryRank(o * 2, l, m, ll, rr, v)
        ;
    if (rr > m) ans += QueryRank(o * 2 + 1, m + 1, r, ll,
         rr, v);
    return ans;
  }
  int QueryPrev(int o, int l, int r, int ll, int rr, int
      v) {
    if (ll <= l && rr >= r) return SplayTree::GetPrevVal(
        tree[o], v);
    int m = (l + r) / 2, ans = -inf;
    if (ll <= m) ans = std::max(ans, QueryPrev(o * 2, l,
        m, ll, rr, v));
    if (rr > m) ans = std::max(ans, QueryPrev(o * 2 + 1,
        m + 1, r, ll, rr, v));
    return ans;
  }
  int QueryNext(int o, int l, int r, int ll, int rr, int
      v) {
    if (ll <= l && rr >= r) return SplayTree::GetNextVal(
        tree[o], v);
    int m = (l + r) / 2, ans = inf;
    if (ll <= m) ans = std::min(ans, QueryNext(o * 2, l,
        m, ll, rr, v));
    if (rr > m) ans = std::min(ans, QueryNext(o * 2 + 1,
        m + 1, r, ll, rr, v));
    return ans;
  }
  int QueryKth(int ll, int rr, int v) {
    int l = 0, r = 1e8 + 10;
    while (l < r) {
      int m = ((l + r) / 2) + 1;
      if (QueryRank(1, 1, n, ll, rr, m) < v) l = m;
      else r = m - 1;
    }
    return l;
  }
};
int main() {
  std::ios::sync_with_stdio(false);
  std::cout.tie(0);
  std::cin.tie(0);
  int m;
  std::cin >> n >> m;
  for (int i = 0; i < n; ++i) std::cin >> arr[i];
  SplayTree::tot = 0;
  SegTree::Build(1, 1, n);
  for (int i = 0, op, l, r, pos, k; i < m; ++i) {
    std::cin >> op;
    if (op == 1) {
      std::cin >> l >> r >> k;
      std::cout << SegTree::QueryRank(1, 1, n, l, r, k) +
          1 << '\n';
    }
    else if (op == 2) {
      std::cin >> l >> r >> k;
      std::cout << SegTree::QueryKth(l, r, k) << '\n';
    }
    else if (op == 3) {
      std::cin >> pos >> k;
      SegTree::Modify(1, 1, n, pos, pos, arr[pos - 1], k)
          ;
      arr[pos - 1] = k;
    }
    else if (op == 4) {
      std::cin >> l >> r >> k;
      std::cout << SegTree::QueryPrev(1, 1, n, l, r, k)
          << '\n';
    }
    else if (op == 5) {
      std::cin >> l >> r >> k;
      std::cout << SegTree::QueryNext(1, 1, n, l, r, k)
          << '\n';
    }
  }
  return 0;
}
```

3.2 线段树

3.2.1 线段树合并

```
// BZOJ2212: 交换左右子树后最小逆序对
#include <bits/stdc++.h>
const int maxn = 1e7 + 5;
template <typename t>
inline bool Read(t &ret) {
  char c; int sgn;
  if (c = getchar(), c == EOF) return false;
  while (c != '-' && (c < '0' || c > '9')) c = getchar();
  sgn = (c == '-') ? -1 : 1;
  ret = (c == '-') ? 0 : (c - '0');
  while (c = getchar(), c >= '0' && c <= '9') ret = ret *
      10 + (c - '0');
  ret *= sgn;
  return true;
}
struct node {
  int sz, lson, rson;
  node() { sz = lson = rson = 0; }
};
int n;
int tot;
node tree[maxn];
```

```
long long ans1, ans2;
long long ans;
int Build(int l, int r, int c) {
  tree[++tot].sz = 1;
  if (l == r) return tot;
  int m = (l + r) / 2, o = tot;
  if (c <= m) tree[o].lson = Build(l, m, c);
  else tree[o].rson = Build(m + 1, r, c);
  return o;
}
int Merge(int l, int r, int x, int y) {
  if (!x || !y) return x + y;
  if (l == r) {
    tree[++tot].sz = tree[x].sz + tree[y].sz;
    return tot;
  }
  ans1 += 1ll * tree[tree[x].rson].sz * tree[tree[y].lson
       ].sz;
  ans2 += 1ll * tree[tree[x].lson].sz * tree[tree[y].rson
       ].sz;
  int m = (l + r) / 2, o = ++tot;
  tree[o].lson = Merge(l, m, tree[x].lson, tree[y].lson);
  tree[o].rson = Merge(m + 1, r, tree[x].rson, tree[y].
       rson);
  tree[o].sz = tree[x].sz + tree[y].sz;
  return o;
}
int Dfs() {
  int c = 0;
  Read(c);
  if (c) return Build(1, n, c);
  int o = Merge(1, n, Dfs(), Dfs());
  ans += std::min(ans1, ans2);
  ans1 = ans2 = 0;
  return o;
}
int main() {
  Read(n);
  Dfs();
  printf("%lld", ans);
  return 0;
}
```

3.2.2 线段树

```
const int maxn = "Edit";
struct SegTree {
  int n;
  long long sum[maxn * 4], lazy[maxn * 4];
  long long Unite(const long long &k1, const long long &
       k2) {
    return k1 + k2;
  }
  void Pull(int o) {
    sum[o] = Unite(sum[o * 2], sum[o * 2 + 1]);
  }
  void Push(int o, int l, int r) {
    int m = (l + r) / 2;
    if (lazy[o] != 0) {
      sum[o * 2] += (m - l + 1) * lazy[o];
      sum[o * 2 + 1] += (r - m) * lazy[o];
      lazy[o * 2] += lazy[o];
      lazy[o * 2 + 1] += lazy[o];
      lazy[o] = 0;
    }
  }
  void Build(int o, int l, int r, long long arr[]) {
    sum[o] = lazy[o] = 0;
    if (l == r) {
      sum[o] = arr[l];
      return;
    }
    int m = (l + r) / 2;
    Build(o * 2, l, m, arr);
    Build(o * 2 + 1, m + 1, r, arr);
    Pull(o);
  }
  void Init(int _n, long long arr[]) {
    n = _n;
    Build(1, 1, n, arr);
  }
  void Modify(int o, int l, int r, int ll, int rr, long
       long v) {
    if (ll <= l && rr >= r) {
      sum[o] += (r - l + 1) * v;
      lazy[o] += v;
      return;
    }
    Push(o, l, r);
    int m = (l + r) / 2;
    if (ll <= m) Modify(o * 2, l, m, ll, rr, v);
    if (rr > m) Modify(o * 2 + 1, m + 1, r, ll, rr, v);
    Pull(o);
  }
  void Modify(int ll, int rr, long long v) {
    Modify(1, 1, n, ll, rr, v);
  }
  long long Query(int o, int l, int r, int ll, int rr) {
    if (ll <= l && rr >= r) return sum[o];
    Push(o, l, r);
    int m = (l + r) / 2;
    long long ret = 0;
    if (ll <= m) ret = Unite(ret, Query(o * 2, l, m, ll,
         rr));
    if (rr > m) ret = Unite(ret, Query(o * 2 + 1, m + 1,
         r, ll, rr));
```

```
    return ret;
  }
  long long Query(int ll, int rr) {
    return Query(1, 1, n, ll, rr);
  }
};
```

3.2.3 矩形面积异或并

```
// CodeForces GYM 101982 F 矩形面积异或并
#include <bits/stdc++.h>
std::vector<int> x;
int Get(int k) {
  return std::lower_bound(x.begin(), x.end(), k) - x.
       begin();
}
struct SegTree {
  struct node {
    int v, lazy;
    node() { v = lazy = 0; }
  };
  int n;
  std::vector<node> tree;
  node Unite(const node &k1, const node &k2) {
    node ans;
    ans.v = k1.v + k2.v;
    return ans;
  }
  void Pull(int o) {
    tree[o] = Unite(tree[o * 2], tree[o * 2 + 1]);
  }
  void Push(int o, int l, int r) {
    int m = (l + r) / 2;
    if (tree[o].lazy != 0) {
      tree[o * 2].v = x[m] - x[l - 1] - tree[o * 2].v;
      tree[o * 2 + 1].v = x[r] - x[m] - tree[o * 2 + 1].v
           ;
      tree[o * 2].lazy ^= 1;
      tree[o * 2 + 1].lazy ^= 1;
      tree[o].lazy = 0;
    }
  }
  void Build(int o, int l, int r) {
    if (l == r) return;
    int m = (l + r) / 2;
    Build(o * 2, l, m);
    Build(o * 2 + 1, m + 1, r);
    Pull(o);
  }
  SegTree(int _n): n(_n) {
    tree.resize(n << 2);
    Build(1, 1, n);
  }
  void Modify(int o, int l, int r, int ll, int rr) {
    if (ll <= l && rr >= r) {
      tree[o].v = x[r] - x[l - 1] - tree[o].v;
      tree[o].lazy ^= 1;
      return;
    }
    Push(o, l, r);
    int m = (l + r) / 2;
    if (ll <= m) Modify(o * 2, l, m, ll, rr);
    if (rr > m) Modify(o * 2 + 1, m + 1, r, ll, rr);
    Pull(o);
  }
  void Modify(int ll, int rr) {
    Modify(1, 1, n, ll, rr);
  }
  node Query(int o, int l, int r, int ll, int rr) {
    if (ll <= l && rr >= r) return tree[o];
    Push(o, l, r);
    int m = (l + r) / 2;
    node ans;
    if (ll <= m) ans = Unite(ans, Query(o * 2, l, m, ll,
         rr));
    if (rr > m) ans = Unite(ans, Query(o * 2 + 1, m + 1,
         r, ll, rr));
    Pull(o);
    return ans;
  }
  node Query() {
    return Query(1, 1, n, 1, n);
  }
};
struct seg { int l, r, h, flag; };
bool operator < (seg k1, seg k2) { return k1.h < k2.h; }
std::vector<seg> s;
int main() {
  std::ios::sync_with_stdio(false);
  std::cout.tie(nullptr);
  std::cin.tie(nullptr);
  int n; std::cin >> n;
  for (int i = 0, x1, y1, x2, y2; i < n; ++i) {
    std::cin >> x1 >> y1 >> x2 >> y2;
    if (x1 > x2) std::swap(x1, x2);
    if (y1 > y2) std::swap(y1, y2);
    x.emplace_back(x1); x.emplace_back(x2);
    s.emplace_back((seg){x1, x2, y1, 1});
    s.emplace_back((seg){x1, x2, y2, -1});
  }
  sort(s.begin(), s.end());
  sort(x.begin(), x.end());
  x.erase(unique(x.begin(), x.end()), x.end());
  SegTree tree((int)x.size());
  long long ans = 0;
  for (int i = 0, l, r; i < (int)s.size() - 1; ++i) {
```

```
    l = Get(s[i].l), r = Get(s[i].r);
    tree.Modify(l + 1, r);
    ans += (long long)tree.Query().v * (s[i + 1].h - s[i
        ].h);
  }
  std::cout << ans << '\n';
  return 0;
}
```

3.2.4 矩形面积并

```
// HDU 1542 矩形面积并
#include <bits/stdc++.h>
typedef double db;
const int maxn = 1e2 + 5;
const db eps = 1e-9;
int Sgn(db k) {
  return std::fabs(k) < eps ? 0 : (k < 0 ? -1 : 1);
}
int Cmp(db k1, db k2) {
  return Sgn(k1 - k2);
}
struct seg {
  db l, r, h;
  int flag;
};
bool operator < (seg &k1, seg &k2) {
  return Cmp(k1.h, k2.h) < 0;
}
std::vector<seg> segs;
std::vector<db> pos;
int BinarySearch(db k) {
  int ret = (int)pos.size() - 1, l = 0, r = (int)pos.size
      () - 1;
  while (l <= r) {
    int m = (l + r) >> 1;
    if (Cmp(pos[m], k) >= 0) {
      ret = m;
      r = m - 1;
    }
    else l = m + 1;
  }
  return ret;
}
struct node {
  int l, r, cnt;
  db len;
};
node seg_tree[maxn * 10];
void Pull(int o) {
  if (seg_tree[o].cnt) seg_tree[o].len = pos[seg_tree[o].
      r + 1] - pos[seg_tree[o].l];
  else if (seg_tree[o].l == seg_tree[o].r) seg_tree[o].
      len = 0.0;
  else seg_tree[o].len = seg_tree[o << 1].len + seg_tree[
      o << 1 | 1].len;
}
void Build(int l, int r, int o) {
  seg_tree[o].l = l; seg_tree[o].r = r;
  seg_tree[o].cnt = 0; seg_tree[o].len = 0.0;
  if (l == r) return;
  int Mid = (l + r) >> 1;
  Build(l, Mid, o << 1);
  Build(Mid + 1, r, o << 1 | 1);
  Pull(o);
}
void Update(int l, int r, int v, int o) {
  if (l <= seg_tree[o].l && r >= seg_tree[o].r) {
    seg_tree[o].cnt += v;
    Pull(o);
    return;
  }
  int Mid = (seg_tree[o].l + seg_tree[o].r) >> 1;
  if (r <= Mid) Update(l, r, v, o << 1);
  else if (l > Mid) Update(l, r, v, o << 1 | 1);
  else {
    Update(l, Mid, v, o << 1);
    Update(Mid + 1, r, v, o << 1 | 1);
  }
  Pull(o);
}
int cas;
int n;
db x1, y1, x2, y2;
db ans;
int main() {
  while (~scanf("%d", &n) && n) {
    segs.clear();
    pos.clear();
    for (int i = 0; i < n; ++i) {
      scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
      segs.push_back((seg){x1, x2, y1, 1});
      segs.push_back((seg){x1, x2, y2, -1});
      pos.push_back(x1);
      pos.push_back(x2);
    }
    std::sort(segs.begin(), segs.end());
    std::sort(pos.begin(), pos.end(), [&](db k1, db k2) {
        return Cmp(k1, k2) < 0; });
    int cur = 1;
    for (int i = 1; i < (int)pos.size(); ++i)
      if (Cmp(pos[i], pos[i - 1]) != 0)
        pos[cur++] = pos[i];
    pos.erase(pos.begin() + cur, pos.end());
    Build(0, (int)pos.size(), 1);
    ans = 0.0;
```

```
    for (int i = 0; i < (int)segs.size() - 1; ++i) {
      int l = BinarySearch(segs[i].l), r = BinarySearch(
          segs[i].r);
      Update(l, r - 1, segs[i].flag, 1);
      ans += (segs[i + 1].h - segs[i].h) * seg_tree[1].
          len;
    }
    printf("Test case #%d\n", ++cas);
    printf("Total explored area: %.2lf\n\n", ans);
  }
  return 0;
}
```

3.3 树链剖分

```
const int maxn = "Edit";
int n;
long long val[maxn];
int fa[maxn], dep[maxn];
int sz[maxn], son[maxn];
int rk[maxn], top[maxn];
int id[maxn];
int dfs_clock;
std::vector<int> g[maxn];
void Dfs1(int u, int p, int d) {
  fa[u] = p;
  dep[u] = d;
  sz[u] = 1;
  for (int &v : g[u]) {
    if (v == p) continue;
    Dfs1(v, u, d + 1);
    sz[u] += sz[v];
    if (sz[v] > sz[son[u]]) son[u] = v;
  }
}
Dfs2(int u, int tp) {
  top[u] = tp;
  id[u] = ++dfs_clock;
  rk[dfs_clock] = u;
  if (!son[u]) return;
  Dfs2(son[u], tp);
  for (int &v : g[u]) {
    if (v == son[u] || v == fa[u]) continue;
    Dfs2(v, v);
  }
}
long long Modify(int u, int v, long long c) {
  while (top[u] != top[v]) {
    if (dep[top[u]] < dep[top[v]]) std::swap(u, v);
    /* modify c from [id[top[u]], id[u]] in val */
    u = fa[top[u]];
  }
  if (id[u] > id[v]) std::swap(u, v);
  /* modify c from [id[u], id[v]] in val */
}
long long Query(int u, int v) {
  long long ret = 0;
  while (top[u] != top[v]) {
    if (dep[top[u]] < dep[top[v]]) std::swap(u, v);
    ret += /* query from [id[top[u]], id[u]] in val */
    u = fa[top[u]];
  }
  if (id[u] > id[v]) std::swap(u, v);
  ret += /* query from [id[u], id[v]] in val */
  return ret;
}
```

3.4 树状数组

```
#define lowbit(x) (x&(-x))
const int maxn = "Edit";
struct BitTree {
  int arr[maxn];
  void Init() {
    memset(arr, 0, sizeof(arr));
  }
  void Modify(int idx, int x) {
    while (idx < maxn) {
      arr[idx] += x;
      idx += lowbit(idx);
    }
  }
  int Query(int idx) {
    int ret = 0;
    while (idx > 0) {
      ret += arr[idx];
      idx -= lowbit(idx);
    }
    return ret;
  }
  int GetRank(int x) {
    int ret = 1;
    --x;
    while (x) {
      ret += arr[x];
      x -= lowbit(x);
    }
    return ret;
  }
  int GetKth(int k) { // kth min
    int ret = 0, cnt = 0, max = log2(maxn);
    for (int i = max; i >= 0; --i) {
```

```
      ret += (1 << i);
      if (ret >= maxn || cnt += arr[ret] >= k) ret -= (1
          << i);
      else cnt += arr[ret];
    }
    return ++ret;
  }
  int GetPrev(int x) {
    return GetKth(GetRank(x) - 1);
  }
  int GetNext(int x) {
    return GetKth(GetRank(x) + 1);
  }
};
```

3.5 最近公共祖先

3.5.1 欧拉序 +RMQ

```
const int maxn = "Edit";
const int maxlog = "Edit";
int n;
std::vector<int> g[maxn];
int ele[maxn * 2], dep[maxn * 2];
int fi[maxn], fa[maxn];
int tot;
int dp[maxn * 2][maxlog];
void Dfs(int u, int p, int d) {
  ele[++tot] = u;
  fi[u] = tot;
  dep[tot] = d;
  fa[u] = p;
  for (int &v : g[u]) {
    if (v == p) continue;
    Dfs(v, u, d + 1);
    ele[++tot] = u;
    dep[tot] = d;
  }
}
void Init() {
  for(int i = 1; i <= 2 * n - 1; ++i) dp[i][0] = i;
  for (int j = 1; (1 << j) <= 2 * n - 1; ++j)
    for (int i = 1; i + (1 << j) - 1 <= 2 * n - 1; ++i)
      dp[i][j] = dep[dp[i][j - 1]] < dep[dp[i + (1 << j -
          1)][j - 1]] ? dp[i][j - 1] : dp[i + (1 << (j
          - 1))][j - 1];
}
int Query(int l, int r) {
  if (l > r) std::swap(l, r);
  int len = log2(r - l + 1);
  return dep[dp[l][len]] <= dep[dp[r - (1 << len) + 1][
      len]] ? dp[l][len] : dp[r - (1 << len) + 1][len];
}
int GetLCA(int u, int v) {
  return ele[Query(fi[u], fi[v])];
}
```

3.5.2 倍增

```
const int maxn = "Edit";
const int maxlog = "Edit";
int n, k; // k = log2(n) + 1
std::vecotr<int> g[maxn];
int anc[maxn][maxlog];
int dep[maxn];
// 从根节点开始深搜预处理
void Dfs(int u, int p, int d) {
  anc[u][0] = p;
  dep[u] = d;
  for (int &v : g[u]) {
    if (v == p) continue;
    Dfs(v, u, d + 1);
  }
}
void Swim(int &u, int h) {
  for (int i = 0; h > 0; ++i) {
    if (h & 1) u = anc[u][i];
    h >>= 1;
  }
}
int GetLCA(int u, int v) {
  if (dep[u] < dep[v]) std::swap(u, v);
  Swim(u, dep[u] - dep[v]);
  if (u == v) return v;
  for (int i = k - 1; i >= 0; --i) {
    if (anc[u][i] != anc[v][i]) {
      u = anc[u][i];
      v = anc[v][i];
    }
  }
  return anc[u][0];
}
```

3.5.3 tarjan

```
const int maxn = "Edit";
const int maxm = "Edit";
int n;
int pre[maxn];
```

```
int Find(int o) {
  return pre[o] == o ? o : pre[o] = Find(pre[o]);
}
void Union(int u, int v) {
  if (Find(u) != Find(v)) pre[Find(u)] = Find(v);
}
std::vector<int> g[maxn];
bool vis[maxn];
struct query { int v, id; };
std::vector<query> qry[maxm];
void Init() {
  for (int i = 1; i <= n; ++i) {
    pre[i] = i;
    vis[i] = false;
  }
}
void Tarjan(int u) {
  vis[u] = true;
  for (int &v : g[u]) {
    if (vis[v]) continue;
    Tarjan(v);
    Union(v, u);
  }
  for (query &q : qry[u]) {
    if (vis[q.v]) ans[q.id] = Find(q.v);
  }
}
```

3.6 伸展树

```
const int inf = "Edit"
const int maxn = "Edit";
struct SplayTree {
  int rt, tot;
  int fa[maxn], son[maxn][2];
  int val[maxn], cnt[maxn];
  int sz[maxn];
  bool lazy[maxn];
  void Pull(int o) {
    sz[o] = sz[son[o][0]] + sz[son[o][1]] + cnt[o];
  }
  void Push(int o) {
    if (lazy[o]) {
      std::swap(son[o][0], son[o][1]);
      if (son[o][0]) lazy[son[o][0]] ^= 1;
      if (son[o][1]) lazy[son[o][1]] ^= 1;
      lazy[o] = 0;
    }
  }
  bool Get(int o) {
    return o == son[fa[o]][1];
  }
  void Clear(int o) {
    son[o][0] = son[o][1] = fa[o] = val[o] = sz[o] = cnt[
        o] = 0;
  }
  void Rotate(int o) {
    int p = fa[o], q = fa[p], ck = Get(o);
    son[p][ck] = son[o][ck ^ 1];
    fa[son[o][ck ^ 1]] = p;
    son[o][ck ^ 1] = p;
    fa[p] = o; fa[o] = q;
    if (q) son[q][p == son[q][1]] = o;
    Pull(p); Pull(o);
  }
  void Splay(int o) {
    for (int f = fa[o]; f = fa[o], f; Rotate(o))
      if (fa[f]) Rotate(Get(o) == Get(f) ? f : o);
    rt = o;
  }
  // 旋转o节点到节点tar
  void Splay(int o, int tar = 0) {
    for (int f = fa[o]; (f = fa[o]) != tar; Rotate(o)) {
      Pull(fa[f]); Pull(f); Pull(o);
      if (fa[f] != tar) {
        if (Get(o) == Get(f)) Rotate(f);
        else Rotate(o);
      }
    }
    if (!tar) rt = o;
  }
  void Insert(int x) {
    if (!rt) {
      val[++tot] = x;
      cnt[tot]++;
      rt = tot;
      Pull(rt);
      return;
    }
    int cur = rt, f = 0;
    while (true) {
      if (val[cur] == x) {
        cnt[cur]++;
        Pull(cur); Pull(f);
        Splay(cur);
        break;
      }
      f = cur;
      cur = son[cur][val[cur] < x];
      if (!cur) {
        val[++tot] = x;
        cnt[tot]++;
        fa[tot] = f;
        son[f][val[f] < x] = tot;
        Pull(tot); Pull(f);
```

```
      Splay(tot);
      break;
    }
  }
}
int GetRank(int x) {
  int ans = 0, cur = rt;
  while (true) {
    if (x < val[cur]) cur = son[cur][0];
    else {
      ans += sz[son[cur][0]];
      if (x == val[cur]) {
        Splay(cur);
        return ans + 1;
      }
      ans += cnt[cur];
      cur = son[cur][1];
    }
  }
}
int GetKth(int k) {
  int cur = rt;
  while (true) {
    if (son[cur][0] && k <= sz[son[cur][0]]) cur = son[
        cur][0];
    else {
      k -= cnt[cur] + sz[son[cur][0]];
      if (k <= 0) return cur;
      cur = son[cur][1];
    }
  }
}
// 获取以r为根节点Splay Tree中的第k大个元素在Splay Tree
    中的位置
int Kth(int r, int k) {
  Pull(r);
  int tmp = sz[son[r][0]] + 1;
  if (tmp == k) return r;
  if (tmp > k) return Kth(son[r][0], k);
  else return Kth(son[r][1], k - tmp);
}
// Insert之后求前驱后继
int GetPrev() {
  int cur = son[rt][0];
  while (son[cur][1]) cur = son[cur][1];
  return cur;
}
int GetNext() {
  int cur = son[rt][1];
  while (son[cur][0]) cur = son[cur][0];
  return cur;
}
// 获取Splay Tree中以o为根节点子树的最小值位置
int GetMin(int o) {
  Pull(o);
  while (son[o][0]) {
    o = son[o][0];
    Pull(o);
  }
  return o;
}
// 获取Splay Tree中以o为根节点子树的最大值位置
int GetMax(int o) {
  Pull(o);
  while (son[o][1]) {
    o = son[o][1];
    Pull(o);
  }
  return o;
}
void Delete(int x) {
  GetRank(x);
  if (cnt[rt] > 1) {
    cnt[rt]--;
    Pull(rt);
    return;
  }
  if (!son[rt][0] && !son[rt][1]) {
    Clear(rt);
    rt = 0;
    return;
  }
  if (!son[rt][0]) {
    int cur = rt;
    rt = son[rt][1];
    fa[rt] = 0;
    Clear(cur);
    return;
  }
  if (!son[rt][1]) {
    int cur = rt;
    rt = son[rt][0];
    fa[rt] = 0;
    Clear(cur);
    return;
  }
  int p = GetPrev(), cur = rt;
  Splay(p);
  fa[son[cur][1]] = p;
  son[p][1] = son[cur][1];
  Clear(cur);
  Pull(rt);
}
/* 维护数组操作 */
// 翻转Splay Tree中l~r区间
void Reverse(int l, int r) {
  int o = Kth(rt, l), Y = Kth(rt, r);
  Splay(o, 0); Splay(Y, o);
```

```
    lazy[son[Y][0]] ^= 1;
  }
  // 建立Splay Tree
  void Build(int l, int r, int o) {
    if (l > r) return;
    int m = (l + r) >> 1;
    Build(l, m - 1, m);
    Build(m + 1, r, m);
    fa[m] = o;
    val[m] = /* 节点权值 */;
    lazy[m] = 0;
    Push(m);
    if (m < o) son[o][0] = m;
    else son[o][1] = m;
  }
  // 输出Splay Tree
  void Print(int o) {
    Pull(o);
    if (son[o][0]) Print(son[o][0]);
    // 哨兵节点判断
    if (val[o] != -inf && val[o] != inf) printf("%d ",
        val[o]);
    if (val[son[o][1]]) Print(son[o][1]);
  }
};
```

3.7 主席树

```
const int maxn = "Edit";
struct FuncSegTree {
  int tot;
  int rt[maxn];
  int lson[maxn * 40], rson[maxn * 40];
  int cnt[maxn * 40];
  int Build(int l, int r) {
    int o = ++tot, m = (l + r) / 2;
    cnt[o] = 0;
    if (l != r) {
      lson[o] = Build(l, m);
      rson[o] = Build(m + 1, r);
    }
    return o;
  }
  int Modify(int prev, int l, int r, int v) {
    int o = ++tot, m = (l + r) >> 1;
    lson[o] = lson[prev];
    rson[o] = rson[prev];
    cnt[o] = cnt[prev] + 1;
    if (l != r) {
      if (v <= m) lson[o] = Modify(lson[o], l,m, v);
      else rson[o] = Modify(rson[o], m + 1, r, v);
    }
    return o;
  }
  // 区间[u+1,v]静态第k小
  int Query(int u, int v, int l, int r, int k) {
    if (l == r) return l;
    int m = (l + r) / 1;
    int num = cnt[lson[v]] - cnt[lson[u]];
    if (num >= k) return Query(lson[u], lson[v], l, m, k)
        ;
    else return Query(rson[u], rson[v], m + 1, r, k - num
        );
  }
  // 区间[u+1,v]内[s,t]数量
  int Query(int u, int v, int s, int t, int l, int r) {
    if (s <= l && t >= r) return cnt[v] - cnt[u];
    int m = (l + r) / 2, ret = 0;
    if (s <= m) ret += Query(lson[u], lson[v], s, t, l, m
        );
    if (t > m) ret += Query(rson[u], rson[v], s, t, m +
        1, r);
    return ret;
  }
};
```

3.8 dfs 序

```
const int maxn = "Edit";
std::vector<int> g[maxn];
int in[maxn], out[maxn];
int ele[maxn];
int dfs_clock;
void DfsSeq(int u, int p) {
  in[u] = ++dfs_clock;
  ele[dfs_clock] = u;
  for (int &v : g[u]) {
    if (v == p) continue;
    DfsSeq(v, u);
  }
  out[u] = dfs_clock;
}
```

3.9 ST 表

```
const int maxn = "Edit";
const int maxlog = "Edit";
int n;
int max[maxn][maxlog], min[maxn][maxlog];
void Init(int arr[]) {
```

```
  int m = log2(n) + 1;
  for (int i = 1; i <= n; ++i) max[i][0] = min[i][0] =
       arr[i];
  for (int j = 1; j < m; ++j) {
    for (int i = 1; i + (1 << j) - 1 <= n; ++i) {
      max[i][j] = std::max(max[i][j - 1], max[i + (1 << (
           j - 1))][j - 1]);
      min[i][j] = std::min(min[i][j - 1], min[i + (1 << (
           j - 1))][j - 1]);
    }
  }
}
// 区间[l,r]最大值
int QueryMax(int l, int r) {
  int k = log2(r - l + 1);
  return std::max(max[l][k], max[r - (1 << k) + 1][k]);
}
// 区间[l,r]最小值
int QueryMin(int l, int r) {
  int k = log2(r - l + 1);
  return std::min(min[l][k], min[r - (1 << k) + 1][k]);
}
```

3.10 Link Cut Tree

```
const int maxn = "Edit";
struct LCT {
  int fa[maxn], son[maxn][2];
  int val[maxn], sum[maxn];
  int rev[maxn], stk[maxn];
  void Init(int n) {
    for (int i = 1; i <= n; ++i) scanf("%d", &val[i]);
    for (int i = 1; i <= n; ++i) fa[i] = son[i][0] = son[
         i][1] = rev[i] = 0;
  }
  bool IsRoot(int o) {
    return son[fa[o]][0] != o && son[fa[o]][1] != o;
  }
  bool Get(int o) {
    return son[fa[o]][1] == o;
  }
  // 更新所需维护的信息
  void Pull(int o) {
    sum[o] = val[o] ^ sum[son[o][0]] ^ sum[son[o][1]];
  }
  void Push(int o) {
    if (rev[o] != 0) {
      std::swap(son[o][0], son[o][1]);
      if (son[o][0]) rev[son[o][0]] ^= 1;
      if (son[o][1]) rev[son[o][1]] ^= 1;
      rev[o] ^= 1;
    }
  }
  void Rotate(int o) {
    int p = fa[o], q = fa[p], ck = Get(o);
    if (!IsRoot(p)) son[q][Get(p)] = o;
    fa[o] = q;
    son[p][ck] = son[o][ck ^ 1];
    fa[son[p][ck]] = p;
    son[o][ck ^ 1] = p;
    fa[p] = o;
    Pull(p);
    Pull(o);
  }
  void Splay(int o) {
    int top = 0;
    stk[++top] = o;
    for (int i = o; !IsRoot(i); i = fa[i]) stk[++top] =
         fa[i];
    for (int i = top; i; --i) Push(stk[i]);
    for (int f = fa[o]; !IsRoot(o); Rotate(o), f = fa[o])
      if (!IsRoot(f)) Rotate(Get(o) == Get(f) ? f : o);
  }
  // 将使o成为一条实路径并在同一棵Splay内
  void Access(int o) {
    for (int p = 0; o; p = o, o = fa[o]) {
      Splay(o);
      son[o][1] = p;
      Pull(o);
    }
  }
  // 返回o所在树的根节点编号
  int Find(int o) {
    Access(o);
    Splay(o);
    while (son[o][0]) o = son[o][0];
    return o;
  }
  // 使o成为其所在树的根
  void MakeRoot(int o) {
    Access(o);
    Splay(o);
    rev[o] ^= 1;
  }
  // u,v之间连边,先判不能在同一棵树内
  void Link(int u, int v) {
    MakeRoot(u);
    fa[u] = v;
    Splay(u);
  }
  // 删除u,v之间的边
  void Cut(int u, int v) {
    MakeRoot(u);
    Access(v);
    Splay(v);
```

```
    fa[u] = son[v][0] = 0;
  }
  // o节点单点修改
  void Modify(int o, int v) {
    val[o] = v;
    Access(o);
    Splay(o);
  }
  // u,v路径信息
  int Query(int u, int v) {
    MakeRoot(v);
    Access(u);
    Splay(u);
    return sum[u];
  }
};
```

4 字符串

4.1 最小表示法

```
int minRepresent(char *s, int len) {
    int i = 0, j = 1, k = 0;
    while (i < len && j < len && k < len) {
        int t = s[(i+k) % len] - s[(j+k) % len];
        if (t == 0) k++;
        else {
            if (t < 0)  j = max(j+k+1, i+1);
            else i = max(i+k+1, j+1);
            k = 0;
        }
    }
    return min(i, j);
}
```

4.2 扩展 kmp

```
struct exKmp{
    // 字符串下标从0开始
    int nex[maxn], ex[maxn]; //模式串nex, 匹配串ex
    void getNex(char *str, int len) {
        int i = 0, j, pos;
        nex[0] = len;
        while (str[i] == str[i+1] && i+1 < len) ++i;
        nex[1] = i;
        pos = 1;
        for (int i = 2; i < len; ++i) {
            if (nex[i-pos] + i < nex[pos] + pos) nex[i] =
                  nex[i-pos];
            else {
                j = nex[pos] + pos - i;
                if (j < 0) j = 0;
                while (i+j < len && str[j] == str[j+i])
                     ++j;
                nex[i] = j;
                pos = i;
            }
        }
    }
    void getEx(char *s1, char *s2) { // s1匹配s2
        int i = 0, j, pos;
        int len1 = strlen(s1);
        int len2 = strlen(s2);
        getNex(s2, len2);
        while (s1[i] == s2[i] && i < len1 && i < len2) ++
             i;
        ex[0] = i;
```

```
        pos = 0;
        for (int i = 1; i < len1; ++i) {
            if (nex[i-pos] + i < ex[pos] + pos) ex[i] =
                 nex[i-pos];
            else {
                j = ex[pos] + pos - i;
                if (j < 0) j = 0;
                while (i+j < len1 && j < len2 && s1[i+j]
                     == s2[j]) ++j;
                ex[i] = j;
                pos = i;
            }
        }
    }
}exkmp;
```

4.3 字典树

```
struct Trie{
    int nex[maxn][26], cnt[maxn], end[maxn];
    int p, root; // root = 0
    int newnode() {
        memset(nex[p], 0, sizeof(nex[p]));
        cnt[p] = end[p] = 0;
        return p++;
    }
    void init() {
        p = 0;
        root = newnode();
    }
    void add(char *s) {
        int now = root;
        for (int i = 0; s[i]; ++i) {
            if (nex[now][s[i] - 'a'] == 0) nex[now][s[i]
                 - 'a'] = newnode();
            now = nex[now][s[i] - 'a'];
            cnt[now]++;
        }
        end[now] = 1;
    }
    int find(char *s) {
        int now = root;
        for (int i = 0; s[i]; ++i) {
            if (nex[now][s[i] - 'a'] == 0) return 0;
            now = nex[now][s[i] - 'a'];
        }
        return cnt[now];
    }
}trie;
```

4.4 回文树

```
struct Palindrome_Tree{
    int nex[maxn][26];
    int fail[maxn], cnt[maxn], num[maxn]; // num 记录每个
         节点右端点的表示回文串的个数
    int len[maxn], S[maxn];
                // cnt 记录每个节点表示的回文串出现的次
         数
    int last, n, p;
    int newnode(int l) { // 新建节点
        for (int i = 0; i < 26; ++i) nex[p][i] = 0;
        cnt[p] = num[p] = 0;
        len[p] = l;
        return p++;
    }
    void init() { // 初始化
        p = 0;
        newnode(0), newnode(-1); // 新建奇根和偶根
        last = n = 0;
        S[n] = -1;
        fail[0] = 1; // 偶根指向
    }
    int get_fail(int x) { // 求fail
        while (S[n - len[x] - 1] != S[n]) x = fail[x];
        return x;
    }
    void add(int c) { // 添加节点
        c -= 'a';
        S[++n] = c;
        int cur = get_fail(last);
        if (!nex[cur][c]) {
            int now = newnode(len[cur] + 2);
            fail[now] = nex[get_fail(fail[cur])][c];
            nex[cur][c] = now;
            num[now] = num[fail[now]] + 1;
        }
        last = nex[cur][c];
        cnt[last]++;
    }
    void count() { // 求cnt
        for (int i = p - 1; i >= 0; --i) cnt[fail[i]] +=
             cnt[i];
    }
}Tree;
```

4.5 哈希

```
struct Hash{
    long long p[maxn], hash[maxn], base = 131;
    long long getHash(int l, int r) {
```

```
        long long ans = (hash[r] - hash[l-1] * p[r-l+1])
             % mod;
        return (ans + mod) % mod;
    }
    void init(string s) {
        int n = s.size();
        p[0] = 1;
        for (int i = 1; i <= n; ++i) p[i] = p[i - 1] *
             base % mod;
        for (int i = 1; i <= n; ++i) {
            hash[i] = (hash[i - 1] * base % mod + (s[i-1]
                 - 'a' + 1)) % mod;
        }
    }
}hash;
```

4.6 后缀自动机

```
struct SAM{
    int trans[maxn<<1][26], slink[maxn<<1], maxlen[maxn
         <<1];
    // 用来求endpos
    int indegree[maxn<<1], endpos[maxn<<1], rank[maxn
         <<1], ans[maxn<<1];
    // 计算所有子串的和(0-9表示)
    long sum[maxn<<1];
    int last, now, root, len;
    inline void newnode (int v) {
        maxlen[++now] = v;
    }
    inline void extend(int c) {
        newnode(maxlen[last] + 1);
        int p = last, np = now;
        // 更新trans
        while (p && !trans[p][c]) {
            trans[p][c] = np;
            p = slink[p];
        }
        if (!p) slink[np] = root;
        else {
            int q = trans[p][c];
            if (maxlen[p] + 1 != maxlen[q]) {
                // 将q点拆出nq, 使得maxlen[p] + 1 ==
                     maxlen[q]
                newnode(maxlen[p] + 1);
                int nq = now;
                memcpy(trans[nq], trans[q], sizeof(trans[
                     q]));
                slink[nq] = slink[q];
                slink[q] = slink[np] = nq;
                while (p && trans[p][c] == q) {
                    trans[p][c] = nq;
                    p = slink[p];
                }
            }else slink[np] = q;
        }
        last = np;
        // 初始状态为可接受状态
        endpos[np] = 1;
    }
    inline void build(char *s) {
        // scanf("%s", s);
        len = strlen(s);
        root = last = now = 1;
        for (int i = 0; i < len; ++i) extend(s[i] - '0');
             // extend(s[i] - '1');
    }
    // 计算所有子串的和 (0-9表示)
    inline long getSum() {
        // 拓扑排序
        for (int i = 1; i <= now; ++i) indegree[ maxlen[i
             ] ]++;
        for (int i = 1; i <= now; ++i) indegree[i] +=
             indegree[i-1];
        for (int i = 1; i <= now; ++i) rank[ indegree[
             maxlen[i] ]-- ] = i;
        mem(endpos, 0);
        endpos[1] = 1; // 从根节点向后求有效的入度
        for (int i = 1; i <= now; ++i) {
            int x = rank[i];
            for (int j = 0; j < 10; ++j) {
                int nex = trans[x][j];
                if (!nex) continue;
                endpos[nex] += endpos[x]; // 有效入度
                long num = (sum[x] * 10 + endpos[x] * j)
                     % mod;
                sum[nex] = (sum[nex] + num) % mod; // 状
                     态转移
            }
        }
        long long ans = 0;
        for (int i = 2; i <= now; ++i) ans = (ans + sum[i
             ]) % mod;
        return ans;
    }
    inline void getEndpos() {
        // topsort
        for (int i = 1; i <= now; ++i) indegree[ maxlen[i
             ] ]++; // 统计相同度数的节点的个数
        for (int i = 1; i <= now; ++i) indegree[i] +=
             indegree[i-1];  // 统计度数小于等于 i 的节点
             的总数
        for (int i = 1; i <= now; ++i) rank[ indegree[
             maxlen[i] ]-- ] = i;  // 为每个节点编号, 节
             点度数越大编号越靠后
```

```
        // 从下往上按照slik更新
        for (int i = now; i >= 1; --i) {
            int x = rank[i];
            endpos[slink[x]] += endpos[x];
        }
    }
    // 求不同的子串种类
    inline long long all () {
        long long ans = 0;
        for (int i = root+1; i <= now; ++i) {
            ans += maxlen[i] - maxlen[ slink[i] ];
        }
        return ans;
    }
    // 长度为K的字符串有多种, 求出现次数最多的次数
    inline void get_Maxk() {
        getEndpos();
        for (int i = 1; i <= now; ++i) {
            ans[maxlen[i]] = max(ans[maxlen[i]], endpos[i
                ]);
        }
        for (int i = len; i >= 1; --i) ans[i] = max(ans[i
            ], ans[i+1]);
        for (int i = 1; i <= len; ++i) //cout << ans[i]
            << endl;
            printf("%d\n", ans[i]);
    }
}sam;
```

4.7 后缀数组

```
int cntA[maxn], cntB[maxn], A[maxn], B[maxn];
int Sa[maxn], tsa[maxn], height[maxn], Rank[maxn];
char s[maxn];
int n;
void SuffixArray () {
        for (int i = 0; i < 1000; ++i) cntA[i] = 0;
        for (int i = 1; i <= n; ++i) cntA[(int)s[i]]++;
        for (int i = 1; i < 1000; ++i) cntA[i] += cntA[i
            -1];
        for (int i = n; i >= 1; --i) Sa[ cntA[(int)s[i
            ]]-- ] = i;
        Rank[ Sa[1] ] = 1;
        for (int i = 2; i <= n; ++i) {
                Rank[Sa[i]] = Rank[Sa[i-1]];
                if (s[Sa[i]] != s[Sa[i-1]]) Rank[Sa[i
                    ]]++;
        }
        for (int l = 1; Rank[Sa[n]] < n; l <<= 1) {
                for (int i = 0; i <= n; ++i) cntA[i] = 0;
                for (int i = 0; i <= n; ++i) cntB[i] = 0;
                for (int i = 1; i <= n; ++i) {
                        cntA[ A[i] = Rank[i] ]++;
                        cntB[ B[i] = (i + l <= n) ? Rank[
                            i+l] : 0]++;
                }
                for (int i = 1; i <= n; ++i) cntB[i] +=
                    cntB[i-1];
                for (int i = n; i >= 1; --i) tsa[ cntB[B[
                    i]]-- ] = i;
                for (int i = 1; i <= n; ++i) cntA[i] +=
                    cntA[i-1];
                for (int i = n; i >= 1; --i) Sa[ cntA[A[
                    tsa[i]]]-- ] = tsa[i];
                Rank[ Sa[1] ] = 1;
                for (int i = 2; i <= n; ++i) {
                        Rank[Sa[i]] = Rank[Sa[i-1]];
                        if (A[Sa[i]] != A[Sa[i-1]] || B[
                            Sa[i]] != B[Sa[i-1]]) Rank[
                            Sa[i]]++;
                }
        }
        for (int i = 1, j = 0; i <= n; ++i) {
                if (j) --j;
                int tmp = Sa[Rank[i] - 1];
                while (i + j <= n && tmp + j <= n && s[i+
                    j] == s[tmp+j]) ++j;
                height[Rank[i]] = j;
        }
}
```

4.8 kmp

```
int a[N], b[N], Next[N];        //从a数组里匹配b数组
void get_next(int m) {  //求Next数组
        Next[0] = -1;
        int i = 0, j = -1;
        while (i < m) {
                if(j == -1 || b[i] == b[j]) {
                        Next[++i] = ++j;        //赋值
                }else {
                        j = Next[j];    //回溯
                }
        }
}
int kmp(int n, int m) {
        get_next(m);    //求Next数组
        int i = 0, j = 0;
        int ans = 0;
        while (i < n) {
                if (j == -1 || a[i] == b[j]) {  //当前匹
                    配成功进行下一个匹配
```

```
                        i++;
                        j++;
                }else {
                        j = Next[j];
                }
                if (j == m) {   //匹配成功
                        ans++;
                        j = Next[j];    //进行下一次匹配
                }
        }
        return ans;
}
```

4.9 AC 自动机

```
char s[maxn];
struct Trie{
    int nex[maxn][26], fail[maxn], end[maxn];
    int root, p;
    inline int newnode() {
        for (int i = 0; i < 26; ++i) {
            nex[p][i] = -1;
        }
        end[p++] = 0;
        return p - 1;
    }
    inline void init() {
        p = 0;
        root = newnode();
    }
    inline void insert(char s[]) {
        int now = root;
        for (int i = 0; s[i]; ++i) {
            if (nex[now][s[i]-'a'] == -1)
                nex[now][s[i]-'a'] = newnode();
            now = nex[now][s[i]-'a'];
        }
        end[now]++;
    }
    inline void build() {
        queue<int> que;
        fail[root] = root;
        for (int i = 0; i < 26; ++i) {
            if (nex[root][i] == -1)
                nex[root][i] = root;
            else {
                fail[nex[root][i]] = root;
                que.push(nex[root][i]);
            }
        }
        while (!que.empty()) {
            int now = que.front();
            que.pop();
            for (int i = 0; i < 26; ++i) {
                if (nex[now][i] == -1)
                    nex[now][i] = nex[fail[now]][i];
                else {
                    fail[nex[now][i]] = nex[fail[now]][i
                        ];
                    que.push(nex[now][i]);
                }
            }
        }
    }
    inline LL query(char s[]) {
        int now = root;
        LL cnt = 0;
        for (int i = 0; s[i]; ++i) {
            now = nex[now][s[i]-'a'];
            int tmp = now;
            while (tmp != root && end[tmp] != -1) {
                cnt += end[tmp];
                end[tmp] = -1; // 统计种类, 加速
                tmp = fail[tmp];
            }
        }
        return cnt;
    }
}ac;
```

5 图论

5.1 次小生成树

```
// Kruskal
int n, m;
struct ac{
        int u, v, w, flag;
        bool operator <(ac t) {
                return w < t.w;
        }
}g[maxn*maxn];
vector<int> son[maxn];
int pre[maxn], dis[maxn][maxn];
int find (int x) {
        return (pre[x] == x) ? x : pre[x] = find(pre[x]);
}
void Kruskal() {
        for (int i = 0; i <= n; ++i) {
                son[i].clear();
                son[i].push_back(i);
                pre[i] = i;
        }
        sort(g, g+m);
        int sum = 0;
        int cnt = 0;
        for (int i = 0; i < m; ++i) {
                if (cnt == n+1) break;
                int fx = find(g[i].u);
                int fy = find(g[i].v);
                if (fx == fy) continue;
                g[i].flag = 1;
                sum += g[i].w;
                cnt++;
                int lenx = son[fx].size();
                int leny = son[fy].size();
                if (lenx < leny) {
                        swap(lenx, leny);
                        swap(fx, fy);
                }
                // 更新两点的距离最大值
                for (int j = 0; j < lenx; ++j) {
                        for (int k = 0; k < leny; ++k) {
                                dis[son[fx][j]][son[fy][k
                                    ]] = dis[son[fy][k
                                    ]][son[fx][j]] = g[i
                                    ].w;
                        }
                }
                pre[fy] = fx;
                //合并子树
                for (int j = 0; j < leny; ++j) {
                        son[fx].push_back(son[fy][j]);
                }
                son[fy].clear();
        }
        int ans = inf;
        for (int i = 0; i < m; ++i) {
                if (g[i].flag) continue;
                ans = min(ans, sum + g[i].w - dis[g[i].u
                    ][g[i].v]);
        }
        printf("%d %d\n", sum, ans);
}
// Prim
int n, m;
int g[maxn][maxn], val[maxn], vis[maxn], dis[maxn];
int pre[maxn], maxd[maxn][maxn];
bool used[maxn][maxn];
void prim(int s) {
    mem(maxd, 0);
    mem(vis, 0);
    mem(used, 0);
    for (int i = 1; i <= n; ++i) {
        dis[i] = g[s][i];
        pre[i] = s;
    }
    vis[s] = 1;
    int sum = 0, cnt = 0;
    for (int i = 1; i < n; ++i) {
        int u = -1, MIN = inf;
        for (int j = 1; j <= n; ++j) {
            if (vis[j]) continue;
            if (MIN > dis[j]) {
                MIN = dis[j];
                u = j;
            }
        }
        if (u == -1) break;
        vis[u] = 1;
        sum += MIN;
        cnt++;
        used[pre[u]][u] = used[u][pre[u]] = 1;
        maxd[u][pre[u]] = maxd[pre[u]][u] = MIN;
        for (int j = 1; j <= n; ++j) {
            if (j == u) continue;
            if (vis[j]) {
                maxd[u][j] = maxd[j][u] = max(maxd[pre[u
                    ]][j], MIN);
            }
            if (vis[j] == 0 && dis[j] > g[u][j]) {
                dis[j] = g[u][j];
                pre[j] = u;
            }
        }
    }
```

```
    if (cnt != n-1) {
        puts("No way");
    }
    int ans = inf;
    for (int i = 1; i <= n; ++i) {
        for (int j = i+1; j <= n; ++j) {
            if (used[i][j]) continue;
            ans = min(ans, sum + g[i][j] - maxd[i][j]);
        }
    }
    printf("%d %d\n", sum, ans);
}
```

5.2 最小树形图

```
struct ac{
    int u, v, w;
};
vector<ac> g(maxn);
int pre[maxn], vis[maxn], id[maxn], in[maxn];
int zhuliu(int rt, int n, int m) {
    int ans = 0, u, v, w;
    while (1) {
        for (int i = 0; i < n; ++i) in[i] = inf;
        for (int i = 0; i < m; ++i) {
            u = g[i].u; v = g[i].v; w = g[i].w;
            if (u != v && w < in[v]) {
                pre[v] = u;
                in[v] = w;
                // if (u == rt) pos = i; // 记录前驱，输
                      出序号最小的根
            }
        }
        for (int i = 0; i < n; ++i) {
            if (i != rt && in[i] == inf) return -1;
        }
        int cnt = 0;
        mem(id, -1);
        mem(vis, -1);
        in[rt] = 0;
        for (int i = 0; i < n; ++i) {
            ans += in[i];
            u = i;
            while (vis[u] != i && id[u] == -1 && u != rt)
                  {
                vis[u] = i;
                u = pre[u];
            }
            if (u != rt &&id[u] == -1) {
                v = pre[u];
                while (v != u) {
                    id[v] = cnt;
                    v = pre[v];
                }
                id[u] = cnt++;
            }
        }
        if (cnt == 0) break;
        for (int i = 0; i < n; ++i) {
            if (id[i] == -1) id[i] = cnt++;
        }
        for (int i = 0; i < m; ++i) {
            v = g[i].v;
            g[i].u = id[g[i].u];
            g[i].v = id[g[i].v];
            if (g[i].u != g[i].v) g[i].w -= in[v];
        }
        n = cnt;
        rt = id[rt];
    }
    return ans;
}
```

5.3 Tarjan

```
int Stack[maxn], low[maxn], dfn[maxn], inStack[maxn],
     belong[maxn];
int now, len, cnt;
// now:时间戳，len：栈的大小，cnt强连通的个数
void init() {
        now = len = cnt = 0;
        mem(inStack, 0);
        mem(belong, 0);
        mem(dfn, 0);
        mem(low, 0);
}
void tarjan(int x) {
    // 打上标记，入栈
    low[x] = dfn[x] = ++now;
    Stack[++len] = x;
    inStack[x] = 1;
    for (int i = 0; i < (int)g[x].size(); ++i) {
        int y = g[x][i];
                // 没有访问过，继续递归
                // 在栈中表示可以形成一个强连通分量，更新
                      根节点的low，继续找
        if (!dfn[y]) tarjan(y), low[x] = min(low[x], low[
             y]);
        else if (inStack[y]) low[x] = min(low[x], low[y])
             ;
    }
```

```
        // 回溯，如果当前节点的dfn = low 表示栈中形成一个
            强连通分量
    if (dfn[x] == low[x]) {
        ++cnt; // 统计个数
        int top;
        while (Stack[len] != x) {
            top = Stack[len--];
            belong[top] = cnt;
            inStack[top] = 0;
        }
        top = Stack[len--];
        belong[top] = cnt; // 记录每个点的隶属关系
        inStack[top] = 0;
    }
}
for (int i = 1; i <= n; ++i) {
    if (!dfn[i])  tarjan(i);
}
```

5.4 Dinic

```
struct ac{
    int v, c, pre;
}edge[maxn<<6];
int s, e;
int head[maxn<<1], dis[maxn<<1], curedge[maxn<<1], cnt;
void init() {
    mem(head, -1);
    cnt = 0;
}
void addedge(int u, int v, int c) { // 记得双向边
    edge[cnt] = {v, c, head[u]};
    head[u] = cnt++;
}
bool bfs() {
    queue<int> que;
    que.push(s);
    mem(dis, 0);
    dis[s] = 1;
    while (!que.empty()) {
        int f = que.front();
        que.pop();
        for (int i = head[f]; i != -1; i = edge[i].pre) {
            if (dis[edge[i].v] || edge[i].c == 0)
                continue;
            dis[edge[i].v] = dis[f] + 1;
            que.push(edge[i].v);
        }
    }
    return dis[e] > 0;
}
```

```
int dfs(int now, int flow) {
    if (now == e || flow == 0) return flow;
    for (int &i = curedge[now]; i != -1; i = edge[i].pre)
        { // 当前弧优化
        if (dis[edge[i].v] != dis[now] + 1 || edge[i].c
            == 0) continue;
        int d = dfs(edge[i].v, min(flow, edge[i].c));
        if (d > 0) {
            edge[i].c -= d;
            edge[i^1].c += d;
            return d;
        }
    }
    dis[now] = -1; // // 炸点优化
    return 0;
}
int Dinic() {
    int sum = 0, d;
    while (bfs()) {
        for (int i = 0; i <= e; ++i) curedge[i] = head[i
            ];
        while (d = dfs(s, inf)) sum += d;
    }
    return sum;
}
```

6 其它

6.1 闰年

```
bool IsLeepYear(int y) {
  return (!(y % 4) && (y % 100)) || !(y % 400);
}
```

6.2 蔡勒公式

```
// 返回y年m月d日是星期几
int Zeller(int y, int m, int d) {
  if (m == 1 || m == 2) {
    --y;
    m += 12;
  }
  int c = y / 100;
  y %= 100;
  //1582年10月4日之前
  return ((y + y / 4 + c / 4 - 2 * c + 13 * (m + 1) / 5 +
       d + 2) % 7) + 7 % 7;
  //1582年10月4日之后
  return ((y + y / 4 + c / 4 - 2 * c + 26 * (m + 1) / 10
       + d - 1) % 7 + 7) % 7;
}
```

6.3 莫队算法

6.3.1 静态莫队

```
const int maxn = "Edit";
// 静态莫队算法求区间不同数字数量
struct MoCap {
  int n, m;
  int block;
  int arr[maxn];
  struct query { int l, r, id; };
  query q[maxn];
  int cnt[maxn << 1];
  int cur;
  int ans[maxn];
  void Add(int x) {
    cur += (++cnt[arr[x]] == 1);
  }
  void Del(int x) {
    cur -= (--cnt[arr[x]] == 0);
  }
  void Solve() {
    scanf("%d%d", &n, &m);
    block = std::sqrt(n);
    for (int i = 1; i <= n; ++i) scanf("%d%d", &arr[i]);
    for (int i = 1; i <= m; ++i) {
      scanf("%d%d", &q[i].l, &q[i].r);
      q[i].id = i;
    }
    std::sort(q + 1, q + m + 1, [&](query k1, query k2) {
          return (k1.l / block) == (k2.l / block) ? k1.r
         < k2.r : k1.l < k2.l; });
    int l = 0, r = 0;
    for (int i = 1; i <= m; ++i) {
      while (l < q[i].l) Del(l++);
      while (l > q[i].l) Add(--l);
      while (r < q[i].r) Add(++r);
      while (r > q[i].r) Del(r--);
      ans[q[i].id] = cur;
    }
    for (int i = 1; i <= m; ++i) printf("%d\n", ans[i]);
  }
}mo;
```

6.3.2 带修莫队

```
const int maxn = "Edit";
// 动态莫队算法求区间不同数字数量（支持单点修改）
struct MoCap {
  int n, m;
  int block;
  int arr[maxn];
  struct query { int l, r, pre, id; };
  int q_tot;
  query q[maxn];
  struct change { int pos, val; };
  int c_tot;
  change c[maxn];
  int cnt[maxn << 7];
  int cur;
  int ans[maxn];
  void Add(int x) {
    cur += (++cnt[arr[x]] == 1);
  }
  void Del(int x) {
    cur -= (--cnt[arr[x]] == 0);
  }
  void Modify(int x, int i) {
    if (c[x].pos >= q[i].l && c[x].pos <= q[i].r) {
      cur -= (--cnt[arr[c[x].pos]] == 0);
      cur += (++cnt[c[x].val] == 1);
    }
    std::swap(c[x].val, arr[c[x].pos]);
```

```
  }
  void Solve() {
    scanf("%d%d", &n, &m);
    block = std::sqrt(n);
    for (int i = 1; i <= n; ++i) scanf("%d", &arr[i]);
    for (int i = 1; i <= m; ++i) {
      char op; getchar();
      scanf("%c", &op);
      if (op == 'Q') {
        int l, r; scanf("%d%d", &l, &r);
        q[++q_tot] = (query){l, r, c_tot, q_tot};
      }
      else {
        int p, v; scanf("%d%d", &p, &v);
        c[++c_tot] = (change){p, v};
      }
    }
    std::sort(q + 1, q + q_tot + 1, [&](query k1, query
         k2) {
      if ((k1.l / block) == k2.l / block) {
        if ((k1.r / block) == (k2.r / block)) return k1.
             pre < k2.pre;
        return k1.r < k2.r;
      }
      return k1.l < k2.l;
    });
    int l = 1, r = 0, t = 0;
    for (int i = 1; i <= q_tot; ++i) {
      while (l < q[i].l) Del(l++);
      while (l > q[i].l) Add(--l);
      while (r < q[i].r) Add(++r);
      while (r > q[i].r) Del(r--);
      while (t < q[i].pre) Modify(++t, i);
      while (t > q[i].pre) Modify(t--, i);
      ans[q[i].id] = cur;
    }
    for (int i = 1; i <= q_tot; ++i) printf("%d\n", ans[i
         ]);
  }
}mo;
```

6.4 快读

```
// 普通快读
template <typename t>
inline bool Read(t &ret) {
  char c; int sgn;
  if (c = getchar(), c == EOF) return false;
  while (c != '-' && (c < '0' || c > '9')) c = getchar();
  sgn = (c == '-') ? -1 : 1;
  ret = (c == '-') ? 0 : (c - '0');
  while (c = getchar(), c >= '0' && c <= '9') ret = ret *
       10 + (c - '0');
  ret *= sgn;
  return true;
}
// 牛逼快读
namespace FastIO {
  const int MX = 4e7;
  char buf[MX];
  int c, sz;
  void Begin() {
    c = 0;
    sz = fread(buf, 1, MX, stdin);
  }
  template <class T>
  inline bool Read(T &t) {
    while (c < sz && buf[c] != '-' && (buf[c] < '0' ||
         buf[c] > '9')) c++;
    if (c >= sz) return false;
    bool flag = 0;
    if (buf[c] == '-') {
      flag = 1;
      c++;
    }
    for (t = 0; c < sz && '0' <= buf[c] && buf[c] <= '9';
         ++c) t = t * 10 + buf[c] - '0';
    if (flag) t = -t;
    return true;
  }
};
using namespace FastIO;
```

6.5 对拍

```
// windows
:loop
data.exe > in.txt
main.exe < in.txt > out.txt
std.exe < in.txt > std.txt
fc out.txt std.txt
if not errorlevel 1 goto loop
pause
:end
// Linux
declare -i n=1
while (true)
  do
  ./dtmk
  ./my < 1.in > my.out
  ./force < 1.in > for.out
  if diff my.out for.out
```

```
  then
    echo right $n
    n=n+1
  else
    exit
  fi
done
```

6.6 vimrc

```
syntax on
set nu ts=2 sw=2 et mouse=a cindent
"map <F9> :call Run()<CR>
"func! Run()
"     exec "W"
"     exec "!g++ % -o %<"
"     exec "! %<"
"endfunc
"map <F2> :call SetTitle()<CR>
"func SetTitle()
"     let l = 0
"     let l = l + 1 | call setline(l, "#include <bits/stdc
      ++.h>")
"     let l = l + 1 | call setline(l, "")
"     let l = l + 1 | call setline(l, "int main() {")
"     let l = l + 1 | call setline(l, "  return 0;")
"     let l = l + 1 | call setline(l, "}")
"     let l = l + 1 | call setline(l, "")
"endfunc
```