Algorithm Library

Tony5t4rk

May 23, 2019

Contents

1 String

1.1 AhoCorasickAutomaton

```
const int maxn = "Edit";

struct AhoCorasickAutomaton {
  int son[maxn][26];
  int val[maxn];
  int fail[maxn];
  int tot;

  // Trie Tree 初始化
  void TrieInit() {
    tot = 0;
    memset(son, 0, sizeof(son));
    memset(val, 0, sizeof(val));
    memset(fail, 0, sizeof(fail));
  }

  // 计算字母下标
  int Pos(char x) { return x - 'a'; }

  // 向 Trie Tree 中插入 str 模式字符串
  void Insert(std::string str) {
    int cur = 0, Len = (int)str.length();
    for (int i = 0; i < Len; ++i) {
      int idx = Pos(str[i]);
      if (!son[cur][idx]) son[cur][idx] = ++tot;
      cur = son[cur][idx];
    }
    val[cur]++;
  }

  // Bfs 求得 Trie Tree 上失配指针
  void GetFail() {
    std::queue<int> que;
    for (int i = 0; i < 26; ++i) {
      if (son[0][i]) {
        fail[son[0][1]] = 0;
        que.push(son[0][i]);
      }
    }
    while (!que.empty()) {
      int cur = que.front(); que.pop();
      for (int i = 0; i < 26; ++i) {
        if (son[cur][i]) {
          fail[son[cur][i]] = son[fail[cur]][i];
          que.push(son[cur][i]);
        }
        else son[cur][i] = son[fail[cur]][i];
      }
    }
  }

  // 询问 str 中出现的模式串数量
  int Query(std::string str) {
    int len = (int)str.length();
    int cur = 0, ret = 0;
    for (int i = 0; i < len; ++i) {
      cur = son[cur][Pos(str[i])];
      for (int j = cur; j && ~val[j]; j = fail[j]) {
        ret += val[j];
        val[j] = -1;
      }
    }
    return ret;
  }
};
```

1.2 KMP

```
// 对模式串 pattern 计算 next 数组
void KMPPre(std::string pattern, std::vector<int> &next) {
  int i = 0, j = -1;
  next[0] = -1;
  int len = (int)pattern.length();
  while (i != len) {
    if (j == -1 || pattern[i] == pattern[j]) next[++i] = ++j;
    else j = next[j];
  }
}

// 优化对模式串 pattern 计算 next 数组
void PreKMP(std::string pattern, std::vector<int> &next) {
  int i, j;
  i = 0;
  j = next[0] = -1;
  int len = (int)pattern.length();
  while (i < len) {
    while (j != -1 && pattern[i] != pattern[j]) j = next[j];
    if (pattern[++i] == pattern[++j]) next[i] = next[j];
    else next[i] = j;
  }
}

// 利用预处理 next 数组计数模式串 pattern 在主串 main 中出现次数
int KMPCount(std::string pattern, std::string main) {
  int pattern_len = (int)pattern.length(), main_len = (int)main.length();
  std::vector<int> next(pattern_len + 1, 0);
  //PreKMP(pattern, next);
  KMPPre(pattern, next);
  int i = 0, j = 0;
  int ret = 0;
  while (i < main_len) {
    while (j != -1 && main[i] != pattern[j]) j = next[j];
    i++; j++;
    if (j >= pattern_len) {
      ret++;
      j = next[j];
    }
  }
  return ret;
}
```

1.3 Manacher

```
const int maxn = "Edit";

char convert_str[maxn << 1];
int len[maxn << 1];

// Manacher 算法求 str 字符串最长回文子串长度
int Manacher(char Str[]) {
  int L = 0, str_len = (int)strlen(Str);
  convert_str[L++] = '$'; convert_str[L++] = '#';
  for (int i = 0; i < str_len; ++i) {
    convert_str[L++] = Str[i];
    convert_str[L++] = '#';
  }
  int mx = 0, id = 0, ret = 0;
  for (int i = 0; i < L; ++i) {
    len[i] = mx > i ? std::min(len[2 * id - i], mx - i) : 1;
    while (convert_str[i + len[i]] == convert_str[i - len[i]]) len[i]++;
    if (i + len[i] > mx) {
      mx = i + len[i];
      id = i;
    }
    ret = std::max(ret, len[i] - 1);
  }
  return ret;
}
```

1.4 PalindromicTree

```
const int maxn = "Edit";

struct PalindromicTree {
  // 子节点记录数组
  int son[maxn][26];
```

```
// 失配指针 fail 数组
int fail[maxn];
// len[i]: 节点 i 表示的回文串长度 (一个节点表示一个回文串)
int len[maxn];
// cnt[i]: 节点 i 表示的本质不同的串的个数 (最后需要运行 Count() 函数才可求出正确结
↪  果)
int cnt[maxn];
// num[i]: 以节点 i 表示的最长回文串的最右端为回文串结尾的回文串个数
int num[maxn];
// 字符
int str[maxn];
// 新添加字符后最长回文串表示的节点
int last;
// 字符数量
int str_len;
// 节点数量
int tot;

// 新建节点
int NewNode(int x) {
  for (int i = 0; i < 26; ++i) son[tot][i] = 0;
  cnt[tot] = 0;
  num[tot] = 0;
  len[tot] = x;
  return tot++;
}

// 初始化
void Init() {
  tot = 0;
  NewNode(0); NewNode(-1);
  last = 0;
  str_len = 0;
  // 开头存字符集中没有的字符, 减少特判
  str[0] = -1;
  fail[0] = 1;
}

int GetFail(int x) {
  while (str[str_len - len[x] - 1] != str[str_len]) x = fail[x];
  return x;
}

void Add(int c) {
  c -= 'a';
  Str[++str_len] = c;
  int Cur = GetFail(last);
  if (!son[Cur][c]) {
    int New = NewNode(len[Cur] + 2);
```

```
      fail[New] = son[GetFail(fail[Cur])][c];
      son[Cur][c] = New;
      num[New] = num[fail[New]] + 1;
    }
    last = son[Cur][c];
    cnt[last]++;
  }

  void Count() {
    // 若 fail[v]=u, 则 u 一定是 v 回文子串, 所以双亲累加孩子的 cnt
    for (int i = tot - 1; i >= 0; --i) cnt[fail[i]] += cnt[i];
  }
};
```

1.5 Trie

```
const int maxn = "Edit";

struct Trie {
  int son[maxn][26];
  int tot;
  int cnt[maxn];

  void TrieInit() {
    tot = 0;
    memset(cnt, 0, sizeof(cnt));
    memset(son, 0, sizeof(son));
  }

  int Pos(char x) { return x - 'a'; }

  // 向 Trie Tree 中插入字符串 str
  void Insert(std::string str) {
    int cur = 0, len = (int)str.length();
    for (int i = 0; i < len; ++i) {
      int idx = Pos(str[i]);
      if (!son[cur][idx]) son[cur][idx] = ++tot;
      cur = son[cur][idx];
      cnt[cur]++;
    }
  }

  // 查找字符串 str, 存在返回 true, 不存在返回 false
  bool Find(std::string str) {
    int cur = 0, len = (int)str.length();
    for (int i = 0; i < len; ++i) {
      int idx = Pos(str[i]);
      if (!son[cur][idx]) return false;
      cur = son[cur][idx];
    }
    return true;
  }

  // 查询字典树中以 str 为前缀的字符串数量
  int PathCnt(std::string str) {
    int cur = 0, len = (int)str.length();
    for (int i = 0; i < len; ++i) {
      int idx = Pos(Str[i]);
      if (!son[cur][idx]) return 0;
      cur = son[cur][idx];
    }
    return cnt[cur];
  }
};
```

2 Math

2.1 Catalan

```
const int maxn = "Edit";

long long cat[maxn];

void GetCat() {
  memset(cat, 0, sizeof(cat));
  cat[0] = cat[1] = 1;
  for (int i = 2; i < maxn; ++i) cat[i] = cat[i - 1] * (4 * i - 2) / (i + 1);
}
```

2.2 CombinatorialNumber

2.2.1 CombinatorialNumber

```
const int mod = "Edit";
const int maxn  = "Edit";

int c[maxn][maxn];

void GetC() {
  memset(c, 0, sizeof(c));
  c[0][0] = 1;
  for (int i = 1; i < maxn; ++i) {
    c[i][0] = 1;
    for (int j = 1; j <= i; ++j) {
      if (j == i) c[i][j] = 1;
      else c[i][j] = (c[i - 1][j - 1] + c[i - 1][j]) % mod;
    }
  }
}
```

2.2.2 Lucas

```
const int mod = "Edit";

long long fac[maxn], facinv[maxn];

void GetFacInv() {
  fac[0] = 0; fac[1] = 1;
  for (int i = 2; i < maxn; ++i) fac[i] = (fac[i - 1] * i) % mod;
  facinv[maxn - 1] = Pow(fac[maxn - 1], mod - 2);
  for (int i = maxn - 2; i >= 0; --i) facinv[i] = (facinv[i + 1] * (i + 1)) % mod;
}

long long Lucas(long long n, long long m) {
  long long ret = 1;
  while (n && m) {
    long long a = n % mod, b = m % mod;
    if (a < b) return 0;
    ret = ret * fac[a] % mod * facinv[b] % mod * facinv[a - b] % mod;
    n /= mod, m /= mod;
  }
  return ret;
}
```

2.3 Derangement

```
const int maxn = "Edit";
const int mod = "Edit";

long long stag[maxn];

// 错排
void GetStag() {
  stag[1] = 0; stag[2] = 1;
  for (int i = 3; i < maxn; ++i) stag[i] = (i - 1) * (stag[i - 1] + stag[i - 2]) %
   ↪  mod;
}
```

2.4 Euler

2.4.1 Euler

```
int GetPhi(int x) {
  int ret = x;
  for (int i = 2; i * i <= x; ++i) {
    if (!(x % i)) {
      ret = ret / i * (i - 1);
      while (!(x % i)) x /= i;
    }
  }
  if (x > 1) ret = ret / x * (x - 1);
  return ret;
}
```

2.4.2 Screen

```
const int maxn = "Edit";

int phi[maxn];

void GetPhi() {
  for (int i = 1; i < maxn; ++i) phi[i] = i;
  for (int i = 2; i < maxn; i += 2) phi[i] /= 2;
  for (int i = 3; i < maxn; i += 2)
    if (phi[i] == i)
      for (int j = i; j < maxn; j += i) phi[j] = phi[j] / i * (i - 1);
}
```

2.4.3 Sieve

```
const int maxn = "Edit";

bool is_prime[maxn];
int phi[maxn];
std::vector<int> prime;

void Sieve() {
  memset(is_prime, true, sizeof(is_prime));
  phi[1] = 1; is_prime[0] = is_prime[1] = false;
  for (long long i = 2; i < maxn; ++i) {
    if (is_prime[i]) {
      phi[i] = i - 1;
      prime.emplace_back(i);
    }
    for (auto &p : prime) {
      if (p * i >= maxn) break;
      is_prime[i * p] = false;
      if (i % p == 0) {
        phi[i * p] = phi[i] * p;
        break;
      }
      phi[i * p] = phi[i] * phi[p];
    }
  }
}
```

2.5 FFT

```
const int maxn = "Edit";
const double pi = acos(-1.0);

// 复数
struct complex {
  double x, y;
  complex operator + (const complex &b) const {return complex {x + b.x, y + b.y};}
  complex operator - (const complex &b) const {return complex {x - b.x, y - b.y};}
  complex operator * (const complex &b) const {return complex {x * b.x - y * b.y, x
  ↪  * b.y + y * b.x};}
  complex operator / (const complex &b) const {
    double tmp = b.x * b.x + b.y * b.y;
    return complex {(x * b.x + y * b.y) / tmp, (y * b.x - x * b.y) / tmp};
  }
};

// 多项式系数数量
int n, m;
int l;
int limit;
int r[maxn << 2];

// 快速傅里叶变换 (FFT)
void FFT(complex f[], int op) {
  for (int i = 0; i < limit; ++i) {
    if (i < r[i]) std::swap(f[i], f[r[i]]);
  }
  for (int j = 1; j < limit; j <<= 1) {
    complex tmp = complex {cos(pi / j), op * sin(pi / j)};
    for (int k = 0; k < limit; k += (j << 1)) {
      complex Buffer = complex {1.0, 0.0};
      for (int l = 0; l < j; ++l) {
        complex tx = f[k + l], ty = Buffer * f[k + j + l];
        f[k + l] = tx + ty;
        f[k + j + l] = tx - ty;
        Buffer = Buffer * tmp;
      }
    }
  }
}

complex a[maxn], b[maxn];

// 多项式卷积计算
void Cal() {
  limit = 1; l = 0;
  while (limit <= n + m) {
    limit <<= 1;
    l++;
  }
  for (int i = 0; i < limit; ++i) r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));
  FFT(a, 1);
  FFT(b, 1);
  for (int i = 0; i <= limit; ++i) a[i] = a[i] * b[i];
  FFT(a, -1);
}
```

2.6 Gauss

```
const int mod = "Edit";

void Gauss(std::vector<std::vector<long long>> &matrix) {
  int n = (int)matrix.size();
  for (int i = 0; i < n; ++i) {
    long long inv = Inv(matrix[i][i]);
    for (int j = i; j <= n; ++j) {
      matrix[i][j] = matrix[i][j] * inv % mod;
    }
    for (int j = 0; j < n; ++j) {
      if (j != i) {
        long long tmp = matrix[j][i];
        for (int k = i; k <= n; ++k) {
          matrix[j][k] = (matrix[j][k] - matrix[i][k] * tmp % mod + mod) % mod;
        }
      }
    }
  }
}
```

2.7 GeneratingFunction

```
const int maxn = "Edit";

int c1[maxn], c2[maxn];

void GetGeneratingFunction(int n) {
  for (int i = 0; i < maxn; ++i) {
    c1[i] = 1;
    c2[i] = 0;
  }
  // c1[i] 为 x^i 的系数
  // c2 为中间变量
  for (int i = 2; i <= n; ++i) {
    for (int j = 0; j <= n; ++j) {
      for (int k = 0; k + j <= n; k += i) {
        c2[j + k] += c1[i];
      }
    }
    for (int j = 0; j <= n; ++j) {
      c1[j] = c2[j];
      c2[j] = 0;
    }
  }
}
```

2.8 InverseElement

2.8.1 ExtendGcd

```
const int mod = "Edit";

// 扩展欧几里得, a*x+b*y=d
long long ExtendGcd(long long a, long long b, long long &x, long long &y) {
  if (a == 0 && b == 0) return -1;
  if (b == 0) {
    x = 1;
    y = 0;
    return a;
  }
  long long d = ExtendGcd(b, a % b, y, x);
  y -= a / b * x;
  return d;
}

// 逆元, ax = 1(mod mod)
long long GetInv(long long a) {
  long long x, y;
  long long d = ExtendGcd(a, mod, x, y);
  if (d == 1) return (x % mod + mod) % mod;
  else return -1;
}
```

2.8.2 Factorial

```
const int mod = "Edit";
const int maxn = "Edit";

// fac: 阶乘, facinv: 阶乘逆元
long long fac[maxn], facinv[maxn];

void GetFacInv() {
  fac[0] = 0; fac[1] = 1;
  for (int i = 2; i < maxn; ++i) fac[i] = (fac[i - 1] * i) % mod;
  facinv[maxn - 1] = Pow(fac[maxn - 1], mod - 2);
  for (int i = maxn - 2; i >= 0; --i) facinv[i] = (facinv[i + 1] * (i + 1)) % mod;
}
```

2.8.3 FermatLittleTheorem

```
const int mod = "Edit";

long long Inv(long long x) {
  return Pow(x, mod - 2);
}
```

2.8.4 Recursive

```
const int mod = "Edit";
const int maxn = "Edit";

long long inv[maxn];

// 递推求逆元
void GetInv() {
  inv[1] = 1;
  for (int i = 2; i < maxn; ++i) inv[i] = (mod - mod / i) * inv[mod % i] % mod;
}
```

2.9 Josephus

```
// n 个人 1~n, 第 k 个人出列, 返回第 m(<=n) 个出列的人编号
long long Josephus(long long n, long long k, long long m) {
  if (k == 1) return m;
  long long x = (k - 1) % (n + 1 - m);
  for (long long i = n + 1 - m; i < n; ) {
    long long y = std::min((i - x + k - 2) / (k - 1) - 1, n - i);
    if (y) {
      x -= y * k;
      i += y;
    }
    else {
      ++i;
      x = (x + k) % i;
    }
  }
  return x + 1;
}
```

2.10 Matrix

```
const int maxn = "Edit";
const int mod = "Edit";

struct matrix {
  long long mat[maxn][maxn];
  matrix() { memset(mat, 0, sizeof(mat)); }
  void Unit() { for (int i = 0; i < maxn; ++i) mat[i][i] = 1; }
};

matrix operator * (matrix k1, matrix k2) {
  matrix ret;
  for (int i = 0; i < maxn; ++i) {
    for (int j = 0; j < maxn; ++j) {
      for (int k = 0; k < maxn; ++k) {
        ret.mat[i][j] = (ret.mat[i][j] + k1.mat[i][k] * k2.mat[k][j]) % mod;
      }
    }
  }
  return ret;
}

matrix Pow(matrix x, long long n) {
  matrix ret;
  ret.Unit();
  while (n) {
    if (n & 1) ret = ret * x;
    x = x * x;
    n >>= 1;
  }
  return ret;
}
```

2.11 Mobius

```cpp
#include <bits/stdc++.h>

const int maxn = "Edit";

bool is_prime[maxn];
std::vector<int> prime;
int mu[maxn];

void Sieve() {
  memset(is_prime, true, sizeof(is_prime));
  mu[1] = 1; is_prime[0] = is_prime[1] = false;
  for (int i = 2; i < maxn; ++i) {
    if (is_prime[i]) {
      prime.emplace_back(i);
      mu[i] = -1;
    }
    for (auto &p : prime) {
      if (p * i >= maxn) break;
      is_prime[i * p] = false;
      if (i % p == 0) {
        mu[i * p] = 0;
        break;
      }
      mu[i * p] = -mu[i];
    }
  }
}
```

2.12 NimGame

```cpp
// ret 不为零则先手赢，否则为后手赢
bool GetNim(std::vector<int> arr) {
  int ret = 0;
  for (auto &v : arr) ret ^= v;
  return ret != 0;
}
```

2.13 Polynomial

```cpp
const int mod = "Edit";

// 多项式求值（低次在前）
long long F(long long x, std::vector<long long> &coef) {
  long long ret = 0;
  for (int i = (int)coef.size() - 1; ~i; --i) {
    ret = (ret * x + coef[i]) % mod;
  }
  return ret;
}
```

2.14 Prime

2.14.1 PrimeFactor

```
const int maxn = "Edit"

bool is_prime[maxn];
vector<int> prime_fac[maxn];

void GetPrimeFac() {
  memset(is_prime, true, sizeof(is_prime));
  for (long long i = 2; i < maxn; ++i) {
    if (is_prime[i]) {
      prime_fac[i].push_back(i);
      for (long long j = i + i; j < maxn; ++j) {
        is_prime[j] = false;
        prime_fac[j].push_back(i);
      }
    }
  }
  is_prime[1] = false;
}
```

2.14.2 SieveOfEratosthenes

```
const int maxn = "Edit";

bool is_prime[maxn];
std::vector<int> prime

void Sieve() {
  memset(is_prime, true, sizeof(is_prime));
  is_prime[0] = is_prime[1] = false;
  for (long long i = 2; i < maxn; ++i) {
    if (is_prime[i]) prime.emplace_back(i);
    for (auto &p : prime) {
      if (p * i >= maxn) break;
      is_prime[i * p] = false;
    }
  }
}
```

2.15 QuickPow

```
const int mod = "edit";

long long Mul(long long x, long long y) {
  long long ret = 0;
  while (y) {
    if (y) ret = (ret + x) % mod;
    x = (x + x) % mod;
    y >>= 1;
  }
  return ret;
}

long long Pow(long long x, long long n) {
  long long ret = 1;
  while (n) {
    if (n & 1) ret = (ret * x) % mod;
    x = x * x % mod;
    n >>= 1;
  }
  return ret;
}
```

2.16 Stirling

```
const double pi = acos(-1.0);
const double e = 2.718281828459;

int GetStirling(int x) {
  if (x <= 1) return 1;
  return (int)ceil(log10(2 * pi * x) / 2 + x * log10(x / e));
}
```

3 DataStructure

3.1 BinaryIndexedTree

```
#define lowbit(x) (x&(-x))
const int maxn = "Edit";

struct BitTree {
  int arr[maxn];

  void Init() { memset(arr, 0, sizeof(0)); }

  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;
  }

  // min
  int GetKth(int k) {
    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); }
};
```

3.2 DfsOrder

```
std::vector<std::vector<int>> g;
int dfs_clock;
std::vector<int> in, out;

// Dfs 序
void DfsOrder(int cur, int pre) {
  in[cur] = ++dfs_clock;
  for (auto &it : g) {
    if (it == pre) continue;
    DfsOrder(it, cur);
  }
  out[cur] = dfs_clock;
}
```

3.3 FunctionalSegmentTree

```
const int maxn = "Edit";

struct FuncSegTree {
  int tot;
  int rt[maxn];
  int lson[maxn << 5], rson[maxn << 5];
  int cnt[maxn << 5];

  int Build(int l, int r) {
    int o = ++tot, m = (l + r) >> 1;
    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);
    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) >> 1, 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;
  }
};
```

3.4 Hash

```
template <typename type>
struct Hash {
    int size;
    vector<int> arr;

    Hash(const vector<type> &v) {
      arr.assign(v.begin(), v.end());
      sort(arr.begin(), arr.end());
      arr.erase(unique(arr.begin(), arr.end()), arr.end());
      size = arr.size();
    }

    int Get(type k) {
      return lower_bound(arr.begin(), arr.end(), k) - arr.begin();
    }
};
```

3.5 LCA

3.5.1 DFS+ST

```
const int maxn = "Edit";

struct edge { int v, c, next; };

edge edges[maxn << 1];
int head[maxn];
int tot;
```

```
void AddEdge(int u, int v, int c) {
  edges[tot] = (edge){v, c, head[u]};
  head[u] = tot++;
}

// 节点深度
int rmq[maxn << 1];
// 深搜遍历顺序
int vertex[maxn << 1];
// 节点在深搜中第一次出现的位置
int first[maxn];
int fa[maxn];
int dis[maxn];
int lca_tot;

// 最小值对应下标
int dp[maxn << 1][20];

// rmq 初始化
void Work(int n) {
  for (int i = 1; i <= n; ++i) dp[i][0] = i;
  for (int j = 1; (1 << j) <= n; ++j) {
    for (int i = 1; i + (1 << j) - 1 <= n; ++i) {
      dp[i][j] = rmq[dp[i][j - 1]] < rmq[dp[i + (1 << (j - 1))][j - 1]] ? dp[i][j -
       ↪  1] : dp[i + (1 << (j - 1))][j - 1];
    }
  }
}

// 深搜
void Dfs(int cur, int pre, int dep) {
  vertex[++lca_tot] = cur;
  first[cur] = lca_tot;
  rmq[lca_tot] = dep;
  fa[cur] = pre;
  for (int i = head[cur]; ~i; i = edges[i].next) {
    if (edges[i].v == pre) continue;
    dis[edges[i].v] = dis[cur] + edges[i].c;
    Dfs(edges[i].v, cur, dep + 1);
    vertex[++lca_tot] = cur;
    rmq[lca_tot] = dep;
  }
}

// rmq 查询
int Query(int l, int r) {
  if (l > r) swap(l, r);
  int len = (int)log2(r - l + 1);
  return rmq[dp[l][len]] <= rmq[dp[r - (1 << len) + 1][len]] ? dp[l][len] : dp[r -
   ↪  (1 << len) + 1][len];
}

// LCA 初始化
void Init(int rt, int num) {
  memset(dis, 0, sizeof(dis));
  lca_tot = 0;
  Dfs(rt, 0, 0);
  fa[1] = 0;
  Work(2 * num - 1);
}

// 查询节点 u、v 的距离
int GetDis(int u, int v) { return dis[u] + dis[v] - 2 * dis[LCA(u, v)]; }

// 查询节点 u、v 的最近公共祖先 (LCA)
int GetLCA(int u, int v) { return vertex[Query(first[u], first[v])]; }
```

3.5.2 Multiplication

```
const int maxn = "Edit";

int n, k;
std::vector<int> g[maxn];

void AddEdge(int u, int v) {
  g[u].push_back(v);
  g[v].push_back(u);
}

int anc[maxn][25];
int dep[maxn];

void Dfs(int u, int prev, int depth) {
  anc[u][0] = prev; dep[u] = depth;
  for (auto &v : g[u]) {
    if (v == prev) continue;
    Dfs(v, u, depth + 1);
  }
}

void Init(int rt) {
  Dfs(rt, 0, 1);
  for (int j = 1; j < k; ++j) {
    for (int i = 1; i <= n; ++i) {
      anc[i][j] = anc[anc[i][j - 1]][j - 1];
    }
  }
}
```

```
void Swim(int &u, int h) {
  for (int i = 0; h > 0; ++i) {
    if (h & 1) u = anc[u][i];
    h >>= 1;
  }
}

int Query(int u, int v) {
  if (dep[u] < dep[v]) std::swap(u, v);
  Swim(u, dep[u] - dep[v]);
  if (u == v) return u;
  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";

int pre[maxn << 2];
struct edge { int v, next; };
edge g[maxn << 2];
int head[maxn];
int tot;
struct query { int q, next, index; };
query qg[maxn << 2];
int qhead[maxn];
int qtot;
int vis[maxn];
int anc[maxn];
int ans[maxn];

int Find(int x) { return pre[x] == x ? x : pre[x] = Find(pre[x]); }

void Union(int x, int y) { pre[Find(x)] = Find(y); }

void AddEdge(int u, int v) {
  g[tot] = edge {v, head[u]};
  head[u] = tot++;
}

// 添加询问
void AddQuery(int u, int v, int index) {
  qg[qtot] = query {v, qhead[u], index};
  qhead[u] = qtot++;
  qg[qtot] = query {u, qhead[v], index};
  qhead[v] = qtot++;
}

// 初始化
void Init() {
  tot = 0;
  memset(head, -1, sizeof(head));
  qtot = 0;
  memset(qhead, -1, sizeof(qhead));
  memset(vis, false, sizeof(vis));
  memset(pre, -1, sizeof(pre));
  memset(anc, 0, sizeof(anc));
  for (int i = 0; i <= n; ++i) pre[i] = i;
}

// LCA 离线 Tarjan 算法
void Tarjan(int u) {
  anc[u] = u;
  vis[u] = true;
  for (int i = head[u]; ~i; i = g[i].next) {
    if (vis[g[i].v]) continue;
    Tarjan(g[i].v);
    Join(u, g[i].v);
    anc[Find(u)] = u;
  }
  for (int i = qhead[u]; ~i; i = qg[i].next) {
    if (vis[qg[i].q]) ans[qg[i].index] = anc[Find(qg[i].q)];
  }
}
```

3.6 MultipleTree

```
/*
  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>
using namespace std;
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 = 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 = 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 << 2];

  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) >> 1;
    Build(o << 1, l, m);
    Build(o << 1 | 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) >> 1;
    if (ll <= m) Modify(o << 1, l, m, ll, rr, u, v);
    if (rr > m) Modify(o << 1 | 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) >> 1, ans = 0;
    if (ll <= m) ans += QueryRank(o << 1, l, m, ll, rr, v);
    if (rr > m) ans += QueryRank(o << 1 | 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) >> 1, ans = -inf;
    if (ll <= m) ans = max(ans, QueryPrev(o << 1, l, m, ll, rr, v));
    if (rr > m) ans = max(ans, QueryPrev(o << 1 | 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) >> 1, ans = inf;
    if (ll <= m) ans = min(ans, QueryNext(o << 1, l, m, ll, rr, v));
    if (rr > m) ans = min(ans, QueryNext(o << 1 | 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) >> 1) + 1;
      if (QueryRank(1, 1, n, ll, rr, m) < v) l = m;
      else r = m - 1;
    }
    return l;
  }
};

int main() {
  ios::sync_with_stdio(false);
  cin.tie(nullptr); cout.tie(nullptr);
  int m; cin >> n >> m;
  for (int i = 0; i < n; ++i) cin >> arr[i];
  SplayTree::tot = 0;
  SegTree::Build(1, 1, n);
  for (int i = 0, op, l, r, pos, k; i < m; ++i) {
    cin >> op;
    if (op == 1) {
      cin >> l >> r >> k;
      cout << SegTree::QueryRank(1, 1, n, l, r, k) + 1 << endl;
    }
    else if (op == 2) {
      cin >> l >> r >> k;
      cout << SegTree::QueryKth(l, r, k) << endl;
    }
    else if (op == 3) {
      cin >> pos >> k;
      SegTree::Modify(1, 1, n, pos, pos, arr[pos - 1], k);
      arr[pos - 1] = k;
    }
    else if (op == 4) {
      cin >> l >> r >> k;
      cout << SegTree::QueryPrev(1, 1, n, l, r, k) << endl;
    }
    else if (op == 5) {
      cin >> l >> r >> k;
      cout << SegTree::QueryNext(1, 1, n, l, r, k) << endl;
    }
  }
  return 0;
}
```

3.7 SegmentTree

3.7.1 AreaCombination

```
// 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 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 << 4];

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;
}
```

3.7.2 AreaXorCombination

```
// 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; }
  };

  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 << 1], tree[o << 1 | 1]); }

  void Push(int o, int l, int r) {
    int m = (l + r) >> 1;
    if (tree[o].lazy != 0) {
      tree[o << 1].v = x[m] - x[l - 1] - tree[o << 1].v;
      tree[o << 1 | 1].v = x[r] - x[m] - tree[o << 1 | 1].v;
      tree[o << 1].lazy ^= 1;
      tree[o << 1 | 1].lazy ^= 1;
      tree[o].lazy = 0;
    }
  }

  int n;
  std::vector<node> tree;

  void Build(int o, int l, int r) {
    if (l == r) return;
    int m = (l + r) >> 1;
    Build(o << 1, l, m);
    Build(o << 1 | 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) >> 1;
    if (ll <= m) Modify(o << 1, l, m, ll, rr);
    if (rr > m) Modify(o << 1 | 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) >> 1;
    node ans;
    if (ll <= m) ans = Unite(ans, Query(o << 1, l, m, ll, rr));
    if (rr > m) ans = Unite(ans, Query(o << 1 | 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::cin.tie(nullptr); std::cout.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;
}
```

3.7.3 MergeSegmentTree

```
// 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) >> 1, 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) >> 1, 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;
}
```

3.7.4 SegmentTree

```
// 求和线段树
template <typename type>
struct SegTree {
  struct node {
    type v, lazy;
    node() { v = lazy = 0; }
  };

  int n;
  std::vector<node> tree;

  node Unite(const node &k1, const node &k2) {
    node ret;
    ret.v = k1.v + k2.v;
    return ret;
  }

  void Pull(int o) { tree[o] = Unite(tree[o << 1], tree[o << 1 | 1]); }

  void Push(int o, int l, int r) {
    int m = (l + r) >> 1;
    if (tree[o].lazy != 0) {
      tree[o << 1].v += (m - l + 1) * tree[o].lazy;
      tree[o << 1 | 1].v += (r - m) * tree[o].lazy;
      tree[o << 1].lazy += tree[o].lazy;
      tree[o << 1 | 1].lazy += tree[o].lazy;
      tree[o].lazy = 0;
    }
  }

  template <typename t>
  void Build(int o, int l, int r, const std::vector<t> &v) {
    if (l == r) {
      tree[o].v = v[l - 1];
      return;
    }
    int m = (l + r) >> 1;
    Build(o << 1, l, m, v);
    Build(o << 1 | 1, m + 1, r, v);
    Pull(o);
  }

  template <typename t>
  SegTree(const std::vector<t> &v) {
    n = v.size();
    tree.resize((n << 2) + 1);
    Build(1, 1, n, v);
  }

  template <typename t>
  void Modify(int o, int l, int r, int ll, int rr, t v) {
    if (ll <= l && rr >= r) {
      tree[o].v += (r - l + 1) * v;
      tree[o].lazy += v;
      return;
    }
    Push(o, l, r);
    int m = (l + r) >> 1;
    if (ll <= m) Modify(o << 1, l, m, ll, rr, v);
    if (rr > m) Modify(o << 1 | 1, m + 1, r, ll, rr, v);
    Pull(o);
  }
  template <typename t>
  void Modify(int ll, int rr, t v) { Modify(1, 1, n, ll, rr, v); }

  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) >> 1;
    node ret;
    if (ll <= m) ret = Unite(ret, Query(o << 1, l, m, ll, rr));
    if (rr > m) ret = Unite(ret, Query(o << 1 | 1, m + 1, r, ll, rr));
    return ret;
  }
  node Query(int ll, int rr) { return Query(1, 1, n, ll, rr); }
};
```

3.8 SparseTable

```
template <typename type>
struct SparseTable {
  std::vector<std::vector<type>> max, min;

  STTable(const std::vector<type> &arr) {
    int n = (int)arr.size(), m = log2(n) + 1;
    max = min = std::vector<std::vector<type>>(n, std::vector<type>(m, 0));
    for (int i = 0; i < n; ++i) max[i][0] = min[i][0] = arr[i];
    for (int j = 1; j < m; ++j) {
      for (int i = 0; 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]);
      }
    }
  }

  type QueryMax(int l, int r) {
    int k = log2(r - l + 1);
    return std::max(max[l][k], max[r - (1 << k) + 1][k]);
  }

  type QueryMin(int l, int r) {
    int k = log2(r - l + 1);
    return std::min(min[l][k], min[r - (1 << k) + 1][k]);
  }
};
```

3.9 SplayTree

```
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];

  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 o) {
    for (int f = fa[o]; f = fa[o], f; Rotate(o))
      if (fa[f]) Rotate(Get(o) == Get(f) ? f : o);
    rt = o;
  }

  void Insert(int x) {
    if (!rt) {
      val[++tot] = x;
      cnt[tot]++;
      rt = tot;
      Push(rt);
      return;
    }
    int cur = rt, f = 0;
    while (true) {
      if (val[cur] == x) {
        cnt[cur]++;
        Push(cur); Push(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;
        Push(tot); Push(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];
    }
  }
}

// after insert, before delete
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;
}

void Delete(int x) {
  GetRank(x);
  if (cnt[rt] > 1) {
    cnt[rt]--;
    Push(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);
    Push(rt);
  }
};
```

3.10 SplayTreeArray

```
const int inf = "Edit"
const int maxn = "Edit";

struct SplayTree {
  // rt:Splay Tree 根节点
  int rt;
  // son[i][0]:i 节点的左孩子, son[i][0]:i 节点的右孩子
  int son[maxn][2];
  // fa[i]:i 节点的父节点
  int fa[maxn];
  // val[i]:i 节点的权值
  int val[maxn];
  // sz[i]: 以 i 节点为根的 Splay Tree 的节点数 (包含自身)
  int sz[maxn];
  // 惰性标记数组
  bool lazy[maxn];

  void Push(int o) { sz[o] = sz[son[o][0]] + sz[son[o][1]] + 1; }

  void Pull(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;
    }
  }

  // 判断 o 节点是其父节点的左孩子还是右孩子
  bool Get(int o) { return son[fa[o]][1] == o; }

  // 旋转节点 o
  void Rotate(int o) {
    int p = fa[o], q = fa[p], ck = Get(o);
    Pull(p); Pull(o);
    son[p][ck] = son[o][ck ^ 1];
    fa[son[p][ck]] = fa[o];
    son[o][ck ^ 1] = fa[o];
    fa[p] = o;
    fa[o] = q;
    if (q) son[q][p == son[q][1]] = o;
    Push(p); Push(o);
  }

  // 旋转 o 节点到节点 tar
  void Splay(int o, int tar = 0) {
    for (int cur = fa[o]; (cur = fa[o]) != tar; Rotate(o)) {
      Pull(fa[cur]); Pull(cur); Pull(o);
      if (fa[cur] != tar) {
        if (Get(o) == Get(cur)) Rotate(cur);
        else Rotate(o);
      }
    }
    if (!tar) rt = o;
  }

  // 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);
  }

  // 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;
  }
```

```
  // 求节点 o 的前驱节点
  int GetPath(int o) {
    Splay(o, rt);
    int cur = son[rt][0];
    while (son[cur][1]) cur = son[cur][1];
    return cur;
  }

  // 求节点 o 的后继节点
  int GetNext(int o) {
    Splay(o, rt);
    int cur = son[rt][1];
    while (son[cur][0]) cur = son[cur][0];
    return cur;
  }

  // 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] = m - 1;
    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]);
  }
};
```

3.11 TreeSplit

```
const int maxn = "Edit";

int n;
int arr[maxn];

int fa[maxn], dep[maxn];
int sz[maxn], son[maxn];
int rk[maxn], top[maxn];
int id[maxn];
int dfs_clock;

struct edge { int v, next; };
edge g[maxn << 1];
int tot;
int head[maxn];

void AddEdge(int u, int v) {
  g[tot] = (edge){v, head[u]};
  head[u] = tot++;
}

long long sum[maxn << 2];
long long lazy[maxn << 2];

void SegTreePull(int o) { sum[o] = sum[o << 1] + sum[o << 1 | 1]; }

void SegTreePush(int o, int l, int r) {
  int m = (l + r) >> 1;
  if (lazy[o] != 0) {
    sum[o << 1] += (m - l + 1) * lazy[o];
    sum[o << 1 | 1] += (r - m) * lazy[o];
    lazy[o << 1] += lazy[o];
    lazy[o << 1 | 1] += lazy[o];
    lazy[o] = 0;
  }
}

void SegTreeBuild(int o, int l, int r) {
  if (l == r) {
    sum[o] = arr[rk[l]];
    return;
  }
  int m = (l + r) >> 1;
  SegTreeBuild(o << 1, l, m);
  SegTreeBuild(o << 1 | 1, m + 1, r);
  SegTreePull(o);
}
```

```
void SegTreeModify(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;
  }
  SegTreePush(o, l, r);
  int m = (l + r) >> 1;
  if (ll <= m) SegTreeModify(o << 1, l, m, ll, rr, v);
  if (rr > m) SegTreeModify(o << 1 | 1, m + 1, r, ll, rr, v);
  SegTreePull(o);
}

long long SegTreeQuery(int o, int l, int r, int ll, int rr) {
  if (ll <= l && rr >= r) return sum[o];
  SegTreePush(o, l, r);
  int m = (l + r) >> 1;
  long long ret = 0;
  if (ll <= m) ret += SegTreeQuery(o << 1, l, m, ll, rr);
  if (rr > m) ret += SegTreeQuery(o << 1 | 1, m + 1, r, ll, rr);
  return ret;
}

void TreeSplitDfs1(int u, int p, int d) {
  fa[u] = p; dep[u] = d; sz[u] = 1;
  for (int i = head[u]; ~i; i = g[i].next) {
    int v = g[i].v;
    if (v == p) continue;
    TreeSplitDfs1(v, u, d + 1);
    sz[u] += sz[v];
    if (sz[v] > sz[son[u]]) son[u] = v;
  }
}

void TreeSplitDfs2(int u, int tp) {
  top[u] = tp; id[u] = ++dfs_clock;
  rk[dfs_clock] = u;
  if (!son[u]) return;
  TreeSplitDfs2(son[u], tp);
  for (int i = head[u]; ~i; i = g[i].next) {
    int v = g[i].v;
    if (v == son[u] || v == fa[u]) continue;
    TreeSplitDfs2(v, v);
  }
}

long long TreeSplitQuery(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 += SegTreeQuery(1, 1, n, id[top[u]], id[u]);
    u = fa[top[u]];
  }
  if (id[u] > id[v]) std::swap(u, v);
  ret += SegTreeQuery(1, 1, n, id[u], id[v]);
  return ret;
}

void TreeSplitModify(int u, int v, int c) {
  while (top[u] != top[v]) {
    if (dep[top[u]] < dep[top[v]]) std::swap(u, v);
    SegTreeModify(1, 1, n, id[top[u]], id[u], c);
    u = fa[top[u]];
  }
  if (id[u] > id[v]) std::swap(u, v);
  SegTreeModify(1, 1, n, id[u], id[v], c);
}
```

4 GraphTheory

4.1 AStar

```
const int inf = "Edit";
const int maxn = "Edit";

struct edge { int v, c, next; };

edge g[maxn << 1];
int head[maxn];
int tot;
edge rev_g[maxn << 1];
int rev_head[maxn];
int rev_tot;

void Init() {
  tot = 0;
  memset(head, -1, sizeof(head));
  rev_tot = 0;
  memset(rev_head, -1, sizeof(rev_head));
}

void AddEdge(int u, int v, int c) {
  g[tot] = edge {v, c, head[u]};
  head[u] = tot++;
  rev_g[rev_tot] = edge {u, c, rev_head[v]};
  rev_head[v] = rev_tot++;
}

int dis[maxn];

struct Cmp { bool operator() (const int &k1, const int &k2) { return dis[k1] >
↪  dis[k2]; } };

// 利用反向边图求各点到终点的最短路
void Dijkstra(int s) {
  priority_queue<int, vector<int>, Cmp> que;
  memset(dis, inf, sizeof(dis));
  dis[s] = 0;
  que.push(s);
  while (!que.empty()) {
    int u = que.top(); que.pop();
    for (int i = rev_head[u]; i != -1; i = rev_g[i].next) {
      if (dis[rev_g[i].v] > dis[u] + rev_g[i].c) {
        dis[rev_g[i].v] = dis[u] + rev_g[i].c;
        que.push(rev_g[i].v);
      }
    }
  }
}

struct node {
  int f, g, p;
  // k* 核心:f=g+H(p), 这里 H(p)=dis[p]
  bool operator < (const node &k) const {
    if (f == k.f) return g > k.g;
    return f > k.f;
  }
};

// A* 算法求起点 s 到终点 t 的第 k 短路
int AStar(int s, int t, int k) {
  int cnt = 0;
  priority_queue<node> que;
  // 注意特盘相同点是否算最短路
  if (s == t) k++;
  if (dis[s] == inf) return -1;
  que.push(node {dis[s], 0, s});
  while (!que.empty()) {
    node keep = que.top(); que.pop();
    if (keep.p == t) {
      cnt++;
      if (cnt == k) return keep.g;
    }
    for (int i = head[keep.p]; i != -1; i = g[i].next) {
      node tmp;
      tmp.p = g[i].v;
      tmp.g = keep.g + g[i].c;
      tmp.f = tmp.g + dis[tmp.p];
      que.push(tmp);
    }
  }
  return -1;
}
```

4.2 MinimumSpanningTree

4.2.1 Kruskal

```
const int maxn = "Edit";

int n, pre[maxn];
struct edge { int u, v, c; };
bool operator < (edge k1 edge k2) { return k1.c < k2.c; }
std::vector<edge> g;

int Find(int x) { return pre[x] == x ? x : pre[x] = Find(pre[x]); }

void Union(int x, int y) { pre[Find(x)] = Find(y); }
```

```
int Kruskal() {
  std::sort(g.begin(), g.end());
  for (int i = 0; i <= n; ++i) pre[i] = i;
  int ret = 0;
  for (auto &e : g) {
    if (Find(e.u) != Find(e.v)) {
      Union(e.u, e.v);
      ret += e.c;
    }
  }
  return ret;
}
```

4.2.2 Prim

```
const int inf = "Edit";
const int maxn = "Edit";

int n;
int dis[maxn];
int vis[maxn];
struct edge { int v, dis; };
std::vector<edge> g[maxn];

void AddEdge(int u, int v, int c) {
  g[u].push_back((edge){v, c});
  g[v].push_back((edge){u, c});
}

// Prim 算法
int Prim(int s) {
  memset(dis, inf, sizeof(dis));
  memset(vis, 0, sizeof(vis));
  dis[s] = 0;
  int ret = 0;
  for (int i = 1; i <= n; ++i) {
    int u = -1, min = inf;
    for (int j = 1; j <= n; ++j) {
      if (!vis[j] && dis[j] < min) {
        u = j;
        min = dis[j];
      }
    }
    vis[u] = 1;
    ret += min;
    for (int j = 0; j < int(g[u].size()); ++j) {
      int v = g[u][j].v;
      if (!vis[v] && g[u][j].dis < dis[v]) dis[v] = g[u][j].dis;
    }
  }
  return ret;
}
```

4.3 NetFlow

4.3.1 Dinic

```
const int maxn = "Edit";
const int inf = "Edit";

struct edge { int v, flow, next; };
edge g[maxn << 2];
int tot;
int head[maxn];
int dep[maxn];
int cur[maxn];

void AddEdge(int u, int v, int flow, int rev = 0) {
  g[tot] = (edge){v, flow, head[u]};
  head[u] = tot++;
  g[tot] = (edge){u, rev, head[v]};
  head[v] = tot++;
}

bool Bfs(int s, int t) {
  memset(dep, -1, sizeof(dep));
  std::queue<int> que;
  dep[s] = 0;
  que.push(s);
  while (!que.empty()) {
    int u = que.front(); que.pop();
    for (int i = head[u]; ~i; i = g[i].next) {
      if (dep[g[i].v] == -1 && g[i].flow > 0) {
        dep[g[i].v] = dep[u] + 1;
        que.push(g[i].v);
      }
    }
  }
  return dep[t] != -1;
}

int Dfs(int u, int t, int flow) {
  if (u == t || flow == 0) return flow;
  int max = 0, find_flow;
  for (int &i = cur[u]; ~i; i = g[i].next) {
    if (g[i].flow > 0 && dep[g[i].v] == dep[u] + 1) {
      find_flow = Dfs(g[i].v, t, std::min(flow - max, g[i].flow));
      if (find_flow > 0) {
        g[i].flow -= find_flow;
        g[i ^ 1].flow += find_flow;
        max += find_flow;
        if (max == flow) return flow;
      }
    }
  }
  if (!max) dep[u] = -2;
  return max;
}

int Dinic(int s, int t) {
  int ret = 0;
  while (Bfs(s, t)) {
    for (int i = s; i <= t; ++i) cur[i] = head[i];
    ret += Dfs(s, t, inf);
  }
  return ret;
}
```

4.3.2 FordFulkerson

```
const int inf = "Edit";
const int maxn = "Edit";

int n, e;
bool vis[maxn];
int g[maxn][maxn];

// Dfs 搜索增广路经, u: 当前搜索顶点, t: 搜索终点, now_flow: 当前最大流量
int Dfs(int u, int t, int now_flow) {
  if (u == t) return now_flow;
  vis[u] = true;
  for (int i = 1; i <= n; ++i) {
    if (!vis[i] && g[u][i]) {
      int FindFlow = Dfs(i, t, now_flow < g[u][i] ? now_flow : g[u][i]);
      if (!FindFlow) continue;
      g[u][i] -= FindFlow;
      g[i][u] += FindFlow;
      return FindFlow;
    }
  }
  return false;
}

// Ford-Fulkersone 算法, s: 起点, t: 终点
int FordFulkerson(int s, int t) {
  int max_flow = 0, Flow = 0;
  memset(vis, false, sizeof(vis));
  while (Flow = Dfs(s, t, inf)) {
    max_flow += Flow;
    memset(vis, false, sizeof(vis));
  }
  return max_flow;
}
```

4.3.3 MaxFlow

```
const int inf = "Edit";

int s, t;
struct edge { int to, cap, rev; };
std::vector<std::vector<edge>> g;
std::vector<bool> vis;

void Init(int n) {
  s = 0; t = n;
  g.resize(n + 1);
}

void AddEdge(int u, int v, int cap, int rev = 0) {
  g[u].push_back((edge){v, cap, (int)g[v].size()});
  g[v].push_back((edge){u, rev, (int)g[u].size() - 1});
}

int Dfs(int u, int t, int flow) {
  if (u == t) return flow;
  vis[u] = true;
  for (edge &e : g[u]) {
    if (!vis[e.to] && e.cap > 0) {
      int f = Dfs(e.to, t, std::min(e.cap, flow));
      if (f > 0) {
        e.cap -= f;
        g[e.to][e.rev].cap += f;
        return f;
      }
    }
  }
  return 0;
}

int GetMaxFlow(int s, int t) {
  int ret = 0;
  while (true) {
    vis.assign(t + 1, false);
    int flow = Dfs(s, t, inf);
    if (flow == 0) return ret;
    ret += flow;
  }
}
```

4.3.4 MinCostMaxFlow

```
const int inf = "Edit";
const int maxn = "Edit";

struct edge { int v, cap, cost, flow, next; };
int n, e;
int head[maxn];
int path[maxn];
int dis[maxn];
bool vis[maxn];
int tot;
edge g[maxn];

void AddEdge(int u, int v, int cap, int cost) {
  g[tot] = (edge){v, cap, cost, 0, head[u]};
  head[u] = tot++;
  g[tot] = (edge){u, 0, -cost, 0, head[v]};
  head[v] = tot++;
}

bool SPFA(int s, int t) {
  memset(dis, inf, sizeof(dis));
  memset(vis, false, sizeof(vis));
  memset(path, -1, sizeof(path));
  dis[s] = 0;
  vis[s] = true;
  std::queue<int> que;
  while (!que.empty()) que.pop();
  que.push(s);
  while (!que.empty()) {
    int U = que.front();
    que.pop();
    vis[U] = false;
    for (int i = head[U]; ~i; i = g[i].next) {
      int v = g[i].v;
      if (g[i].cap > g[i].flow && dis[v] > dis[U] + g[i].cost) {
        dis[v] = dis[U] + g[i].cost;
        path[v] = i;
        if (!vis[v]) {
          vis[v] = true;
          que.push(v);
        }
      }
    }
  }
  return path[t] != -1;
}

int MinCostMaxflow(int s, int t, int &min_cost) {
  int max_flow = 0;
  min_cost = 0;
  while (SPFA(s, t)) {
    int min = inf;
    for (int i = path[t]; ~i; i = path[g[i ^ 1].v]) {
      if (g[i].cap - g[i].flow < min) min = g[i].cap - g[i].flow;
    }
    for (int i = path[t]; ~i; i = path[g[i ^ 1].v]) {
      g[i].flow += min;
      g[i ^ 1].flow -= min;
      min_cost += g[i].cost * min;
    }
    max_flow += min;
  }
  return max_flow;
}
```

4.4 ShortestPath

4.4.1 BellmanFord

```
const int inf = "Edit";
const int maxn = "Edit";

int n;
struct edge { int u, v, dis; };
int dis[maxn];
std::vector<edge> g;

// Bellman_Ford 算法判断是否存在负环回路
bool BellmanFord(int s) {
  memset(dis, inf, sizeof(dis));
  dis[s] = 0;
  // 最多做 N-1 次
  for (int i = 1; i < n; ++i) {
    bool flag = false;
    for (int j = 0; j < (int)g.size(); ++j) {
      if (dis[g[j].v] > dis[g[j].u] + g[j].dis) {
        dis[g[j].v] = dis[g[j].u] + g[j].dis;
        flag = true;
      }
    }
    if (!flag) return true;
  }
  for (int j = 0; j < (int)g.size(); ++j) {
    if (dis[g[j].v] > dis[g[j].u] + g[j].dis) return false;
  }
  return true;
}
```

4.4.2 Dijkstra

```
const int maxn = "Edit";
const int inf = "Edit";

struct edge { int v, c, next; };
edge g[maxn << 1];
int head[maxn];
int tot;
int dis[maxn];

void AddEdge(int u, int v, int c) {
  g[tot] = (edge){v, c, head[u]};
  head[u] = tot++;
}

struct Cmp { bool operator() (const int &k1, const int &k2) { return dis[k1] >
↪  dis[k2]; } };

int n, e;

void Dijkstra(int s) {
  std::priority_queue<int, std::vector<int>, Cmp> que;
  memset(dis, inf, sizeof(dis));
  dis[s] = 0;
  que.push(s);
  while (!que.empty()) {
    int u = que.top(); que.pop();
    for (int i = head[u]; ~i; i = g[i].next) {
      if (dis[g[i].v] > dis[u] + g[i].c) {
        dis[g[i].v] = dis[u] + g[i].c;
        que.push(g[i].v);
      }
    }
  }
}
```

4.4.3 Floyd

```
const int maxn = "Edit";

int n;
int dis[maxn][maxn];

void Floyd() {
  for (int k = 1; k <= n; ++k) {
    for (int i = 1; i <= n; ++i) {
      for (int j = 1; j <= n; ++j) {
        dis[i][j] = std::min(dis[i][j], dis[i][k] + dis[k][j]);
      }
    }
  }
}
```

4.4.4 SPFA

```
const int inf = "Edit";
const int maxn = "Edit";

struct edge { int v, dis; };
int n, e;
bool vis[maxn];
int cnt[maxn];
int dis[maxn];
std::vector<edge> g[maxn];

void AddEdge (int u, int v, int c) {
  g[u].push_back((edge){v, c});
  g[v].push_back((edge){u, c});
}

bool SPFA(int s) {
  memset(vis, false, sizeof(vis));
  memset(dis, inf, sizeof(dis));
  memset(cnt, 0, sizeof(cnt));
  vis[s] = true;
  dis[s] = 0;
  cnt[s] = 1;
  std::queue<int> que;
  while (!que.empty()) que.pop();
  que.push(s);
  while (!que.empty()) {
    int U = que.front();
    que.pop();
    vis[U] = false;
    for (int i = 0; i < (int)g[U].size(); ++i) {
      int v = g[U][i].v;
      if (dis[v] > dis[U] + g[U][i].dis) {
        dis[v] = dis[U] + g[U][i].dis;
        if (!vis[v]) {
          vis[v] = true;
          que.push(v);
          if (++cnt[v] > N) return false;
        }
      }
    }
  }
  return true;
}
```

5 DynamicProgramming

5.1 Contour

```
const int maxn = "Edit";

int dp[2][1 << maxn];

void Update(int cur, int a, int b) {
  if (b & (1 << M)) dp[cur][b ^ (1 << M)] = dp[cur][b ^ (1 << M)] + dp[cur ^ 1][a];
}

// 轮廓线 dp(1*2 在 N*M 图上摆放数)
int Contour(int N, int M) {
  memset(dp, 0, sizeof(dp));
  int cur = 0;
  dp[cur][(1 << M) - 1] = 1;
  for (int i = 0; i < N; ++i) {
    for (int j = 0; j < M; ++j) {
      cur ^= 1;
      memset(dp[cur], 0, sizeof(dp[cur]));
      for (int k = 0; k < (1 << M); ++k) {
        Update(cur, k, k << 1);
        if (i && !(k & (1 << (M - 1)))) Update(cur, k, (k << 1) ^ (1 << M) ^ 1);
        if (j && (!(k & 1))) Update(cur, k, (k << 1) ^ 3);
      }
    }
  }
  return dp[cur][(1 << M) - 1];
}
```

5.2 Digit

```
const int maxn = "Edit";

int digit[25];
int dp[25][maxn];

// site: 数位,status: 状态,pre: 前导零,limit: 数位上界
int Dfs(int site, int status, bool pre, bool limit) {
  if (site == 0) return ?;
  if (!limit && ~dp[site][status]) return dp[site][status];
  int max = limit ? digit[site] : 9;
  int ret = 0;
  for (int i = 0; i <= max; ++i) {
    int new_status = /* 状态转移 */;
    if (new_status?) ret += Dfs(site - 1, new_status, pre && i == 0, limit && i ==
    ↪  max);
  }
  if (!limit) dp[site][status] = ret;
  return ret;
}

int Get(int x) {
  int len = 0;
  while (x) {
    digit[++len] = x % 10;
    x /= 10;
  }
  return Dfs(len, 0, true, true);
}
```

5.3 LCS

```
const int maxn = "Edit";

// dp[i][j]:str1[1]~str1[i] 和 str2[1]~str2[j] 对应的公共子序列长度
int dp[maxn][maxn];

// 最长公共子序列 (LCS)
void GetLCS(std::string str1, std::string str2) {
  for (int i = 0; i < (int)str1.length(); ++i) {
    for (int j = 0; j < (int)str2.length(); ++j) {
      if (str1[i] == str2[j]) dp[i + 1][j + 1] = dp[i][j] + 1;
      else dp[i + 1][j + 1] = std::max(dp[i][j + 1], dp[i + 1][j]);
    }
  }
}
```

5.4 LIS

```
// 最长不下降子序列 (LIS), arr: 序列
int GetLIS(std::vector<int> &arr) {
  int ret = 1;
  // last[i]: 长度为 i 的不下降子序列末尾元素的最小值
  std::vector<int> last((int)arr.size(), 0);
  last[0] = arr[0];
  for (int i = 1; i < (int)arr.size(); ++i) {
    if (arr[i] >= last[ret]) last[++ret] = arr[i];
    else {
      int pos = std::upper_bound(last.begin(), last.end(), arr[i]) - last.begin();
      last[pos] = arr[i];
    }
  }
  return ret;
}
```

5.5 Pack

```
const int maxn = "Edit";
```

```
int dp[maxn];
// cap: 背包容量, cnt: 总物品数
int cap, cnt;

// 01 背包, 代价为 cost, 获得的价值为 weight
void ZeroOnePack(int cost, int weight) {
  for (int i = cap; i >= cost; --i) dp[i] = std::max(dp[i], dp[i - cost] + weight);
}

// 完全背包, 代价为 cost, 获得的价值为 weight
void CompletePack(int cost, int weight) {
  for (int i = cost; i <= cap; ++i) dp[i] = std::max(dp[i], dp[i - cost] + weight);
}

// 多重背包, 代价为 cost, 获得的价值为 weight, 数量为 amount
void MultiplePack(int cost, int weight, int amount) {
  if (cost * amount >= cap) CompletePack(cost, weight);
  else {
    int k = 1;
    while (k < amount) {
      ZeroOnePack(k * cost, k * weight);
      amount -= k;
      k <<= 1;
    }
    ZeroOnePack(amount * cost, amount * weight);
  }
}
```

6 Geometry

6.1 DynamicConvexhull

```
// 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;
}
```

6.2 Pick

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

6.3 Plane

```
namespace Geometry {
  typedef double db;
  const db inf = "Edit";
  const int maxn = "Edit";
  const db eps = "Edit";
  const db delta = 0.98;

  int Sgn(db k) { return fabs(k) < eps ? 0 : (k < 0 ? -1 : 1); }
  int Cmp(db k1, db k2) { return Sgn(k1 - k2); }
  db Max(db k1, db k2) { return Cmp(k1, k2) > 0 ? k1 : k2; }
  db Min(db k1, db k2) { return Cmp(k1, k2) < 0 ? k1 : k2; }

  struct point { db x, y; };
  bool operator == (point k1, point k2) { return Cmp(k1.x, k2.x) == 0 && Cmp(k1.y,
   ↪  k2.y) == 0; }
  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; }
  point operator * (point k1, db k2) { return (point){k1.x * k2, k1.y * k2}; }
  point operator / (point k1, db k2) { return (point){k1.x / k2, k1.y / k2}; }
  db GetLen(point k) { return sqrt(k * k); }
  point GetUnit(point k) { db len = Getlen(k); return (point){k.x / len, k.y /
  ↪  len}; }
  db GetDisP2P(point k1, point k2) { return sqrt((k1 - k2) * (k1 - k2)); }
  db GetDisP2P2(point k1, point k2) { return (k1 - k2) * (k1 - k2); }
  db GetAng(point k1, point k2) { return fabs(atan2(fabs(k1 ^ k2), k1 * k2)); }
  point Rotate(point k, db ang) { return (point){k.x * cos(ang) - k.y * sin(ang),
   ↪  k.x * sin(ang) + k.y * cos(ang)}; }
  point Rotate90(point k) { return (point){-k.y, k.x}; }
  bool IsConvexhull(const std::vector<point> &p) {
    int sz = (int)p.size();
    for (int i = 0; i < sz; ++i)
      if (Sgn((p[(i + 1) % sz] - p[i]) ^ (p[(i + 2) % sz] - p[(i + 1) % sz])) < 0)
        return false;
    return true;
  }
  db ClosestP2P(point p[], int l, int r) {
    if (l + 1 == r) return GetDisP2P(p[l], p[r]);
    if (l + 2 == r) return Min(GetDisP2P(p[l + 1], p[r]), Min(GetDisP2P(p[l], p[l +
     ↪  1]), GetDisP2P(p[l], p[r])));
    int mid = (l + r) >> 1;
    db ret = Min(ClosestP2P(l, mid), ClosestP2P(mid + 1, r));
    std::vector<point> mid_p;
    for (int i = l; i <= r; ++i) {
      if (Cmp(fabs(p[i].x - p[mid].x), ret) <= 0) mid_p.push_back(p[i]);
    }
    std::sort(mid_p.begin(), mid_p.end(), [&](point k1, point k2) { return
     ↪  Cmp(k1.y, k2.y) < 0; });
    for (int i = 0; i < (int)mid_p.size(); ++i) {
      for (int j = i + 1; j < (int)mid_p.size(); ++j) {
        if (Cmp(mid_p[j].y - mid_p[i].y, ret) >= 0) break;
        ret = Min(ret, GetDisP2P(mid_p[i], mid_p[j]));
      }
    }
    return ret;
  }
```

```
typedef std::vector<point> poly;
db RotateCaliper(poly p) {
  db ret = -inf;
  if ((int)p.size() == 3) {
    if (Cmp(GetDisP2P(p[0], p[1]), ret) > 0) ret = GetDisP2P(p[0], p[1]);
    if (Cmp(GetDisP2P(p[0], p[2]), ret) > 0) ret = GetDisP2P(p[0], p[2]);
    if (Cmp(GetDisP2P(p[1], p[2]), ret) > 0) ret = GetDisP2P(p[1], p[2]);
    return;
  }
  int cur = 2, size = (int)p.size();
  for (int i = 0; i < size; ++i) {
    while (Cmp(fabs((p[i] - p[(i + 1) % size]) ^ (p[cur] - p[(i + 1) % size])),
    ↪  fabs((p[i] - p[(i + 1) % size]) ^ (p[(cur + 1) % size] - p[(i + 1) %
    ↪  size]))) < 0) cur = (cur + 1) % size;
    if (Cmp(GetDisP2P(p[i], p[cur]), ret) > 0) ret = GetDisP2P(p[i], p[cur]);
  }
  return ret;
}
poly Grahamscan(std::vector<point> p) {
  poly ret;
  if ((int)p.size() < 3) {
    for (point &v : p) ret.emplace_back(v);
    return ret;
  }
  int idx = 0;
  for (int i = 0; i < (int)p.size(); ++i)
    if (Cmp(p[i].x, p[idx].x) < 0 || (Cmp(p[i].x, p[idx].x) == 0 && Cmp(p[i].y,
    ↪  p[idx].y) < 0))
      idx = i;
  std::swap(p[0], p[idx]);
  std::sort(p.begin() + 1, p.end(), [&](point k1, point k2) {
    db tmp = (k1 - p[0]) ^ (k2 - p[0]);
    if (Sgn(tmp) > 0) return true;
    else if (Sgn(tmp) == 0 && Cmp(GetDisP2P(k1, p[0]), GetDisP2P(k2, p[0])) < 0)
    ↪  return true;
    return false;
  });
  ret.emplace_back(p[0]);
  for (int i = 1; i < (int)p.size(); ++i) {
    while ((int)ret.size() >= 2 && Sgn((ret.back() - ret[(int)ret.size() - 2]) ^
    ↪  (p[i] - ret[(int)ret.size() - 2])) <= 0) ret.pop_back();
    ret.emplace_back(p[i]);
  }
  return ret;
}
bool IsIn(point p, const poly &ch) {
  point base = ch[0];
  if (Sgn((p - base) ^ (ch[1] - p)) > 0 || Sgn((p - base) ^ (ch.back() - base)) <
  ↪  0) return false;
  if (Sgn((p - base) ^ (ch[1] - p)) == 0 && Cmp(GetLen(p - base), GetLen(ch[1] -
  ↪  base)) <= 0) return true;
  int idx = std::lower_bound(ch.begin(), ch.end(), p, [&] (point k1, point k2) {
  ↪  return Sgn((k1 - base) ^ (k2 - base)) > 0; }) - ch.begin() - 1;
  return Sgn((ch[idx + 1] - ch[idx]) ^ (p - ch[idx])) >= 0;
}
poly Minkowski(const poly &k1, const poly &k2) {
  int sz1 = (int)k1.size(), sz2 = (int)k2.size();
  std::queue<point> buf1, buf2;
  for (int i = 0; i < sz1; ++i) buf1.push(k1[(i + 1) % sz1] - k1[i]);
  for (int i = 0; i < sz2; ++i) buf2.push(k2[(i + 1) % sz2] - k2[i]);
  poly ret;
  ret.push_back(k1[0] + k2[0]);
  while (!buf1.empty() && !buf2.empty()) {
    point tmp1 = buf1.front(), tmp2 = buf2.front();
    if (Sgn(tmp1 ^ tmp2) > 0) {
      ret.push_back(ret.back() + tmp1);
      buf1.pop();
    }
    else {
      ret.push_back(ret.back() + tmp2);
      buf2.pop();
    }
  }
  while (!buf1.empty()) {
    ret.push_back(ret.back() + buf1.front());
    buf1.pop();
  }
  while (!buf2.empty()) {
    ret.push_back(ret.back() + buf2.front());
    buf2.pop();
  }
  return Grahamscan(ret);
}
db GetMinCircle(std::vector<point> p) {
  point cur = p[0];
  db pro = 10000, ret = inf;
  while (pro > eps) {
    int book = 0;
    for (int i = 0; i < (int)p.size(); ++i)
      if (GetDisP2P(cur, p[i]) > GetDisP2P(cur, p[book]))
        book = i;
    db r = GetDisP2P(cur, p[book]);
    if (Cmp(r, ret) < 0) ret = r;
    cur = cur + (p[book] - cur) / r * pro;
    pro *= delta;
  }
  return ret;
}
```

```
struct line { point s, t; };
typedef line seg;
db GetLen(seg k) { return GetDisP2P(k.s, k.t); }
db GetDisP2Line(point k1, line k2) { return fabs((k1 - k2.s) ^ (k2.t - k2.s)) /
↪  GetLen(k2); }
db GetDisP2Seg(point k1, seg k2) {
  if (Sgn((k1 - k2.s) * (k2.t - k2.s)) < 0 || Sgn((k1 - k2.t) * (k2.s - k2.t)) <
  ↪  0) {
    return Min(GetDisP2P(k1, k2.s), GetDisP2P(k1, k2.t));
  }
  return GetDisP2Line(k1, k2);
}
bool IsParallel(line k1, line k2) { return Sgn((k1.s - k1.t) ^ (k2.s - k2.t)) ==
↪  0; }
bool IsSegInterSeg(seg k1, seg k2) {
  return
    Max(k1.s.x, k1.t.x) >= Min(k2.s.x, k2.t.x) &&
    Max(k2.s.x, k2.t.x) >= Min(k1.s.x, k1.t.x) &&
    Max(k1.s.y, k1.t.y) >= Min(k2.s.y, k2.t.y) &&
    Max(k2.s.y, k2.t.y) >= Min(k1.s.y, k1.t.y) &&
    Sgn((k2.s - k1.t) ^ (k1.s - k1.t)) * Sgn((k2.t - k1.t) ^ (k1.s - k1.t)) <= 0
    ↪  &&
    Sgn((k1.s - k2.t) ^ (k2.s - k2.t)) * Sgn((k1.t - k2.t) ^ (k2.s - k2.t)) <= 0;
}
bool IsLineInterSeg(line k1, seg k2) {
  return Sgn((k2.s - k1.t) ^ (k1.s - k1.t)) * Sgn((k2.t - k1.t) ^ (k1.s - k1.t))
  ↪  <= 0;
}
bool IsLineInterLine(line k1, line k2) {
  return !IsParallel(k1, k2) || (IsParallel(k1, k2) && !(Sgn((k1.s - k2.s) ^
  ↪  (k2.t - k2.s)) == 0));
}
bool IsPointOnSeg(point k1, seg k2) {
  return Sgn((k1 - k2.s) ^ (k2.t - k2.s)) == 0 && Sgn((k1 - k2.s) * (k1 - k2.t))
  ↪  <= 0;
}
point Cross(line k1, line k2) {
  db temp = ((k1.s - k2.s) ^ (k2.s - k2.t)) / ((k1.s - k1.t) ^ (k2.s - k2.t));
  return (point){k1.s.x + (k1.t.x - k1.s.x) * temp, k1.s.y + (k1.t.y - k1.s.y) *
  ↪  temp};
}

// 表示 s->t 逆时针 (左侧) 的半平面
struct hulfplane:public line { db ang; };
void GetAng(halfplane k) { k.ang = atan2(k.t.y - k.s.y, k.t.x - k.s.x); }
bool operator < (halfplane k1, halfplane k2) {
  if (Sgn(k1.ang - k2.ang) > 0) return k1.ang < k2.ang;
  return Sgn((k1.s - k2.s) ^ (k2.t - k2.s)) < 0;
}
struct HalfPlaneInsert {
  int tot;
  halfplane hp[maxn];
  halfplane deq[maxn];
  point points[maxn];
  point res[maxn];
  int front, tail;

  void Push(halfplane k) { hp[tot++] = k; }

  void Unique() {
    int cnt = 1;
    for (int i = 1; i < tot; ++i)
      if (fabs(hp[i].ang - hp[i - 1].ang) > eps)
        hp[cnt++] = hp[i];
    tot = cnt;
  }

  bool IsHalfPlaneInsert() {
    for (int i = 0; i < tot; ++i) GetAng(hp[i]);
    sort(hp, hp + tot);
    Unique();
    deq[front = 0] = hp[0];
    deq[tail = 1] = hp[1];
    for (int i = 2; i < tot; ++i) {
      if (fabs((deq[tail].t - deq[tail].s) ^ (deq[tail - 1].t - deq[tail - 1].s))
       ↪  < eps || fabs((deq[front].t - deq[front].s) ^ (deq[front + 1].t -
       ↪  deq[front + 1].s)) < eps) return false;
      while (front < tail && ((Cross(deq[tail], deq[tail - 1]) - hp[i].s) ^
       ↪  (hp[i].t - hp[i].s)) > eps) tail--;
      while (front < tail && ((Cross(deq[front], deq[front + 1]) - hp[i].s) ^
       ↪  (hp[i].t - hp[i].s)) > eps) front++;
      deq[++tail] = hp[i];
    }
    while (front < tail && ((Cross(deq[tail], deq[tail - 1]) - deq[front].s) ^
     ↪  (deq[front].t - deq[front].s)) > eps) tail--;
    while (front < tail && ((Cross(deq[front], deq[front - 1]) - deq[tail].s) ^
     ↪  (deq[tail].t - deq[tail].t)) > eps) front++;
    if (tail <= front + 1) return false;
    return true;
  }

  void GetHalfPlaneInsertConvex() {
    int cnt = 0;
    for (int i = front; i < tail; ++i) res[cnt++] = Cross(deq[i], deq[i + 1]);
    if (front < tail - 1) res[cnt++] = Cross(deq[front], deq[tail]);
  }
};
```

```
  struct circle {point o; db r;};
  std::pair<point, pair> TangentP2Cir(circle k1, point k2) {
    db a = GetLen(k2 - k1.o), b = k1.r * k1.r / a, c = sqrt(Max(0.0, k1.r * k1.r -
    ↪  b * b));
    point k = GetUnit(k2 - k1.o), m = k1.o + k * b, del = Rotate90(k) * c;
    return {m - del, m + del};
  }
  circle GetCircle(point k1, point k2, point k3) {
    db a1 = k2.x - k1.x, b1 = k2.y - k1.y, c1 = (a1 * a1 + b1 * b1) / 2;
    db a2 = k3.x - k1.x, b2 = k3.y - k1.y, c2 = (a2 * a2 + b2 * b2) / 2;
    db d = a1 * b2 - a2 * b1;
    point o = (point){k1.x + (c1 * b2 - c2 * b1) / d, k1.y + (a1 * c2 - a2 * c1) /
    ↪  d};
    return (circle){o, GetDisP2P(k1, o)};
  }
  circle GetMinCircle(std::vector<point> p) {
    std::random_shuffle(p.begin(), p.end());
    int n = (int)p.size();
    circle ret = (circle){p[0], 0.0};
    for (int i = 1; i < n; ++i) {
      if (Cmp(GetDisP2P(ret.o, p[i]), ret.r) <= 0) continue;
      ret = (circle){p[i], 0.0};
      for (int j = 0; j < i; ++j) {
        if (Cmp(GetDisP2P(ret.o, p[j]), ret.r) <= 0) continue;
        ret.o = (p[i] + p[j]) / 2; ret.r = GetDisP2P(ret.o, p[i]);
        for (int k = 0; k < j; ++k) {
          if (Cmp(GetDisP2P(ret.o, p[k]), ret.r) <= 0) continue;
          ret = GetCircle(p[i], p[j], p[k]);
        }
      }
    }
    return ret;
  }
};
using namespace Geometry;
```

6.4 Simpson

```
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);
  }
};
```

6.5 Stereoscopic

```
namespace Geometry3D {
  typedef double db;
  const db inf = "Edit";
  const int maxn = "Edit";
  const db eps = "Edit";
  const db delta = 0.98;

  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, z; };
  bool operator == (point k1, point k2) { return Sgn(k1.x - k2.x) == 0 && Sgn(k1.y
  ↪  - k2.y) == 0 && Sgn(k1.z - k1.z) == 0; }
  point operator + (point k1, point k2) { return (point){k1.x + k2.x, k1.y + k2.y,
  ↪  k1.z + k2.z}; }
  point operator - (point k1, point k2) { return (point){k1.x - k2.x, k1.y - k2.y,
  ↪  k1.z - k2.z}; }
  db operator * (point k1, point k2) { return k1.x * k2.x + k1.y * k2.y + k1.z *
  ↪  k2.z; }
  db GetLen(point k) { return sqrt(k * k); }
  db GetLen2(point k) { return k * k; }
  db operator ^ (point k1, point k2) { return GetLen((point){k1.y * k2.z - k1.z *
  ↪  k2.y, k1.z * k2.x - k1.x * k2.z, k1.x * k2.y - k1.y * k2.x}); }
  point operator * (point k1, db k2) { return (point){k1.x * k2, k1.y * k2, k1.z *
  ↪  k2}; }
  point operator / (point k1, db k2) { return (point){k1.x / k2, k1.y / k2, k1.z /
  ↪  k2}; }
  db GetDisP2P(point k1, point k2) { return GetLen(k2 - k1); }
  db GetDisP2P2(point k1, point k2) { return GetLen2(k2 - k1); }
  db GetAngle(point k1, point k2) { return fabs(atan2(fabs(k1 ^ k2), k1 * k2)); }
  db GetMinSphere(std::vector<point> p) {
    point cur = p[0];
    db pro = 10000, ret = inf;
    while (pro > eps) {
      int mark = 0;
      for (int i = 0; i < (int)p.size(); ++i) {
        if (Cmp(GetDisP2P(cur, p[i]), GetDisP2P(cur, p[mark])) > 0) mark = i;
      }
      db r = GetDisP2P(cur, p[mark]);
      ret = min(ret, r);
      cur = cur + (p[mark] - cur) / r * pro;
      pro *= delta;
    }
    return ret;
  }

  struct line { point s, t; };
  typedef line seg;
  db GetLen(seg k) { return GetDisP2P(k.s, k.t); }
  db GetLe2(seg k) { return GetDisP2P2(k.s, k.t); }
  db GetDisP2Line(point k1, line k2) { return fabs((k1 - k2.s) ^ (k2.t - k2.s)) /
  ↪   GetLen(k2); }
  db GetDisP2Seg(point k1, seg k2) {
    if (Sgn((k1 - k2.s) * (k2.T - k2.s)) < 0 || Sgn((k1 - k2.t) * (k2.s - k2.t)) <
    ↪   0) {
      return min(GetDisP2P(k1, k2.s), GetDisP2P(k1, k2.t));
    }
    return GetDisP2Line(k1, k2);
  }
  struct sphere { point o;db r; };
  db GetV(sphere k) { return 4.0 / 3.0 * pi * k.r * k.r * k.r; }
  db GetSphereInterV(sphere k1, sphere k2) {
    db ret = 0.0;
    db dis = GetDisP2P(k1.o, k2.o);
    if (Sgn(dis - k1.r - k2.r) >= 0) return ret;
    if (Sgn(k2.r - (dis + k1.r)) >= 0) return GetV(k1);
    else if (Sgn(k1.r - (dis + k2.r)) >= 0) return GetV(k2);
    db len1 = ((k1.r * k1.r - k2.r * k2.r) / dis + dis) / 2;
    db len2 = dis - len1;
    db x1 = k1.r - len1, x2 = k2.r - len2;
    db v1 = pi * x1 * x1 * (k1.r - x1 / 3.0);
    db v2 = pi * x2 * x2 * (k2.r - x2 / 3.0);
    return v1 + v2;
  }

  struct ray { point o, dir; };
  bool IsRayInterSphere(ray k1, sphere k2, db &dis) {
    db a = k1.dir * k1.dir;
    db b = (k1.o - k2.o) * k1.dir * 2.0;
    db c = ((k1.o - k2.o) * (k1.o - k2.o)) - (k2.r * k2.r);
    db dlt = b * b - 4.0 * a * c;
    if (Sgn(dlt) < 0) return false;
    db x1 = (-b - sqrt(dlt)) / (2.0 * a), x2 = (-b + sqrt(dlt)) / (2.0 * a);
    if (Cmp(x1, x2) > 0) swap(x1, x2);
    if (Sgn(x1) <= 0) return false;
    dis = x1;
    return true;
  }
  void Reflect(ray &k1, sphere k2, db dis) {
    point pos = k1.o + (k1.dir * dis);
    Vector temp = k2.o + (((pos - k2.o) * ((pos - k2.o) * (k1.o - k2.o))) /
    ↪  GetLen2(pos - k2.o));
    k1.dir = temp * 2.0 - k1.o - pos; k1.o = pos;
  }
};
using namespace Geometry3D;
```

7 Others

7.1 Checker

```
/*
  // 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
*/
```

7.2 FastIO

```
// 普通读入挂
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;
}

// 普通输出挂
template <typename t>
inline void Out(t x) {
  if (x < 0) {
    putchar('-');
    x = -x;
  }
  if (x > 9) Out(x / 10);
  putchar(x % 10 + '0');
}

// 牛逼读入挂
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;
  }
};

// 超级读写挂
namespace IO{
  #define BUF_SIZE 100000
  #define OUT_SIZE 100000
  #define ll long long
  //fread->read

  bool IOerror=0;
  inline char nc(){
    static char buf[BUF_SIZE],*p1=buf+BUF_SIZE,*pend=buf+BUF_SIZE;
    if (p1==pend){
      p1=buf; pend=buf+fread(buf,1,BUF_SIZE,stdin);
      if (pend==p1){IOerror=1;return -1;}
      //{printf("IO error!\n");system("pause");for (;;);exit(0);}
    }
    return *p1++;
```

```
}
inline bool blank(char ch){return ch==' '||ch=='\n'||ch=='\r'||ch=='\t';}
inline void read(int &x){
  bool sign=0; char ch=nc(); x=0;
  for (;blank(ch);ch=nc());
  if (IOerror)return;
  if (ch=='-')sign=1,ch=nc();
  for (;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
  if (sign)x=-x;
}
inline void read(ll &x){
  bool sign=0; char ch=nc(); x=0;
  for (;blank(ch);ch=nc());
  if (IOerror)return;
  if (ch=='-')sign=1,ch=nc();
  for (;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
  if (sign)x=-x;
}
inline void read(double &x){
  bool sign=0; char ch=nc(); x=0;
  for (;blank(ch);ch=nc());
  if (IOerror)return;
  if (ch=='-')sign=1,ch=nc();
  for (;ch>='0'&&ch<='9';ch=nc())x=x*10+ch-'0';
  if (ch=='.'){
    double tmp=1; ch=nc();
    for (;ch>='0'&&ch<='9';ch=nc())tmp/=10.0,x+=tmp*(ch-'0');
  }
  if (sign)x=-x;
}
inline void read(char *s){
  char ch=nc();
  for (;blank(ch);ch=nc());
  if (IOerror)return;
  for (;!blank(ch)&&!IOerror;ch=nc())*s++=ch;
  *s=0;
}
inline void read(char &c){
  for (c=nc();blank(c);c=nc());
  if (IOerror){c=-1;return;}
}
//fwrite->write
struct Ostream_fwrite{
  char *buf,*p1,*pend;
  Ostream_fwrite(){buf=new char[BUF_SIZE];p1=buf;pend=buf+BUF_SIZE;}
  void out(char ch){
    if (p1==pend) fwrite(buf,1,BUF_SIZE,stdout);p1=buf;
    *p1++=ch;
  }
```

```
    void print(int x){
      static char s[15],*s1;s1=s;
      if (!x)*s1++='0';if (x<0)out('-'),x=-x;
      while(x)*s1++=x%10+'0',x/=10;
      while(s1--!=s)out(*s1);
    }
    void println(int x){
      static char s[15],*s1;s1=s;
      if (!x)*s1++='0';if (x<0)out('-'),x=-x;
      while(x)*s1++=x%10+'0',x/=10;
      while(s1--!=s)out(*s1); out('\n');
    }
    void print(ll x){
      static char s[25],*s1;s1=s;
      if (!x)*s1++='0';if (x<0)out('-'),x=-x;
      while(x)*s1++=x%10+'0',x/=10;
      while(s1--!=s)out(*s1);
    }
    void println(ll x){
      static char s[25],*s1;s1=s;
      if (!x)*s1++='0';if (x<0)out('-'),x=-x;
      while(x)*s1++=x%10+'0',x/=10;
      while(s1--!=s)out(*s1); out('\n');
    }
    void print(double x,int y){
      static ll mul[]={1,10,100,1000,10000,100000,1000000,10000000,100000000,
          1000000000,10000000000LL,100000000000LL,1000000000000LL,10000000000000LL,

          ↪  100000000000000LL,1000000000000000LL,10000000000000000LL,100000000000000000LL};
      if (x<-1e-12)out('-'),x=-x;x*=mul[y];
      ll x1=(ll)floor(x); if (x-floor(x)>=0.5)++x1;
      ll x2=x1/mul[y],x3=x1-x2*mul[y]; print(x2);
      if (y>0){out('.'); for (size_t i=1;i<y&&x3*mul[i]<mul[y];out('0'),++i);
      ↪  print(x3);}
    }
    void println(double x,int y){print(x,y);out('\n');}
    void print(char *s){while (*s)out(*s++);}
    void println(char *s){while (*s)out(*s++);out('\n');}
    void flush(){if (p1!=buf){fwrite(buf,1,p1-buf,stdout);p1=buf;}}
    ~Ostream_fwrite(){flush();}
  }Ostream;
  inline void print(int x){Ostream.print(x);}
  inline void println(int x){Ostream.println(x);}
  inline void print(char x){Ostream.out(x);}
  inline void println(char x){Ostream.out(x);Ostream.out('\n');}
  inline void print(ll x){Ostream.print(x);}
  inline void println(ll x){Ostream.println(x);}
  inline void print(double x,int y){Ostream.print(x,y);}
  inline void println(double x,int y){Ostream.println(x,y);}
  inline void print(char *s){Ostream.print(s);}
  inline void println(char *s){Ostream.println(s);}
  inline void println(){Ostream.out('\n');}
  inline void flush(){Ostream.flush();}
  #undef ll
  #undef OUT_SIZE
  #undef BUF_SIZE
};
using namespace IO;
```

7.3 LeepYear

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

7.4 MoAlgorithm

7.4.1 Dynamic

```
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 = (int)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;
```

7.4.2 Static

```
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 = (int)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;
```

7.5 STL

```
// pbds
#include <bits/extc++.h>

// --- tree ---
template< typename Key,
          typename Mapped,
          typename Cmp_Fn = std::less<Key>,
          typename Tag = rb_tree_tag,
          template<typename Node_Cltr,
                   typename Node_Itr,
                   typename Cmp_Fn_,
                   typename _Alloc_> class Node_Update = null_node_update,
          typename _Alloc = std::allocator<char> >
__gnu_pbds::tree< Key, Mapped, Cmp_Fn, Tag, Node_Update, _Alloc >

/*
Member types:
  Key:
    Key type.
  Mapped:
    Map type.
  Cmp_Fn:
    Comparison functior.
  Tag:
    Instantiating data structure type
    __gnu_pbds::ov_tree_tag: Ordered-vector tree.
    __gnu_pbds::rb_tree_tag: Red-black tree;
    __gnu_pbds::splay_tree_tag: Splay tree.
  Node_Update:
    Updates nodes, restores invariants when invalidated.
    XXX See design::tree-based-containers::node invariants.
  _Alloc:
    Allocator type.


Member functions:
  insert({key, mapped}):
    Insert element.
  erase({key, mapped}):
    Delete element.
  order_of_key({key, mapped}):
    Get the rank of {key, mapped}.
  find_by_order(int x):
    Find the k-th element. Return iterator.
  join(tree t):
    Insert the tree t into the tree if the origin tree type is the same ans there
↪   are no duplicate elements.
  split(value, t):
    Split the tree, the origin tree contains elements that are less than or equal
↪   to value, the tree t contains elements that are greater to value.
  lower_bound(value):
    Reture the iterator of the first element greater or equal to value.
  upper_bound(value):
    Reture the iterator of the first element greater to value.

  Examples:
    typedef __gnu_pbds::tree<int, int, std::less<int>, __gnu_pbds::splay_tree_tag,
↪   __gnu_pbds::tree_order_statistics_node_update> splay_tree;
*/
```

7.6 vim

```
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
```