Algorithm Library

Liu Yang

April 6, 2019

Contents

1 String

1.1 AhoCorasickAutomaton

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

class AhoCorasickAutomaton {
  public:
    // 子节点记录数组
    int son[maxn][26];
    int val[maxn];
    // 失配指针 Fail 数组
    int fail[maxn];
    // 节点数量
    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(string str) {
      int cur = 0, Len = int(str.length());
      for (int i = 0; i < Len; ++i) {
        int index = Pos(str[i]);
        if (!son[cur][index]) son[cur][index] = ++tot;
        cur = son[cur][index];
      }
      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(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, 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, 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());
  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";

class PalindromicTree {
  public:
    // 子节点记录数组
    long long son[maxn][26];
    // 失配指针 Fail 数组
    long long 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)
    long long 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)
    long long 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
    long long num[maxn];
    // 字符
    long long str[maxn];
    // 新添加字符后最长回文串表示的节点
    long long last;
    // 字符数量
    long long str_len;
    // 节点数量
    long long tot;

    // 新建节点
    long long NewNode(long long x) {
      for (long long 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;
    }

    long long GetFail(long long x) {
      while (str[str_len - len[x] - 1] != str[str_len]) x = fail[x];
      return x;
    }
    void Add(long long c) {
      c -= 'a';
      Str[++str_len] = c;
      long long Cur = GetFail(last);
      if (!son[Cur][c]) {
        long long 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 (long long i = tot - 1; i >= 0; --i) cnt[fail[i]] += cnt[i];
    }
};
```

1.5 TrieTree

```
const int maxn = "Edit";

class Trie {
  public:
    // Trie Tree 节点
    int son[maxn][26];
    // Trie Tree 节点数量
    int tot;

    // 字符串数量统计数组
    int cnt[maxn];

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

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

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

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

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

2 Math

2.1 Catalan

```
const int maxn = "Edit";

long long catalan[maxn];

// 递推求卡特兰数
void GetCalalan() {
  memset(catalan, 0, sizeof(catalan));
  catalan[0] = catalan[1] = 1;
  for (int i = 2; i < maxn; ++i) catalan[i] = Catalan[i - 1] * (4 * i - 2) / (i +
  ↪  1);
}
```

2.2 Derangement

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

long long staggered[maxn];

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

2.3 Euler

2.3.1 Euler

```
// 单独求解欧拉函数
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;
}
```

2.3.2 Screen

```
const int maxn = "Edit";

// 欧拉函数
```

```
int phi[maxn];

// 筛法求欧拉函数
void Euler() {
  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);
}
```

2.3.3 Sieve

```
const int maxn = "Edit";

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

// 同时求得欧拉函数和素数表
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];
    }
  }
}
```

2.4 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;
      }
    }
  }
}

// 多项式系数
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);
}
```

2.5 Fibonacci

```
const int mod = "Edit";

// 矩阵
struct matrix {long long mat[2][2];

// 重载矩阵乘法
matrix operator * (matrix &k1, matrix &k2) const {
  matrix ret;
  memset(ret.mat, 0, sizeof(ret.mat));
  for (int i = 0; i < 2; ++i) {
    for (int j = 0; j < 2; ++j) {
      for (int k = 0; k < 2; ++k) {
          ret.mat[i][j] = (ret.mat[i][j] + k1.mat[i][k] * k2.mat[k][j] % mod) %
          ↪  mod;
      }
    }
  }
  return ret;
}

// 重载矩阵快速幂
matrix operator ^ (matrix base, long long k) {
  matrix ret;
  memset(ret.mat, 0, sizeof(ret.mat));
  ret.mat[0][0] = ret.mat[1][1] = 1;
  while (k) {
    if (k & 1) ret = ret * base;
    base = base * base;
    k >>= 1;
  }
  return ret;
}

// 斐波那契数列中第 x 项
long long Fib(long long x) {
  matrix base;
  base.mat[0][0] = base.mat[1][0] = base.mat[0][1] = 1;
  base.mat[1][1] = 0;
  return (base ^ x).mat[0][1];
}
```

2.6 GeneratingFunction

```
const int maxn = "Edit";

void GetGeneratingFunction() {
  int n;
  int c1[maxn], c2[maxn];
  scanf("%d", &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;
    }
  }
}
```

2.7 InverseElement

2.7.1 ExtendGcd

```
// 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) {
  // 无最大公约数
  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 M)
long long GetInv(long long a, long long N) {
  long long x, y;
  long long d = ExtendGcd(a, N, x, y);
  if (d == 1) return (x % N + N) % N;
  else return -1;
}
```

2.7.2 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;
}
```

2.7.3 FermatLittleTheorem

```
const int mod = "Edit";

// 快速幂、费马小定理求逆元
long long Inv(long long x) {
  return Pow(x, mod - 2);
}
```

2.7.4 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;
}
```

2.8 Mobius

```
#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 &it : prime) {
      if (it * i >= maxn) break;
      is_prime[i * it] = false;
      if (i % it == 0) {
        mu[i * it] = 0;
        break;
      }
      mu[i * it] = -mu[i];
    }
  }
}
```

2.9 NimGame

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

2.10 Prime

2.10.1 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;
}
```

2.10.2 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;
    }
  }
}
```

2.11 QuickPow

```
const int mod = "edit";

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

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

2.12 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";

class binary_indexed_tree {
  public:
    int arr[maxn];

    void Update(int x, int v) {
      while (x < maxn) {
        arr[x] += v;
        x += lowbit(x);
      }
    }

    int GetSum(int x) {
      int ans = 0;
      while (x > 0) {
        ans += arr[x];
        x -= lowbit(x);
      }
    }
};
```

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

```
// 主席树, 静态区间第 k 小
const int maxn = "Edit";

class FuncSegTree {
  public:
    int tot;
    int rt[maxn];
    int lson[maxn << 5], rson[maxn << 5];
    int cnt[maxn << 5];

    int Build(int l, int r) {
      int t = ++tot;
      int m = (l + r) >> 1;
      if (l != r) {
        lson[t] = Build(l, m);
        rson[t] = Build(m + 1, r);
      }
      return t;
    }

    int Modify(int prev, int l, int r, int x) {
      int t = ++tot;
      lson[t] = lson[prev]; rson[t] = rson[prev];
      cnt[t] = cnt[prev] + 1;
      int m = (l + r) >> 1;
      if (l != r) {
        if (x <= m) lson[t] = Modify(lson[t], l, m, x);
        else rson[t] = Modify(rson[t], m + 1, r, x);
      }
      return t;
    }

    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);
    }
};
```

3.4 Hash

```
// 离散化
class Hash {
  public:
    int size;
    vector<int> arr;

    Hash(const vector<int> &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(int k) {
      return lower_bound(arr.begin(), arr.end(), k) - arr.begin();
    }
};
```

3.5 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(0); cout.tie(0);
  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;
}
```

3.6 SegmentTree

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

3.7 SplayTree

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

class splay_tree {
 public:
  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);
    }
};
```

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;

// 链式前向星存图初始化
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 &A, const int &B) {
    return dis[A] > dis[B];
  }
};

// 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++;
  // 起点与终点不连通
  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) {
        // 返回第 k 短路长度
        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 LCA

4.2.1 DFS+ST

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

```
// 链式前向星存图
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++;
}

namespace LCAOnline {
  // 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];
  // 记录父节点
  int fa[maxn];
  // 记录与根节点距离
  int dis[maxn];
  // 遍历节点数量
  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])];
  }
}
```

4.2.2 Tarjan

```
const int maxn = "Edit";

struct edge {int v, next;};
struct query {int q, next, index;};

int pre[maxn << 2];
edge g[maxn << 2];
int head[maxn];
int tot;
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)];
  }
}
```

4.3 MinimumSpanningTree

4.3.1 Kruskal

```
const int maxn = "Edit";

struct edge {int u, v, c;};
bool operator < (edge k1 edge k2) {return k1.c < k2.c;}

int n, e, pre[maxn];
std::vector<edge> g;

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

4.3.2 Prim

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

struct edge {int v, dis;};

int n;
int dis[maxn];
int vis[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));
}

// 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;
}
```

4.4 NetFlow

4.4.1 Dinic

```
namespace NetFlow {
  const int maxn = 1e5 + 5;
  const int inf = 0x3f3f3f3f;
  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 ans = 0;
    while (Bfs(s, t)) {
      for (int i = s; i <= t; ++i) cur[i] = head[i];
      ans += Dfs(s, t, inf);
    }
    return ans;
  }
};
```

4.4.2 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;
}
```

4.4.3 MaxFlow

```
namespace NetFlow {
  const int inf = 0x3f3f3f3f;
  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 ans = 0;
    while (true) {
      vis.assign(t + 1, false);
      int flow = Dfs(s, t, inf);
      if (flow == 0) return ans;
      ans += flow;
    }
  }
};
```

4.4.4 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 Init() {
  tot = 0;
  memset(head, -1, sizeof(head));
}

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;
}
```

4.5 ShortestPath

4.5.1 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;
}
```

4.5.2 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 Init() {
  tot = 0;
  memset(head, -1, sizeof(head));
}

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

struct Cmp {
  bool operator() (const int &A, const int &B) {
    return dis[A] > dis[B];
  }
};
```

```
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);
      }
    }
  }
}
```

4.5.3 Floyd

```
const int maxn = "Edit";

// n: 顶点数
int n;
// dis[i][j] 为 i 点到 j 点的最短路
int dis[maxn][maxn];

// Floyd 算法
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]);
      }
    }
  }
}
```

4.5.4 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";

long long digit[25];
long long dp[25][maxn];

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

long long Get(long long x) {
  long long 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] 为长度为 i 的不下降子序列末尾元素的最小值
  std::vector<int> last(int(arr.size()) + 1, 0);
  last[1] = arr[1];
  for (int i = 2; i <= int(arr.size()); ++i) {
    if (arr[i] >= last[ret]) last[++ret] = arr[i];
    else {
      int pos = std::upper_bound(last.begin() + 1, last.end(), arr[i]) -
      ↪ last.begin();
      last[pos] = arr[i];
    }
  }
  // 返回结果
  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 JlsGeo

```
#define mp make_pair
#define fi first
#define se second
#define pb push_back
typedef double db;
const db eps=1e-6;
const db pi=acos(-1);
int sign(db k){
  if (k>eps) return 1; else if (k<-eps) return -1; return 0;
}
int cmp(db k1,db k2){return sign(k1-k2);}
int inmid(db k1,db k2,db k3){return sign(k1-k3)*sign(k2-k3)<=0;}// k3 在 [k1,k2] 内
 ↪
struct point{
  db x,y;
  point operator + (const point &k1) const{return (point){k1.x+x,k1.y+y};}
  point operator - (const point &k1) const{return (point){x-k1.x,y-k1.y};}
  point operator * (db k1) const{return (point){x*k1,y*k1};}
  point operator / (db k1) const{return (point){x/k1,y/k1};}
  int operator == (const point &k1) const{return cmp(x,k1.x)==0&&cmp(y,k1.y)==0;}
  // 逆时针旋转
  point turn(db k1){return (point){x*cos(k1)-y*sin(k1),x*sin(k1)+y*cos(k1)};}
  point turn90(){return (point){-y,x};}
  bool operator < (const point k1) const{
    int a=cmp(x,k1.x);
    if (a==-1) return 1; else if (a==1) return 0; else return cmp(y,k1.y)==-1;
  }
  db abs(){return sqrt(x*x+y*y);}
  db abs2(){return x*x+y*y;}
  db dis(point k1){return ((*this)-k1).abs();}
  point unit(){db w=abs(); return (point){x/w,y/w};}
  void scan(){double k1,k2; scanf("%lf%lf",&k1,&k2); x=k1; y=k2;}
  void print(){printf("%.11lf %.11lf\n",x,y);}
  db getw(){return atan2(y,x);}
  point getdel(){if (sign(x)==-1||(sign(x)==0&&sign(y)==-1)) return (*this)*(-1);
   ↪  else return (*this);}
        int getP() const{return sign(y)==1||(sign(y)==0&&sign(x)==-1);}
};
int inmid(point k1,point k2,point k3){return
 ↪  inmid(k1.x,k2.x,k3.x)&&inmid(k1.y,k2.y,k3.y);}
db cross(point k1,point k2){return k1.x*k2.y-k1.y*k2.x;}
db dot(point k1,point k2){return k1.x*k2.x+k1.y*k2.y;}
db rad(point k1,point k2){return atan2(cross(k1,k2),dot(k1,k2));}
// -pi -> pi
int compareangle (point k1,point k2){
  return k1.getP()<k2.getP()||(k1.getP()==k2.getP()&&sign(cross(k1,k2))>0);
```

```
}
point proj(point k1,point k2,point q){ // q 到直线 k1,k2 的投影
  point k=k2-k1; return k1+k*(dot(q-k1,k)/k.abs2());
}
point reflect(point k1,point k2,point q){return proj(k1,k2,q)*2-q;}
int clockwise(point k1,point k2,point k3){// k1 k2 k3 逆时针 1 顺时针 -1 否则 0
  return sign(cross(k2-k1,k3-k1));
}
int checkLL(point k1,point k2,point k3,point k4){// 求直线 (L) 线段 (S)k1,k2 和
↪  k3,k4 的交点
  return cmp(cross(k3-k1,k4-k1),cross(k3-k2,k4-k2))!=0;
}
point getLL(point k1,point k2,point k3,point k4){
  db w1=cross(k1-k3,k4-k3),w2=cross(k4-k3,k2-k3); return (k1*w2+k2*w1)/(w1+w2);
}
int intersect(db l1,db r1,db l2,db r2){
  if (l1>r1) swap(l1,r1); if (l2>r2) swap(l2,r2); return
  ↪  cmp(r1,l2)!=-1&&cmp(r2,l1)!=-1;
}
int checkSS(point k1,point k2,point k3,point k4){
  return intersect(k1.x,k2.x,k3.x,k4.x)&&intersect(k1.y,k2.y,k3.y,k4.y)&&
  sign(cross(k3-k1,k4-k1))*sign(cross(k3-k2,k4-k2))<=0&&
  sign(cross(k1-k3,k2-k3))*sign(cross(k1-k4,k2-k4))<=0;
}
db disSP(point k1,point k2,point q){
  point k3=proj(k1,k2,q);
  if (inmid(k1,k2,k3)) return q.dis(k3); else return min(q.dis(k1),q.dis(k2));
}
db disSS(point k1,point k2,point k3,point k4){
  if (checkSS(k1,k2,k3,k4)) return 0;
  else return
  ↪  min(min(disSP(k1,k2,k3),disSP(k1,k2,k4)),min(disSP(k3,k4,k1),disSP(k3,k4,k2)));
}
int onS(point k1,point k2,point q){return
↪  inmid(k1,k2,q)&&sign(cross(k1-q,k2-k1))==0;}
struct circle{
  point o; db r;
  void scan(){o.scan(); scanf("%lf",&r);}
  int inside(point k){return cmp(r,o.dis(k));}
};
struct line{
  // p[0]->p[1]
  point p[2];
  line(point k1,point k2){p[0]=k1; p[1]=k2;}
  point& operator [] (int k){return p[k];}
  int include(point k){return sign(cross(p[1]-p[0],k-p[0]))>0;}
  point dir(){return p[1]-p[0];}
  line push(){ // 向外 ( 左手边 ) 平移 eps
    const db eps = 1e-6;
    point delta=(p[1]-p[0]).turn90().unit()*eps;
    return {p[0]-delta,p[1]-delta};
  }
};
point getLL(line k1,line k2){return getLL(k1[0],k1[1],k2[0],k2[1]);}
int parallel(line k1,line k2){return sign(cross(k1.dir(),k2.dir()))==0;}
int sameDir(line k1,line k2){return
↪   parallel(k1,k2)&&sign(dot(k1.dir(),k2.dir()))==1;}
int operator < (line k1,line k2){
  if (sameDir(k1,k2)) return k2.include(k1[0]);
  return compareangle(k1.dir(),k2.dir());
}
int checkpos(line k1,line k2,line k3){return k3.include(getLL(k1,k2));}
vector<line> getHL(vector<line> &L){ // 求半平面交 , 半平面是逆时针方向 , 输出按照逆
↪   时针
  sort(L.begin(),L.end()); deque<line> q;
  for (int i=0;i<(int)L.size();i++){
    if (i&&sameDir(L[i],L[i-1])) continue;
    while (q.size()>1&&!checkpos(q[q.size()-2],q[q.size()-1],L[i])) q.pop_back();
    while (q.size()>1&&!checkpos(q[1],q[0],L[i])) q.pop_front();
    q.push_back(L[i]);
  }
  while (q.size()>2&&!checkpos(q[q.size()-2],q[q.size()-1],q[0])) q.pop_back();
  while (q.size()>2&&!checkpos(q[1],q[0],q[q.size()-1])) q.pop_front();
  vector<line>ans; for (int i=0;i<q.size();i++) ans.push_back(q[i]);
  return ans;
}
db closepoint(vector<point>&A,int l,int r){ // 最近点对 , 先要按照 x 坐标排序
  if (r-l<=5){
    db ans=1e20;
    for (int i=l;i<=r;i++) for (int j=i+1;j<=r;j++) ans=min(ans,A[i].dis(A[j]));
    return ans;
  }
  int mid=l+r>>1; db ans=min(closepoint(A,l,mid),closepoint(A,mid+1,r));
  vector<point>B; for (int i=l;i<=r;i++) if (abs(A[i].x-A[mid].x)<=ans)
  ↪   B.push_back(A[i]);
  sort(B.begin(),B.end(),[](point k1,point k2){return k1.y<k2.y;});
  for (int i=0;i<B.size();i++) for (int j=i+1;j<B.size()&&B[j].y-B[i].y<ans;j++)
  ↪   ans=min(ans,B[i].dis(B[j]));
  return ans;
}
int checkposCC(circle k1,circle k2){// 返回两个圆的公切线数量
  if (cmp(k1.r,k2.r)==-1) swap(k1,k2);
  db dis=k1.o.dis(k2.o);  int w1=cmp(dis,k1.r+k2.r),w2=cmp(dis,k1.r-k2.r);
  if (w1>0) return 4; else if (w1==0) return 3; else if (w2>0) return 2;
  else if (w2==0) return 1; else return 0;
}
vector<point> getCL(circle k1,point k2,point k3){ // 沿着 k2->k3 方向给出 , 相切给出
↪   两个
```

```
  point k=proj(k2,k3,k1.o); db d=k1.r*k1.r-(k-k1.o).abs2();
  if (sign(d)==-1) return {};
  point del=(k3-k2).unit()*sqrt(max((db)0.0,d)); return {k-del,k+del};
}
vector<point> getCC(circle k1,circle k2){// 沿圆 k1 逆时针给出 , 相切给出两个
  int pd=checkposCC(k1,k2); if (pd==0||pd==4) return {};
  db
  ↪  a=(k2.o-k1.o).abs2(),cosA=(k1.r*k1.r+a-k2.r*k2.r)/(2*k1.r*sqrt(max(a,(db)0.0)));
  db b=k1.r*cosA,c=sqrt(max((db)0.0,k1.r*k1.r-b*b));
  point k=(k2.o-k1.o).unit(),m=k1.o+k*b,del=k.turn90()*c;
  return {m-del,m+del};
}
vector<point> TangentCP(circle k1,point k2){// 沿圆 k1 逆时针给出
  db a=(k2-k1.o).abs(),b=k1.r*k1.r/a,c=sqrt(max((db)0.0,k1.r*k1.r-b*b));
  point k=(k2-k1.o).unit(),m=k1.o+k*b,del=k.turn90()*c;
  return {m-del,m+del};
}
vector<line> TangentoutCC(circle k1,circle k2){
  int pd=checkposCC(k1,k2); if (pd==0) return {};
  if (pd==1){point k=getCC(k1,k2)[0]; return {(line){k,k}};}
  if (cmp(k1.r,k2.r)==0){
    point del=(k2.o-k1.o).unit().turn90().getdel();
    return
    ↪  {(line){k1.o-del*k1.r,k2.o-del*k2.r},(line){k1.o+del*k1.r,k2.o+del*k2.r}};
  } else {
    point p=(k2.o*k1.r-k1.o*k2.r)/(k1.r-k2.r);
    vector<point>A=TangentCP(k1,p),B=TangentCP(k2,p);
    vector<line>ans; for (int i=0;i<A.size();i++) ans.push_back((line){A[i],B[i]});
    return ans;
  }
}
vector<line> TangentinCC(circle k1,circle k2){
  int pd=checkposCC(k1,k2); if (pd<=2) return {};
  if (pd==3){point k=getCC(k1,k2)[0]; return {(line){k,k}};}
  point p=(k2.o*k1.r+k1.o*k2.r)/(k1.r+k2.r);
  vector<point>A=TangentCP(k1,p),B=TangentCP(k2,p);
  vector<line>ans; for (int i=0;i<A.size();i++) ans.push_back((line){A[i],B[i]});
  return ans;
}
vector<line> TangentCC(circle k1,circle k2){
  int flag=0; if (k1.r<k2.r) swap(k1,k2),flag=1;
  vector<line>A=TangentoutCC(k1,k2),B=TangentinCC(k1,k2);
  for (line k:B) A.push_back(k);
  if (flag) for (line &k:A) swap(k[0],k[1]);
  return A;
}
db getarea(circle k1,point k2,point k3){
  // 圆 k1 与三角形 k2 k3 k1.o 的有向面积交
  point k=k1.o; k1.o=k1.o-k; k2=k2-k; k3=k3-k;
  int pd1=k1.inside(k2),pd2=k1.inside(k3);
  vector<point>A=getCL(k1,k2,k3);
  if (pd1>=0){
    if (pd2>=0) return cross(k2,k3)/2;
    return k1.r*k1.r*rad(A[1],k3)/2+cross(k2,A[1])/2;
  } else if (pd2>=0){
    return k1.r*k1.r*rad(k2,A[0])/2+cross(A[0],k3)/2;
  }else {
    int pd=cmp(k1.r,disSP(k2,k3,k1.o));
    if (pd<=0) return k1.r*k1.r*rad(k2,k3)/2;
    return cross(A[0],A[1])/2+k1.r*k1.r*(rad(k2,A[0])+rad(A[1],k3))/2;
  }
}
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,k1.dis(o)};
}
circle getScircle(vector<point> A){
  random_shuffle(A.begin(),A.end());
  circle ans=(circle){A[0],0};
  for (int i=1;i<A.size();i++)
    if (ans.inside(A[i])==-1){
      ans=(circle){A[i],0};
      for (int j=0;j<i;j++)
        if (ans.inside(A[j])==-1){
          ans.o=(A[i]+A[j])/2; ans.r=ans.o.dis(A[i]);
          for (int k=0;k<j;k++)
            if (ans.inside(A[k])==-1)
              ans=getcircle(A[i],A[j],A[k]);
        }
    }
  return ans;
}
db area(vector<point> A){ // 多边形用 vector<point> 表示 , 逆时针
  db ans=0;
  for (int i=0;i<A.size();i++) ans+=cross(A[i],A[(i+1)%A.size()]);
  return ans/2;
}
int checkconvex(vector<point>A){
  int n=A.size(); A.push_back(A[0]); A.push_back(A[1]);
  for (int i=0;i<n;i++) if (sign(cross(A[i+1]-A[i],A[i+2]-A[i]))==-1) return 0;
  return 1;
}
int contain(vector<point>A,point q){ // 2 内部 1 边界 0 外部
  int pd=0; A.push_back(A[0]);
  for (int i=1;i<A.size();i++){
    point u=A[i-1],v=A[i];
    if (onS(u,v,q)) return 1; if (cmp(u.y,v.y)>0) swap(u,v);
    if (cmp(u.y,q.y)>=0||cmp(v.y,q.y)<0) continue;
    if (sign(cross(u-v,q-v))<0) pd^=1;
  }
  return pd<<1;
}
vector<point> ConvexHull(vector<point>A,int flag=1){ // flag=0 不严格 flag=1 严格
  int n=A.size(); vector<point>ans(n*2);
  sort(A.begin(),A.end()); int now=-1;
  for (int i=0;i<A.size();i++){
    while (now>0&&sign(cross(ans[now]-ans[now-1],A[i]-ans[now-1]))<flag) now--;
    ans[++now]=A[i];
  } int pre=now;
  for (int i=n-2;i>=0;i--){
    while (now>pre&&sign(cross(ans[now]-ans[now-1],A[i]-ans[now-1]))<flag) now--;
    ans[++now]=A[i];
  } ans.resize(now); return ans;
}
db convexDiameter(vector<point>A){
  int now=0,n=A.size(); db ans=0;
  for (int i=0;i<A.size();i++){
    now=max(now,i);
    while (1){
      db k1=A[i].dis(A[now%n]),k2=A[i].dis(A[(now+1)%n]);
      ans=max(ans,max(k1,k2)); if (k2>k1) now++; else break;
    }
  }
  return ans;
}
vector<point> convexcut(vector<point>A,point k1,point k2){
  // 保留 k1,k2,p 逆时针的所有点
  int n=A.size(); A.push_back(A[0]); vector<point>ans;
  for (int i=0;i<n;i++){
    int w1=clockwise(k1,k2,A[i]),w2=clockwise(k1,k2,A[i+1]);
    if (w1>=0) ans.push_back(A[i]);
    if (w1*w2<0) ans.push_back(getLL(k1,k2,A[i],A[i+1]));
  }
  return ans;
}
int checkPoS(vector<point>A,point k1,point k2){
  // 多边形 A 和直线 ( 线段 )k1->k2 严格相交 , 注释部分为线段
  struct ins{
    point m,u,v;
    int operator < (const ins& k) const {return m<k.m;}
  }; vector<ins>B;
  //if (contain(A,k1)==2||contain(A,k2)==2) return 1;
  vector<point>poly=A; A.push_back(A[0]);
  for (int i=1;i<A.size();i++) if (checkLL(A[i-1],A[i],k1,k2)){
    point m=getLL(A[i-1],A[i],k1,k2);
    if (inmid(A[i-1],A[i],m)/*&&inmid(k1,k2,m)*/)
     ↪  B.push_back((ins){m,A[i-1],A[i]});
  }
  if (B.size()==0) return 0; sort(B.begin(),B.end());
  int now=1; while (now<B.size()&&B[now].m==B[0].m) now++;
  if (now==B.size()) return 0;
  int flag=contain(poly,(B[0].m+B[now].m)/2);
  if (flag==2) return 1;
  point d=B[now].m-B[0].m;
  for (int i=now;i<B.size();i++){
    if (!(B[i].m==B[i-1].m)&&flag==2) return 1;
    int tag=sign(cross(B[i].v-B[i].u,B[i].m+d-B[i].u));
    if (B[i].m==B[i].u||B[i].m==B[i].v) flag+=tag; else flag+=tag*2;
  }
  //return 0;
  return flag==2;
}
int checkinp(point r,point l,point m){
        if (compareangle(l,r)){return compareangle(l,m)&&compareangle(m,r);}
        return compareangle(l,m)||compareangle(m,r);
}
int checkPosFast(vector<point>A,point k1,point k2){ // 快速检查线段是否和多边形严格
 ↪  相交
        if (contain(A,k1)==2||contain(A,k2)==2) return 1; if (k1==k2) return 0;
        A.push_back(A[0]); A.push_back(A[1]);
        for (int i=1;i+1<A.size();i++)
                if (checkLL(A[i-1],A[i],k1,k2)){
                        point now=getLL(A[i-1],A[i],k1,k2);
                        if (inmid(A[i-1],A[i],now)==0||inmid(k1,k2,now)==0)
                         ↪  continue;
                        if (now==A[i]){
                                if (A[i]==k2) continue;
                                point pre=A[i-1],ne=A[i+1];
                                if (checkinp(pre-now,ne-now,k2-now)) return 1;
                        } else if (now==k1){
                                if (k1==A[i-1]||k1==A[i]) continue;
                                if (checkinp(A[i-1]-k1,A[i]-k1,k2-k1)) return 1;
                        } else if (now==k2||now==A[i-1]) continue;
                        else return 1;
                }
        return 0;
}
// 拆分凸包成上下凸壳 凸包尽量都随机旋转一个角度来避免出现相同横坐标
// 尽量特判只有一个点的情况 凸包逆时针
void getUDP(vector<point>A,vector<point>&U,vector<point>&D){
  db l=1e100,r=-1e100;
  for (int i=0;i<A.size();i++) l=min(l,A[i].x),r=max(r,A[i].x);
  int wherel,wherer;
  for (int i=0;i<A.size();i++) if (cmp(A[i].x,l)==0) wherel=i;
  for (int i=A.size();i;i--) if (cmp(A[i-1].x,r)==0) wherer=i-1;
  U.clear(); D.clear(); int now=wherel;
  while (1){D.push_back(A[now]); if (now==wherer) break; now++; if (now>=A.size())
   ↪  now=0;}
  now=wherel;
  while (1){U.push_back(A[now]); if (now==wherer) break; now--; if (now<0)
   ↪  now=A.size()-1;}
}
// 需要保证凸包点数大于等于 3,2 内部 ,1 边界 ,0 外部
int containCoP(const vector<point>&U,const vector<point>&D,point k){
  db lx=U[0].x,rx=U[U.size()-1].x;
  if (k==U[0]||k==U[U.size()-1]) return 1;
  if (cmp(k.x,lx)==-1||cmp(k.x,rx)==1) return 0;
  int where1=lower_bound(U.begin(),U.end(),(point){k.x,-1e100})-U.begin();
  int where2=lower_bound(D.begin(),D.end(),(point){k.x,-1e100})-D.begin();
  int w1=clockwise(U[where1-1],U[where1],k),w2=clockwise(D[where2-1],D[where2],k);
  if (w1==1||w2==-1) return 0; else if (w1==0||w2==0) return 1; return 2;
}
// d 是方向 , 输出上方切点和下方切点
pair<point,point> getTangentCow(const vector<point> &U,const vector<point> &D,point
 ↪  d){
  if (sign(d.x)<0||(sign(d.x)==0&&sign(d.y)<0)) d=d*(-1);
  point whereU,whereD;
  if (sign(d.x)==0) return mp(U[0],U[U.size()-1]);
  int l=0,r=U.size()-1,ans=0;
  while (l<r){int mid=l+r>>1; if (sign(cross(U[mid+1]-U[mid],d))<=0)
   ↪  l=mid+1,ans=mid+1; else r=mid;}
  whereU=U[ans]; l=0,r=D.size()-1,ans=0;
  while (l<r){int mid=l+r>>1; if (sign(cross(D[mid+1]-D[mid],d))>=0)
   ↪  l=mid+1,ans=mid+1; else r=mid;}
  whereD=D[ans]; return mp(whereU,whereD);
}
// 先检查 contain, 逆时针给出
pair<point,point> getTangentCoP(const vector<point>&U,const vector<point>&D,point
 ↪  k){
  db lx=U[0].x,rx=U[U.size()-1].x;
  if (k.x<lx){
    int l=0,r=U.size()-1,ans=U.size()-1;
    while (l<r){int mid=l+r>>1; if (clockwise(k,U[mid],U[mid+1])==1) l=mid+1; else
     ↪  ans=mid,r=mid;}
    point w1=U[ans]; l=0,r=D.size()-1,ans=D.size()-1;
    while (l<r){int mid=l+r>>1; if (clockwise(k,D[mid],D[mid+1])==-1) l=mid+1; else
     ↪  ans=mid,r=mid;}
    point w2=D[ans]; return mp(w1,w2);
  } else if (k.x>rx){
    int l=1,r=U.size(),ans=0;
    while (l<r){int mid=l+r>>1; if (clockwise(k,U[mid],U[mid-1])==-1) r=mid; else
     ↪  ans=mid,l=mid+1;}
```

```
      point w1=U[ans]; l=1,r=D.size(),ans=0;
      while (l<r){int mid=l+r>>1; if (clockwise(k,D[mid],D[mid-1])==1) r=mid; else
       ↪  ans=mid,l=mid+1;}
      point w2=D[ans]; return mp(w2,w1);
    } else {
      int where1=lower_bound(U.begin(),U.end(),(point){k.x,-1e100})-U.begin();
      int where2=lower_bound(D.begin(),D.end(),(point){k.x,-1e100})-D.begin();
      if ((k.x==lx&&k.y>U[0].y)||(where1&&clockwise(U[where1-1],U[where1],k)==1)){
        int l=1,r=where1+1,ans=0;
        while (l<r){int mid=l+r>>1; if (clockwise(k,U[mid],U[mid-1])==1)
         ↪  ans=mid,l=mid+1; else r=mid;}
        point w1=U[ans]; l=where1,r=U.size()-1,ans=U.size()-1;
        while (l<r){int mid=l+r>>1; if (clockwise(k,U[mid],U[mid+1])==1) l=mid+1;
         ↪  else ans=mid,r=mid;}
        point w2=U[ans]; return mp(w2,w1);
      } else {
        int l=1,r=where2+1,ans=0;
        while (l<r){int mid=l+r>>1; if (clockwise(k,D[mid],D[mid-1])==-1)
         ↪  ans=mid,l=mid+1; else r=mid;}
        point w1=D[ans]; l=where2,r=D.size()-1,ans=D.size()-1;
        while (l<r){int mid=l+r>>1; if (clockwise(k,D[mid],D[mid+1])==-1) l=mid+1;
         ↪  else ans=mid,r=mid;}
        point w2=D[ans]; return mp(w1,w2);
      }
    }
}
struct P3{
  db x,y,z;
  P3 operator + (P3 k1){return (P3){x+k1.x,y+k1.y,z+k1.z};}
  P3 operator - (P3 k1){return (P3){x-k1.x,y-k1.y,z-k1.z};}
  P3 operator * (db k1){return (P3){x*k1,y*k1,z*k1};}
  P3 operator / (db k1){return (P3){x/k1,y/k1,z/k1};}
  db abs2(){return x*x+y*y+z*z;}
  db abs(){return sqrt(x*x+y*y+z*z);}
  P3 unit(){return (*this)/abs();}
  int operator < (const P3 k1) const{
    if (cmp(x,k1.x)!=0) return x<k1.x;
    if (cmp(y,k1.y)!=0) return y<k1.y;
    return cmp(z,k1.z)==-1;
  }
  int operator == (const P3 k1){
    return cmp(x,k1.x)==0&&cmp(y,k1.y)==0&&cmp(z,k1.z)==0;
  }
  void scan(){
    double k1,k2,k3; scanf("%lf%lf%lf",&k1,&k2,&k3);
    x=k1; y=k2; z=k3;
  }
};
P3 cross(P3 k1,P3 k2){return
 ↪  (P3){k1.y*k2.z-k1.z*k2.y,k1.z*k2.x-k1.x*k2.z,k1.x*k2.y-k1.y*k2.x};}
```

```
db dot(P3 k1,P3 k2){return k1.x*k2.x+k1.y*k2.y+k1.z*k2.z;}
//p=(3,4,5),l=(13,19,21),theta=85 ans=(2.83,4.62,1.77)
P3 turn3D(db k1,P3 l,P3 p){
  l=l.unit(); P3 ans; db c=cos(k1),s=sin(k1);
  ans.x=p.x*(l.x*l.x*(1-c)+c)+p.y*(l.x*l.y*(1-c)-l.z*s)+p.z*(l.x*l.z*(1-c)+l.y*s);
  ans.y=p.x*(l.x*l.y*(1-c)+l.z*s)+p.y*(l.y*l.y*(1-c)+c)+p.z*(l.y*l.z*(1-c)-l.x*s);
  ans.z=p.x*(l.x*l.z*(1-c)-l.y*s)+p.y*(l.y*l.z*(1-c)+l.x*s)+p.z*(l.x*l.x*(1-c)+c);
  return ans;
}
typedef vector<P3> VP;
typedef vector<VP> VVP;
db Acos(db x){return acos(max(-(db)1,min(x,(db)1)));}
// 球面距离 , 圆心原点 , 半径 1
db Odist(P3 a,P3 b){db r=Acos(dot(a,b)); return r;}
db r; P3 rnd;
vector<db> solve(db a,db b,db c){
  db r=sqrt(a*a+b*b),th=atan2(b,a);
  if (cmp(c,-r)==-1) return {0};
  else if (cmp(r,c)<=0) return {1};
  else {
    db tr=pi-Acos(c/r); return {th+pi-tr,th+pi+tr};
  }
}
vector<db> jiao(P3 a,P3 b){
  // dot(rd+x*cos(t)+y*sin(t),b) >= cos(r)
  if (cmp(Odist(a,b),2*r)>0) return {0};
  P3 rd=a*cos(r),z=a.unit(),y=cross(z,rnd).unit(),x=cross(y,z).unit();
  vector<db> ret =
  ↪  solve(-(dot(x,b)*sin(r)),-(dot(y,b)*sin(r)),-(cos(r)-dot(rd,b)));
  return ret;
}
db norm(db x,db l=0,db r=2*pi){ // change x into [l,r)
  while (cmp(x,l)==-1) x+=(r-l); while (cmp(x,r)>=0) x-=(r-l);
  return x;
}
db disLP(P3 k1,P3 k2,P3 q){
  return (cross(k2-k1,q-k1)).abs()/(k2-k1).abs();
}
db disLL(P3 k1,P3 k2,P3 k3,P3 k4){
  P3 dir=cross(k2-k1,k4-k3); if (sign(dir.abs())==0) return disLP(k1,k2,k3);
  return fabs(dot(dir.unit(),k1-k2));
}
VP getFL(P3 p,P3 dir,P3 k1,P3 k2){
  db a=dot(k2-p,dir),b=dot(k1-p,dir),d=a-b;
  if (sign(fabs(d))==0) return {};
  return {(k1*a-k2*b)/d};
}
VP getFF(P3 p1,P3 dir1,P3 p2,P3 dir2){// 返回一条线
  P3 e=cross(dir1,dir2),v=cross(dir1,e);
  db d=dot(dir2,v); if (sign(abs(d))==0) return {};
  P3 q=p1+v*dot(dir2,p2-p1)/d; return {q,q+e};
}
// 3D Covex Hull Template
db getV(P3 k1,P3 k2,P3 k3,P3 k4){ // get the Volume
  return dot(cross(k2-k1,k3-k1),k4-k1);
}
db rand_db(){return 1.0*rand()/RAND_MAX;}
VP convexHull2D(VP A,P3 dir){
  P3 x={(db)rand(),(db)rand(),(db)rand()}; x=x.unit();
  x=cross(x,dir).unit(); P3 y=cross(x,dir).unit();
  P3 vec=dir.unit()*dot(A[0],dir);
  vector<point>B;
  for (int i=0;i<A.size();i++) B.push_back((point){dot(A[i],x),dot(A[i],y)});
  B=ConvexHull(B); A.clear();
  for (int i=0;i<B.size();i++) A.push_back(x*B[i].x+y*B[i].y+vec);
  return A;
}
namespace CH3{
  VVP ret; set<pair<int,int> >e;
  int n; VP p,q;
  void wrap(int a,int b){
    if (e.find({a,b})==e.end()){
      int c=-1;
      for (int i=0;i<n;i++) if (i!=a&&i!=b){
        if (c==-1||sign(getV(q[c],q[a],q[b],q[i]))>0) c=i;
      }
      if (c!=-1){
        ret.push_back({p[a],p[b],p[c]});
        e.insert({a,b}); e.insert({b,c}); e.insert({c,a});
        wrap(c,b); wrap(a,c);
      }
    }
  }
  VVP ConvexHull3D(VP _p){
    p=q=_p; n=p.size();
    ret.clear(); e.clear();
    for (auto &i:q) i=i+(P3){rand_db()*1e-4,rand_db()*1e-4,rand_db()*1e-4};
    for (int i=1;i<n;i++) if (q[i].x<q[0].x) swap(p[0],p[i]),swap(q[0],q[i]);
    for (int i=2;i<n;i++) if
    ↪  ((q[i].x-q[0].x)*(q[1].y-q[0].y)>(q[i].y-q[0].y)*(q[1].x-q[0].x))
    ↪  swap(q[1],q[i]),swap(p[1],p[i]);
    wrap(0,1);
    return ret;
  }
}
VVP reduceCH(VVP A){
  VVP ret; map<P3,VP> M;
  for (VP nowF:A){
    P3 dir=cross(nowF[1]-nowF[0],nowF[2]-nowF[0]).unit();
    for (P3 k1:nowF) M[dir].pb(k1);
  }
  for (pair<P3,VP> nowF:M) ret.pb(convexHull2D(nowF.se,nowF.fi));
  return ret;
}
// 把一个面变成 ( 点 , 法向量 ) 的形式
pair<P3,P3> getF(VP F){
  return mp(F[0],cross(F[1]-F[0],F[2]-F[0]).unit());
}
// 3D Cut 保留 dot(dir,x-p)>=0 的部分
VVP ConvexCut3D(VVP A,P3 p,P3 dir){
  VVP ret; VP sec;
  for (VP nowF: A){
    int n=nowF.size(); VP ans; int dif=0;
    for (int i=0;i<n;i++){
      int d1=sign(dot(dir,nowF[i]-p));
      int d2=sign(dot(dir,nowF[(i+1)%n]-p));
      if (d1>=0) ans.pb(nowF[i]);
      if (d1*d2<0){
        P3 q=getFL(p,dir,nowF[i],nowF[(i+1)%n])[0];
        ans.push_back(q); sec.push_back(q);
      }
      if (d1==0) sec.push_back(nowF[i]); else dif=1;
      dif|=(sign(dot(dir,cross(nowF[(i+1)%n]-nowF[i],nowF[(i+1)%n]-nowF[i])))==-1);
    }
    if (ans.size()>0&&dif) ret.push_back(ans);
  }
  if (sec.size()>0) ret.push_back(convexHull2D(sec,dir));
  return ret;
}
db vol(VVP A){
  if (A.size()==0) return 0; P3 p=A[0][0]; db ans=0;
  for (VP nowF:A)
    for (int i=2;i<nowF.size();i++)
      ans+=abs(getV(p,nowF[0],nowF[i-1],nowF[i]));
  return ans/6;
}
VVP init(db INF) {
  VVP pss(6,VP(4));
  pss[0][0] = pss[1][0] = pss[2][0] = {-INF, -INF, -INF};
  pss[0][3] = pss[1][1] = pss[5][2] = {-INF, -INF, INF};
  pss[0][1] = pss[2][3] = pss[4][2] = {-INF, INF, -INF};
  pss[0][2] = pss[5][3] = pss[4][1] = {-INF, INF, INF};
  pss[1][3] = pss[2][1] = pss[3][2] = {INF, -INF, -INF};
  pss[1][2] = pss[5][1] = pss[3][3] = {INF, -INF, INF};
  pss[2][2] = pss[4][3] = pss[3][1] = {INF, INF, -INF};
  pss[5][0] = pss[4][0] = pss[3][0] = {INF, INF, INF};
  return pss;
```

```
}
```

6.2 Plane

```
namespace Geometry {
  typedef double db;
  const db inf = 1e20;
  const int maxn = 1;
  const db eps = 1e-8;
  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;}

  /*----------点 (向量)----------*/
  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);}
  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(std::vector<point> points) {
    int N = (int)points.size();
    for (int i = 0; i < N; ++i)
      if (Sgn((points[(i + 1) % N] - points[i]) ^ (points[(i + 2) % N] - points[(i
      ↪  + 1) % N])) < 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 ans = min(solve(l, mid), solve(mid + 1, r));
    std::vector<point> mid_p;
    for (int i = l; i <= r; ++i) {
```

```
    if (Cmp(fabs(p[i].x - p[mid].x), ans) <= 0) mid_p.push_back(p[i]);
  }
  sort(mid_p.begin(), mid_p.end(), [&](point k1, point k2) {return Cmp(k1.y,
  ↪  k2.y) < 0;});
  for (int i = 0; i < mid_p.size(); ++i) {
    for (int j = i + 1; j < mid_p.size(); ++j) {
      if (Cmp(mid_p[j].y - mid_p[i].y, ans) >= 0) break;
      ans = min(ans, GetDisP2P(mid_p[i], mid_p[j]));
    }
  }
  return ans;
}

/*----------多边形----------*/
typedef std::vector<point> poly;
void RotateCaliper() {
  ans = -1e20;
  if (ConvexHull.size() == 3) {
    if (Cmp(GetDisP2P(ConvexHull[0], ConvexHull[1]), ans) > 0) ans =
    ↪  GetDisP2P(ConvexHull[0], ConvexHull[1]);
    if (Cmp(GetDisP2P(ConvexHull[0], ConvexHull[2]), ans) > 0) ans =
    ↪  GetDisP2P(ConvexHull[0], ConvexHull[2]);
    if (Cmp(GetDisP2P(ConvexHull[1], ConvexHull[2]), ans) > 0) ans =
    ↪  GetDisP2P(ConvexHull[1], ConvexHull[2]);
    return;
  }
  int cur = 2, size = ConvexHull.size();
  for (int i = 0; i < size; ++i) {
    while (Cmp(fabs((ConvexHull[i] - ConvexHull[(i + 1) % size]) ^
    ↪  (ConvexHull[cur] - ConvexHull[(i + 1) % size])), fabs((ConvexHull[i] -
    ↪  ConvexHull[(i + 1) % size]) ^ (ConvexHull[(cur + 1) % size] -
    ↪  ConvexHull[(i + 1) % size]))) < 0) cur = (cur + 1) % size;
    if (Cmp(GetDisP2P(ConvexHull[i], ConvexHull[cur]), ans) > 0) ans =
    ↪  GetDisP2P(ConvexHull[i], ConvexHull[cur]);
  }
}

poly Grahamscan(std::vector<point> p) {
  poly ans;
  if ((int)p.size() < 3) {
    for (int i = 0; i < (int)p.size(); ++i) ans.push_back(p[i]);
    return ans;
  }
  int Basic = 0;
  for (int i = 0; i < (int)p.size(); ++i)
    if (Cmp(p[i].X, p[Basic].X) < 0 || (Cmp(p[i].X, p[Basic].X) == 0 &&
    ↪  Cmp(p[i].Y, p[Basic].Y) < 0))
      Basic = i;
  std::swap(p[0], p[Basic]);
```

```
    std::sort(p.begin() + 1, p.end(), [&](point k1, point k2) {
      double temp = (k1 - p[0]) ^ (k2 - p[0]);
      if (Sgn(temp) > 0) return true;
      else if (Sgn(temp) == 0 && Cmp(GetDisP2P(k2, p[0]), GetDisP2P(k1, p[0])) > 0)
       ↪  return true;
      return false;
    });
    ans.push_back(p[0]);
    for (int i = 1; i < (int)p.size(); ++i) {
      while ((int)ans.size() >= 2 && Sgn((ans.back() - ans[(ans.size()) - 2]) ^
       ↪  (p[i] - ans[(int)ans.size() - 2])) <= 0) {
        ans.pop_back();
      }
      ans.push_back(p[i]);
    }
    return ans;
  }

  db GetMinCircle(std::vector<point> p) {
    point cur = p[0];
    db pro = 10000, ans = 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, ans) < 0) ans = r;
      cur = cur + (p[Book] - cur) / r * pro;
      pro *= delta;
    }
    return ans;
  }

  /*----------线 (线段)----------*/
  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};
}

/*----------半平面----------*/
// 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]);
  }
};

/*----------圆----------*/
struct circle {point o; db r;};

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) {
    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;
```

6.3 Simpson

```
typedef double db;

namespace Simpson {
  db a, b, c, d;

  db F(db x) {
    return (c * x + d) / (a * x + b);
  }

  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);
  }
};
```

6.4 Stereoscopic

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

namespace Geometry3D {
```

```
typedef double db;
const db inf = 1e20;
const int maxn = "Edit";
const db eps = 1e-9;
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);}

/*----------点 (向量)----------*/
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(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;
}

/*----------线 (线段)----------*/
struct line {point s, t;};
```

```
typedef line seg;
db GetLen(seg k) {return GetDisP2P(k.s, k.t);}
db DisP2Line(point k1, line k2) {return fabs((k1 - k2.s) ^ (k2.t - k2.s)) /
↪  GetLen(k2);}
db DisP2Seg(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 DisP2Line(k1, k2);
}

/*----------球----------*/
struct sphere {point o;db r;};
db GetVolume(sphere k) {return 4.0 / 3.0 * pi * k.r * k.r * k.r;}
db GetSphereInterVolume(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 CalVolume(k1);
  else if (Sgn(k1.r - (dis + k2.r)) >= 0) return CalVolume(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 FastIO

```
// 普通读入挂
template <class 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 <class 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;
```

7.2 LeepYear

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

7.3 vim

```
syntax on
set nu
set tabstop=2
set shiftwidth=2
set cindent
set mouse=a
set expandtab
" set backspace=indent,eol,start

"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, "using namespace std;")
"    let l = l + 1 | call setline(l, "")
"    let l = l + 1 | call setline(l, "int main(int argc, char *argv[]) {")
"    let l = l + 1 | call setline(l, "    return 0;")
"    let l = l + 1 | call setline(l, "}")
"    let l = l + 1 | call setline(l, "")
"endfunc
```