Algorithm Library

Liu Yang

March 22, 2019

Contents

1 String

1.1 AhoCorasickAutomaton

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

const int maxn = "Edit";

struct AhoCorasickAutomaton {
    // 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, Ans = 0;
        for (int i = 0; i < Len; ++i) {
            Cur = Son[Cur][Pos(Str[i])];
            for (int j = Cur; j && ~Val[j]; j = Fail[j]) {
                Ans += Val[j];
                Val[j] = -1;
            }
        }
        return Ans;
    }
};
```

1.2 KMP

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

// 对模式串 Pattern 计算 Next 数组
void KMPPre(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(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(string Pattern, string Main) {
    int PatternLen = int(Pattern.length()), MainLen = int(Main.length());
    vector<int> Next(PatternLen + 1, 0);
    //PreKMP(Pattern, Next);
    KMPPre(Pattern, Next);
    int i = 0, j = 0;
    int Ans = 0;
    while (i < MainLen) {
        while (j != -1 && Main[i] != Pattern[j]) {
            j = Next[j];
        }
        i++; j++;
        if (j >= PatternLen) {
            Ans++;
            j = Next[j];
        }
    }
    return Ans;
}
```

1.3 Manacher

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

const int maxn = "Edit";

char ConvertStr[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, StrLen = int(strlen(Str));
    ConvertStr[L++] = '$'; ConvertStr[L++] = '#';
    for (int i = 0; i < StrLen; ++i) {
        ConvertStr[L++] = Str[i];
        ConvertStr[L++] = '#';
    }
    int MX = 0, ID = 0, Ans = 0;
    for (int i = 0; i < L; ++i) {
        Len[i] = MX > i ? min(Len[2 * ID - i], MX - i) : 1;
        while (ConvertStr[i + Len[i]] == ConvertStr[i - Len[i]]) {
            Len[i]++;
        }
        if (i + Len[i] > MX) {
            MX = i + Len[i];
            ID = i;
        }
        Ans = max(Ans, Len[i] - 1);
    }
    return Ans;
}
```

1.4 PalindromicTree

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

const int maxn = "Edit";

struct PalindromicTree {
    // 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 StrLen;
    // 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;
        StrLen = 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[StrLen - Len[X] - 1] != Str[StrLen]) {
            X = Fail[X];
        }
        return X;
    }

    void Add(long long Char) {
        Char -= 'a';
        Str[++StrLen] = Char;
        long long Cur = GetFail(Last);
        if (!Son[Cur][Char]) {
            long long New = NewNode(Len[Cur] + 2);
            Fail[New] = Son[GetFail(Fail[Cur])][Char];
            Son[Cur][Char] = New;
            Num[New] = Num[Fail[New]] + 1;
        }
        Last = Son[Cur][Char];
        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

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

const int maxn = "Edit";

struct Trie {
    // 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

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

const int maxn = "Edit";

long long Catalan[maxn];

// 递推求卡特兰数
void CalalanInit() {
    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

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

const int maxn = "Edit";
const int mod = 1e9 + 7;

// Staggered: 错排数
long long Staggered[maxn];

// 求错排数
void StaggeredInit() {
    Staggered[1] = 0;
    Staggered[2] = 1;
    // 递推求错排数
    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

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

// 单独求解欧拉函数
int Phi(int X) {
    int Ans = X;
    for (int i = 2; i * i <= X; ++i) {
        if (!(X % i)) {
            Ans = Ans / i * (i - 1);
            while (!(X % i)) {
                X /= i;
            }
        }
    }
    if (X > 1) {
        Ans = Ans / X * (X - 1);
    }
    return Ans;
}
```

2.3.2 Screen

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

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

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

const int maxn = "Edit";

// 素数标记
bool IsPrime[maxn];
// 欧拉函数
int Phi[maxn];
// 素数
int Prime[maxn];
// 素数个数
```

```
int Tot;

// 同时求得欧拉函数和素数表
void PhiPrime() {
    memset(IsPrime, false, sizeof(IsPrime));
    Phi[1] = 1;
    Tot = 0;
    for (int i = 2; i < maxn; ++i) {
        if (!IsPrime[i]) {
            Prime[Tot++] = i;
            Phi[i] = i - 1;
        }
        for (int j = 0; j < Tot && i * Prime[j] < maxn; ++j) {
            IsPrime[i * Prime[j]] = true;
            if (!(i % Prime[j])) {
                Phi[i * Prime[j]] = Phi[i] * Prime[j];
                break;
            }
            else {
                Phi[i * Prime[j]] = Phi[i] * (Prime[j] - 1);
            }
        }
    }
}
```

2.4 FFT

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

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 Temp = B.X * B.X + B.Y * B.Y;
        return Complex {(X * B.X + Y * B.Y) / Temp, (Y * B.X - X * B.Y) / Temp};
    }
};

// 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 Temp = 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 * Temp;
            }
        }
    }
}

// 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

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

const int mod = 1e9 + 7;

// 矩阵结构体
struct Matrix {
    // 矩阵
    long long Mat[2][2];
};

// 重载矩阵乘法
Matrix operator * (Matrix &Key1, Matrix &Key2) const {
    Matrix Res;
    memset(Res.Mat, 0, sizeof(Res.Mat));
    for (int i = 0; i < 2; ++i) {
        for (int j = 0; j < 2; ++j) {
            for (int k = 0; k < 2; ++k) {
                Res.Mat[i][j] = (Res.Mat[i][j] + Key1.Mat[i][k] * Key2.Mat[k][j] %
                 ↪  mod) % mod;
            }
        }
    }
    return Res;
}

// 重载矩阵快速幂
Matrix operator ^ (Matrix Base, long long K) {
    Matrix Res;
    memset(Res.Mat, 0, sizeof(Res.Mat));
    Res.Mat[0][0] = Res.Mat[1][1] = 1;
    while (K) {
        if (K & 1) {
            Res = Res * Base;
        }
        Base = Base * Base;
        K >>= 1;
    }
    return Res;
}

// 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

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

const int maxn = "Edit";

void GeneratingFunction() {
    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

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

// 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 Inv(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

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

const int mod = 1e9 + 7;
const int maxn = "Edit";

// 快速乘
long long QuickMul(long long A, long long B) {
    long long Ans = 0;
    while (B) {
        if (B & 1) {
            Ans = (Ans + A) % mod;
        }
        A = (A + A) % mod;
        B >>= 1;
    }
    return Ans;
}

// 快速幂
long long QuickPow(long long A, long long B) {
    long long Ans = 1;
    while (B) {
        if (B & 1) {
            Ans = QuickMul(Ans, A) % mod;
        }
        A = QuickMul(A, A) % mod;
        B >>= 1;
    }
    return Ans;
}

// Factorial: 阶乘, FactorialInv: 阶乘逆元
long long Factorial[maxn], FactorialInv[maxn];
```

```
// 求阶乘逆元
void FactorialInvInit() {
    // 求阶乘
    Factorial[0] = 0;
    Factorial[1] = 1;
    for (int i = 2; i < maxn; ++i) {
        Factorial[i] = (Factorial[i - 1] * i) % mod;
    }
    // 飞马小定理求最大值阶乘逆元
    FactorialInv[maxn - 1] = QuickPow(Factorial[maxn - 1], mod - 2);
    // 递推求阶乘逆元
    for (int i = maxn - 2; i >= 0; --i) {
        FactorialInv[i] = (FactorialInv[i + 1] * (i + 1)) % mod;
    }
}
```

2.7.3 FermatLittleTheorem

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

const int mod = 1e9 + 7;

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

2.7.4 Recursive

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

const int mod = 1e9 + 7;
const int maxn = "Edit";

long long Inv[maxn];

// 递推求逆元
void InvInit() {
    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];
vector<int> prime;
int mu[maxn];

void Sieve() {
  memset(is_prime, true, sizeof(is_prime));
  mu[1] = 1;
  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

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

// 尼姆博弈
bool Nim(std::vector<int> Num) {
    int Ans = 0;
    for (int i = 0; i < int(Num.size()); ++i) {
        Ans ^= Num[i];
    }
    // ans 不为零则先手赢, 否则为后手赢
    return Ans != 0 ? true : false;
}
```

2.10 Prime

2.10.1 PrimeFactor

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

const int maxn = "Edit"

bool IsPrime[maxn];
vector<int> PrimeFactor[maxn];

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

2.10.2 SieveOfEratosthenes

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

typedef long long ll;
const int maxn = "Edit";

bool is_prime[maxn];
vector<int> prime

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

2.11 QuickPow

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

const int mod = 1e9 + 7;

// 快速乘求 A*B%mod
long long QuickMul(long long A, long long B) {
    long long Ans = 0;
    while (B) {
        if (B & 1) {
            Ans = (Ans + A) % mod;
        }
        A = (A + A) % mod;
        B >>= 1;
    }
    return Ans;
}

// 快速幂求 A^B%mod
long long QuickPow(long long A, long long B) {
    long long Ans = 1;
    while (B) {
        if (B & 1) {
            // Ans = Ans * A % mod;
            Ans = QuickMul(Ans, A) % mod;
        }
        // A = A * A % mod;
        A = QuickMul(A, A) % mod;
        B >>= 1;
    }
    return Ans;
}
```

2.12 Stirling

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

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

int Stirling(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";

namespace binary_indexed_tree {
    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

```
const int maxn = "Edit";

vector<vector<int>> g;

int dfs_clock;
int in[maxn], out[maxn];

// Dfs 序
void DfsOrder(int cur, int pre) {
    dfs_clock++;
    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";
```

```
namespace FuncSegTree {
    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
namespace Hash {
    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
const int maxn = "Edit"

namespace SegTree {
    int n;
    long long sum[maxn << 2], lazy[maxn << 2];

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

    void Push(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;
      }
    }

    template <typename t>
    void Build(int o, int l, int r, const vector<t> &v) {
      sum[o] = 0; lazy[o] = 0;
      if (l == r) {
        sum[o] = 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>
    void Init(const vector<t> &v) {
      n = v.size();
      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) {
        sum[o] += (r - l + 1) * v;
        lazy[o] += 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);
    }

    long long Query(int o, int l, int r, int ll, int rr) {
      if (ll <= l && rr >= r) return sum[o];
      Push(o, l, r);
      int m = (l + r) >> 1;
      long long ans = 0;
      if (ll <= m) ans += Query(o << 1, l, m, ll, rr);
      if (rr > m) ans += Query(o << 1 | 1, m + 1, r, ll, rr);
      return ans;
    }
    long long Query(int ll, int rr) {
      return Query(1, 1, n, ll, rr);
    }
};
```

3.7 SplayTree

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

namespace 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]); 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];
      }
    }
  }

  int GetPrev() {
    int cur = son[rt][0];
    while (son[cur][1]) cur = son[cur][1];
    return cur;
  }
  int GetPrevVal(int x) {
    int ans = -inf, cur = rt;
    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 cur = son[rt][1];
    while (son[cur][0]) cur = son[cur][0];
    return cur;
  }
  int GetNextVal(int x) {
    int ans = inf, cur = rt;
    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 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 LCA

4.1.1 DFS+ST

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

const int maxn = "Edit";

// 链式前向星存图
struct Edge {
    int V, Weight, Next;
};

Edge edges[maxn << 1];
int Head[maxn];
int Tot;

void Init() {
    Tot = 0;
    memset(Head, -1, sizeof(Head));
}

void AddEdge(int U, int V, int Weight) {
    edges[Tot] = Edge {V, Weight, Head[U]};
    Head[U] = Tot++;
}

struct 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 Parent[maxn];
    // 记录与根节点距离
    int Dis[maxn];
    // 遍历节点数量
    int LCATot;

    // 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 Depth) {
        Vertex[++LCATot] = Cur;
        First[Cur] = LCATot;
        Rmq[LCATot] = Depth;
        Parent[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].Weight;
            Dfs(edges[i].V, Cur, Depth + 1);
            Vertex[++LCATot] = Cur;
            Rmq[LCATot] = Depth;
        }
    }

    // RMQ 查询
    int Query(int Left, int Right) {
        if (Left > Right) {
            swap(Left, Right);
        }
        int Len = int(log2(Right - Left + 1));
        return Rmq[Dp[Left][Len]] <= Rmq[Dp[Right - (1 << Len) + 1][Len]] ?
         ↪  Dp[Left][Len] : Dp[Right - (1 << Len) + 1][Len];
    }

    // LCA 初始化
    void Init(int Root, int NodeNum) {
        memset(Dis, 0, sizeof(Dis));
        LCATot = 0;
        Dfs(Root, 0, 0);
        Parent[1] = 0;
        Work(2 * NodeNum - 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 LCA(int U, int V) {
        return Vertex[Query(First[U], First[V])];
    }
}LCA;
```

4.1.2 Tarjan

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

const int maxn = "Edit";

// 树边
struct Edge {
    int V, Next;
};

// 询问
struct Query {
    int Q, Next;
    int Index;
};

// 并查集数组
int Pre[maxn << 2];
// 树边
Edge edges[maxn << 2];
int Head[maxn];
int Tot;
// 询问
Query querys[maxn << 2];
int QHead[maxn];
int QTot;
// 访问标记
int Vis[maxn];
int Ancestor[maxn];
// 结果
int Answer[maxn];

// 并查集查找
int Find(int X) {
    int R = X;
    while (Pre[R] != -1) {
        R = Pre[R];
    }
    return R;
}

// 并查集合并
void Join(int U, int V) {
    int RU = Find(U);
    int RV = Find(V);
    if (RU != RV) {
        Pre[RU] = RV;
    }
}

// 添加树边
void AddEdge(int U, int V) {
    edges[Tot] = Edge {V, Head[U]};
    Head[U] = Tot++;
}

// 添加询问
void AddQuery(int U, int V, int Index) {
    querys[QTot] = Query {V, QHead[U], Index};
    QHead[U] = QTot++;
    querys[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(Ancestor, 0, sizeof(Ancestor));
}

// LCA 离线 Tarjan 算法
void Tarjan(int Node) {
    Ancestor[Node] = Node;
    Vis[Node] = true;
    for (int i = Head[Node]; i != -1; i = edges[i].Next) {
        if (Vis[edges[i].V]) {
            continue;
        }
        Tarjan(edges[i].V);
        Join(Node, edges[i].V);
        Ancestor[Find(Node)] = Node;
    }
    for (int i = QHead[Node]; i != -1; i = querys[i].Next) {
        if (Vis[querys[i].Q]) {
            Answer[querys[i].Index] = Ancestor[Find(querys[i].Q)];
        }
    }
```

```
}
```

4.2 MinimumSpanningTree

4.2.1 Kruskal

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

const int maxn = "Edit";

struct Edge {
    int U, V, Dis;

    bool operator < (const Edge &B) const {
        return Dis < B.Dis;
    }
};

// N: 顶点数, E: 边数, Pre 并查集
int N, E, Pre[maxn];
// edges: 边
Edge edges[maxn];

void Init() {
    // 并查集初始化
    for (int i = 0; i <= N; ++i) {
        Pre[i] = i;
    }
}

// 并查集查询
int Find(int X) {
    return Pre[X] == X ? X : Pre[X] = Find(Pre[X]);
}

// 并查集合并
void Join(int X, int Y) {
    int XX = Find(X);
    int YY = Find(Y);
    if (XX != YY) {
        Pre[XX] = YY;
    }
}

// Kruskal 算法
int Kruskal() {
    // 贪心排序
    std::sort(edges + 1, edges + E + 1);
    Init();
    int Res = 0;
    // 选边计算
    for (int i = 1; i <= E; ++i) {
        Edge Temp = edges[i];
        if (Find(Temp.U) != Find(Temp.V)) {
            Join(Temp.U, Temp.V);
            Res += Temp.Dis;
        }
    }
    return Res;
}
```

4.2.2 Prim

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

const int INF = "Edit";
const int maxn = "Edit";

struct Edge {
    // V: 连接点, Dis: 边权
    int V, Dis;
};

// N: 顶点数, E: 边数
int N, E;
// 松弛更新权值数组
int Dis[maxn];
// 访问标记数组
int Vis[maxn];
// 邻接表
std::vector<Edge> Adj[maxn];

// 建图加边, U、V: 顶点,Weight: 权值
void AddEdge(int U, int V, int Weight) {
    Adj[U].push_back(Edge (V, Weight));
    // 无向图反向建边
    Adj[V].push_back(Edge (U, Weight));
}

// Prim 算法
int Prim(int Start) {
    memset(Dis, INF, sizeof(Dis));
    memset(Vis, 0, sizeof(Vis));
    Dis[Start] = 0;
    int Res = 0;
    for (int i = 1; i <= N; ++i) {
        // 选择距已生成树权值最小的顶点
        int U = -1, Min = INF;
        for (int j = 1; j <= N; ++j) {
            if (!Vis[j] && Dis[j] < Min) {
                U = j;
                Min = Dis[j];
            }
        }
        // 更新、标记
        Vis[U] = 1;
        Res += Min;
        // 松弛
        for (int j = 0; j < int(Adj[U].size()); ++j) {
            int V = Adj[U][j].V;
            if (!Vis[V] && Adj[U][j].Dis < Dis[V]) {
                Dis[V] = Adj[U][j].Dis;
            }
        }
    }
    // 返回结果
    return Res;
}
```

4.3 NetFlow

4.3.1 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;
  }
};
```

4.3.2 FordFulkerson

```
#include <bits/stdc++.h>
// 正无穷
const int INF = "Edit";
const int maxn = "Edit";

// N: 顶点数, E: 边数
int N, E;
// 访问标记数组
bool Vis[maxn];
// 邻接矩阵
int Adj[maxn][maxn];

// Dfs 搜索增广路经, Vertex: 当前搜索顶点, End: 搜索终点, NowFlow: 当前最大流量
```

```
int Dfs(int Vertex, int End, int NowFlow) {
    // 搜索到终点结束
    if (Vertex == End) {
        return NowFlow;
    }
    // 标记访问过的顶点
    Vis[Vertex] = true;
    // 枚举寻找顶点
    for (int i = 1; i <= N; ++i) {
        if (!Vis[i] && Adj[Vertex][i]) {
            int FindFlow = Dfs(i, End, NowFlow < Adj[Vertex][i] ? NowFlow :
            ↪  Adj[Vertex][i]);
            if (!FindFlow) {
                continue;
            }
            // 找到增广路径后更新邻接矩阵残留网
            Adj[Vertex][i] -= FindFlow;
            Adj[i][Vertex] += FindFlow;
            // 返回搜索结果
            return FindFlow;
        }
    }
    // 未找到增广路径，搜索失败
    return false;
}

// Ford-Fulkersone 算法，Start: 起点，End: 终点
int FordFulkerson(int Start, int End) {
    // MaxFlow: 最大流，Flow: 搜索到的增广路径最大流
    int MaxFlow = 0, Flow = 0;
    memset(Vis, false, sizeof(Vis));
    // 搜索增广路径
    while (Flow = Dfs(Start, End, INF)) {
        MaxFlow += Flow;
        memset(Vis, false, sizeof(Vis));
    }
    // 返回结果
    return MaxFlow;
}
```

4.3.3 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;
    }
  }
};
```

4.3.4 MinCostMaxFlow

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

const int INF = "Edit";
const int maxn = "Edit";

// 边
struct Edge {
    // V: 连接点, Flow: 流量, Cost: 费用
    int V, Cap, Cost, Flow, Next;
};

// N: 顶点数, E: 边数
```

```
int N, E;
int Head[maxn];
// 前驱记录数组
int Path[maxn];
int Dis[maxn];
// 访问标记数组
bool Vis[maxn];
int Tot;
// 链式前向星
Edge edges[maxn];

// 链式前向星初始化
void Init() {
    Tot = 0;
    memset(Head, -1, sizeof(Head));
}

// 建图加边, U、V 之间建立一条费用为 Cost 的边
void AddEdge(int U, int V, int Cap, int Cost) {
    edges[Tot] = Edge {V, Cap, Cost, 0, Head[U]};
    Head[U] = Tot++;
    edges[Tot] = Edge {U, 0, -Cost, 0, Head[V]};
    Head[V] = Tot++;
}

//  SPFA 算法, Start: 起点, End: 终点
bool SPFA(int Start, int End) {
    memset(Dis, INF, sizeof(Dis));
    memset(Vis, false, sizeof(Vis));
    memset(Path, -1, sizeof(Path));
    Dis[Start] = 0;
    Vis[Start] = true;
    std::queue<int> Que;
    while (!Que.empty()) {
        Que.pop();
    }
    Que.push(Start);
    while (!Que.empty()) {
        int U = Que.front();
        Que.pop();
        Vis[U] = false;
        for (int i = Head[U]; ~i; i = edges[i].Next) {
            int V = edges[i].V;
            if (edges[i].Cap > edges[i].Flow && Dis[V] > Dis[U] + edges[i].Cost) {
                Dis[V] = Dis[U] + edges[i].Cost;
                Path[V] = i;
                if (!Vis[V]) {
                    Vis[V] = true;
                    Que.push(V);
                }
            }
        }
    }
    return Path[End] != -1;
}

// 最小费用最大流, Start: 起点, End: 终点, Cost: 最小费用
int MinCostMaxFlow(int Start, int End, int &MinCost) {
    int MaxFlow = 0;
    MinCost = 0;
    while (SPFA(Start, End)) {
        int Min = INF;
        for (int i = Path[End]; ~i; i = Path[edges[i ^ 1].V]) {
            if (edges[i].Cap - edges[i].Flow < Min) {
                Min = edges[i].Cap - edges[i].Flow;
            }
        }
        for (int i = Path[End]; ~i; i = Path[edges[i ^ 1].V]) {
            edges[i].Flow += Min;
            edges[i ^ 1].Flow -= Min;
            MinCost += edges[i].Cost * Min;
        }
        MaxFlow += Min;
    }
    // 返回最大流
    return MaxFlow;
}
```

4.4 ShortestPath

4.4.1 BellmanFord

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

const int INF = "Edit";
const int maxn = "Edit";

struct Edge {
    // U、V: 顶点, Dis: 边权
    int U, V;
    int Dis;
};
// 松弛更新数组
int Dis[maxn];
// 边
std::vector<Edge> edges;

// Bellman_Ford 算法判断是否存在负环回路
bool BellmanFord(int Start, int N) {
    memset(Dis, INF, sizeof(Dis));
    Dis[Start] = 0;
    // 最多做 N-1 次
    for (int i = 1; i < N; ++i) {
        bool flag = false;
        for (int j = 0; j < int(edges.size()); ++j) {
            if (Dis[edges[j].V] > Dis[edges[j].U] + edges[j].Dis) {
                Dis[edges[j].V] = Dis[edges[j].U] + edges[j].Dis;
                flag = true;
            }
        }
        // 没有负环回路
        if (!flag) {
            return true;
        }
    }
    // 有负环回路
    for (int j = 0; j < int(edges.size()); ++j) {
        if (Dis[edges[j].V] > Dis[edges[j].U] + edges[j].Dis) {
            return false;
        }
    }
    // 没有负环回路
    return true;
}
```

4.4.2 Dijkstra

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

const int maxn = "Edit";
const int INF = "Edit";

// 边
struct Edge {
    // V: 连接点, Weight: 权值, Next: 上一条边的编号
    int V, Weight, Next;
};

// 边, 一定要开到足够大
Edge edges[maxn << 1];
// Head[i] 为点 i 上最后一条边的编号
int Head[maxn];
// 增加边时更新编号
int Tot;
// 松弛更新数组, 最短路
int Dis[maxn];

// 链式前向星初始化
void Init() {
    Tot = 0;
    memset(Head, -1, sizeof(Head));
}

// 添加一条 U 至 V 权值为 Weight 的边
void AddEdge(int U, int V, int Weight) {
    edges[Tot] = Edge (V, Weight, Head[U]);
    Head[U] = Tot++;
}

// 最短路优化堆排序规则
struct Cmp {
    bool operator() (const int &A, const int &B) {
        return Dis[A] > Dis[B];
    }
};

// N: 顶点数, E: 边数
int N, E;

// Dijkstra 算法, Start: 起点
void Dijkstra(int Start) {
    std::priority_queue<int, std::vector<int>, Cmp> Que;
    memset(Dis, INF, sizeof(Dis));
    Dis[Start] = 0;
    Que.push(Start);
    while (!Que.empty()) {
        int U = Que.top(); Que.pop();
        for (int i = Head[U]; ~i; i = edges[i].Next) {
            if (Dis[edges[i].V] > Dis[U] + edges[i].Weight) {
                Dis[edges[i].V] = Dis[U] + edges[i].Weight;
                Que.push(edges[i].V);
            }
        }
    }
}
```

4.4.3 Floyd

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

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]);
            }
        }
    }
}
```

4.4.4 SPFA

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

const int INF = "Edit";
const int maxn = "Edit";

// 边
struct Edge {
    // V: 连接点, Dis: 边权
    int V, Dis;
};

// N: 顶点数, E: 边数
int N, E;
// 访问标记数组
bool Vis[maxn];
// 每个点的入队列次数
int Cnt[maxn];
// 最短路数组
int Dis[maxn];
// 邻接表
std::vector<Edge> Adj[maxn];

// 建图加边, U、V 之间权值为 Weight 的边
void AddEdge (int U, int V, int Weight) {
    Adj[U].push_back(Edge (V, Weight));
    // 无向图建立反向边
    Adj[V].push_back(Edge (U, Weight));
}

// SPFA 算法, Start: 起点
bool SPFA(int Start) {
    memset(Vis, false, sizeof(Vis));
    memset(Dis, INF, sizeof(Dis));
    memset(Cnt, 0, sizeof(Cnt));
    Vis[Start] = true;
    Dis[Start] = 0;
    Cnt[Start] = 1;
    std::queue<int> Que;
    while (!Que.empty()) {
        Que.pop();
    }
    Que.push(Start);
    while (!Que.empty()) {
        int U = Que.front();
        Que.pop();
        Vis[U] = false;
        for (int i = 0; i < int(Adj[U].size()); ++i) {
            int V = Adj[U][i].V;
            if (Dis[V] > Dis[U] + Adj[U][i].Dis) {
                Dis[V] = Dis[U] + Adj[U][i].Dis;
                if (!Vis[V]) {
                    Vis[V] = true;
                    Que.push(V);
                    // Cnt[i] 为 i 顶点入队列次数，用来判定是否存在负环回路
                    if (++Cnt[V] > N) {
                        return false;
                    }
                }
            }
        }
    }
    return true;
}
```

5 DynamicProgramming

5.1 Contour

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

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

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

const int maxn = "Edit";

long long Digit[25];
long long Dp[25][maxn];

// Site: 数位,Statu: 状态,Pre: 前导零,Limit: 数位上界
long long Dfs(long long Site, long long Statu, bool Pre, bool Limit) {
    if (Site == 0) {
        return ?;
    }
    if (!Limit && ~Dp[Site][Statu]) {
        return Dp[Site][Statu];
    }
    long long Max = Limit ? Digit[Site] : 9;
    long long Ans = 0;
    for (int i = 0; i <= Max; ++i) {
        long long NowStatu = /* 状态转移 */;
        if (NowStatu?) {
            Ans += Dfs(Site - 1, NowStatu, Pre && i == 0, Limit && i == Max);
        }
    }
    if (!Limit) {
        Dp[Site][Statu] = Ans;
    }
    return Ans;
}

long long Cal(long long X) {
    // 数位分解
    long long Len = 0;
    while (X) {
        Digit[++Len] = X % 10;
        X /= 10;
    }
    return Dfs(Len, 0, true, true);
}
```

5.3 LCS

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

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 LCS(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

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

// 最长不下降子序列 (LIS), Num: 序列
int LIS(std::vector<int> &Num) {
    int Ans = 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(Num.size()) + 1, 0);
    Last[1] = Num[1];
    for (int i = 2; i <= int(Num.size()); ++i) {
        if (Num[i] >= Last[Ans]) {
            Last[++Ans] = Num[i];
        }
        else {
            int Index = std::upper_bound(Last.begin() + 1, Last.end(), Num[i]) -
             ↪  Last.begin();
            Last[Index] = Num[i];
        }
    }
    // 返回结果
    return Ans;
}
```

5.5 Pack

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

const int maxn = "Edit";

int Dp[maxn];
// NValue: 背包容量, NKind: 总物品数
int NValue, NKind;

// 01 背包, 代价为 Cost, 获得的价值为 Weight
void ZeroOnePack(int Cost, int Weight) {
    for (int i = NValue; 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 <= NValue; ++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 >= NValue) {
        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 ComputationalGeometry

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

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

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);}

    /*----------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 DisP2P(point k1, point k2) {return sqrt((k1 - k2) * (k1 - k2));}
    db DisP2P2(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(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));
        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 vector<point> poly;
    void RotateCaliper() {
        ans = -1e20;
        if (ConvexHull.size() == 3) {
            if (Cmp(DisP2P(ConvexHull[0], ConvexHull[1]), ans) > 0) ans =
             ↪ DisP2P(ConvexHull[0], ConvexHull[1]);
            if (Cmp(DisP2P(ConvexHull[0], ConvexHull[2]), ans) > 0) ans =
             ↪ DisP2P(ConvexHull[0], ConvexHull[2]);
            if (Cmp(DisP2P(ConvexHull[1], ConvexHull[2]), ans) > 0) ans =
             ↪ DisP2P(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(DisP2P(ConvexHull[i], ConvexHull[cur]), ans) > 0) ans =
             ↪ DisP2P(ConvexHull[i], ConvexHull[cur]);
        }
    }

    poly Grahamscan(point points[], int N) {
        poly ans;
        if (N < 3) {
            for (int i = 0; i < N; ++i) ans.push_back(points[i]);
            return ans;
        }
        int Basic = 0;
        for (int i = 0; i < N; ++i)
            if (Cmp(points[i].X, points[Basic].X) < 0 || (Cmp(points[i].X,
             ↪  points[Basic].X) == 0 && Cmp(points[i].Y, points[Basic].Y) < 0))
                Basic = i;
        std::swap(points[0], points[Basic]);
        std::sort(points + 1, points + N, [&](point k1, point k2) {
            double temp = (k1 - points[0]) ^ (k2 - points[0]);
            if (Sgn(temp) > 0) return true;
            else if (Sgn(temp) == 0 && Cmp(DisP2P(k2, points[0]), DisP2P(k1,
             ↪  points[0])) > 0) return true;
            return false;
        });
        ans.push_back(points[0]);
        for (int i = 1; i < N; ++i) {
            while ((int)ans.size() >= 2 && Sgn((ans.back() - ans[(ans.size()) - 2])
             ↪  ^ (points[i] - ans[(int)ans.size() - 2])) <= 0) {
                ans.pop_back();
            }
            ans.push_back(points[i]);
        }
        return ans;
    }

    db MinCircleCoverage(vector<point> points) {
        point cur = points[0];
        db Probability = 10000, ans = inf;
        while (Probability > eps) {
            int Book = 0;
            for (int i = 0; i < (int)points.size(); ++i)
                if (DisP2P(cur, points[i]) > DisP2P(cur, points[Book]))
                    Book = i;
            db r = DisP2P(cur, points[Book]);
            if (Cmp(r, ans) < 0) ans = r;
            cur = cur + (points[Book] - cur) / r * Probability;
            Probability *= delta;
        }
        return ans;
    }
```

```
/*----------线 (线段)----------*/
struct line {point s, t;};
typedef line seg;
db GetLen(seg k) {return Disp2p(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(DisP2P(k1, k2.s), DisP2P(k1, k2.t));
    }
    return DisP2P(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;};
};
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 Key) {return fabs(Key) < eps ? 0 : (Key < 0 ? -1 : 1);}
    int Cmp(db Key1, db Key2) {return Sgn(Key1 - Key2);}
```

```
/*----------点 (向量)----------*/
struct Point {db X, Y, Z;};
typedef Point Vector;
bool operator == (Point Key1, Point Key2) {return Sgn(Key1.X - Key2.X) == 0 &&
↪  Sgn(Key1.Y - Key2.Y) == 0 && Sgn(Key1.Z - Key1.Z) == 0;}
Vector operator + (Vector Key1, Vector Key2) {return (Vector){Key1.X + Key2.X,
↪  Key1.Y + Key2.Y, Key1.Z + Key2.Z};}
Vector operator - (Vector Key1, Vector Key2) {return (Vector){Key1.X - Key2.X,
↪  Key1.Y - Key2.Y, Key1.Z - Key2.Z};}
db operator * (Vector Key1, Vector Key2) {return Key1.X * Key2.X + Key1.Y *
↪  Key2.Y + Key1.Z * Key2.Z;}
db GetLen(Vector Key) {return sqrt(Key * Key);}
db GetLen2(Vector Key) {return Key * Key;}
db operator ^ (Vector Key1, Vector Key2) {return GetLen((Vector){Key1.Y *
↪  Key2.Z - Key1.Z * Key2.Y, Key1.Z * Key2.X - Key1.X * Key2.Z, Key1.X *
↪  Key2.Y - Key1.Y * Key2.X});}
Vector operator * (Vector Key1, db Key2) {return (Vector){Key1.X * Key2, Key1.Y
↪  * Key2, Key1.Z * Key2};}
Vector operator / (Vector Key1, db Key2) {return (Vector){Key1.X / Key2, Key1.Y
↪  / Key2, Key1.Z / Key2};}
db DisPointToPoint(Point Key1, Point Key2) {return GetLen(Key2 - Key1);}
db DisPointToPoint2(Point Key1, Point Key2) {return GetLen2(Key2 - Key1);}
db GetAngle(Vector Key1, Vector Key2) {return fabs(atan2(fabs(Key1 ^ Key2),
↪  Key1 * Key2));}

db MinimimSphereCoverage(vector<Point> points, int N) {
    Point Cur = points[0];
    db Probability = 10000, Ans = INF;
    while (Probability > eps) {
        int Book = 0;
        for (int i = 0; i < (int)points.size(); ++i) {
            if (Cmp(Distance(Cur, points[i]), Distance(Cur, points[Book])) > 0)
            ↪  {
                Book = i;
            }
        }
        db Radius = Distance(Cur, points[Book]);
        Ans = min(Ans, Radius);
        Cur = Cur + (points[Book] - Cur) / Radius * Probability;
        Probability *= delta;
    }
    return Ans;
}

/*----------线 (线段)----------*/
struct Line {Point S, T;};
typedef Line Segment;
db Length(Segment Key) {return DisPointToPoint(Key.S, Key.T);}
db DisPointToLine(Point Key1, Line Key2) {return fabs((Key1 - Key2.S) ^ (Key2.T
↪  - Key2.S)) / Length(Key2);}
db DisPointToSeg(Point Key1, Segment Key2) {
    if (Sgn((Key1 - Key2.S) * (Key2.T - Key2.S)) < 0 || Sgn((Key1 - Key2.T) *
    ↪  (Key2.S - Key2.T)) < 0) {
        return min(DisPointToPoint(Key1, Key2.S), DisPointToPoint(Key1,
        ↪  Key2.T));
    }
    return DisPointToLine(Key1, Key2);
}

/*----------球----------*/
struct Sphere {Point Center;db Radius;};
db GetVolume(Sphere Key) {return 4.0 / 3.0 * pi * Key.Radius * Key.Radius *
↪  Key.Radius;}
db SphereIntersectVolume(Sphere Key1, Sphere Key2) {
    db Ans = 0.0;
    db Dis = DisPointToPoint(Key1.Center, Key2.Center);
    if (Sgn(Dis - Key1.Radius - Key2.Radius) >= 0) {
        return Ans;
    }
    if (Sgn(Key2.Radius - (Dis + Key1.Radius)) >= 0) {
        return CalVolume(Key1);
    }
    else if (Sgn(Key1.Radius - (Dis + Key2.Radius)) >= 0) {
        return CalVolume(Key2);
    }
    db Length1 = ((Key1.Radius * Key1.Radius - Key2.Radius * Key2.Radius) / Dis
    ↪  + Dis) / 2;
    db Length2 = Dis - Length1;
    db X1 = Key1.Radius - Length1, X2 = Key2.Radius - Length2;
    db V1 = pi * X1 * X1 * (Key1.Radius - X1 / 3.0);
    db V2 = pi * X2 * X2 * (Key2.Radius - X2 / 3.0);
    return V1 + V2;
}

bool IsRayInterSphere(Ray Key1, Sphere Key2, db &Dis) {
    db A = Key1.Dir * Key1.Dir;
    db B = (Key1.Origin - Key2.Center) * Key1.Dir * 2.0;
    db C = ((Key1.Origin - Key2.Center) * (Key1.Origin - Key2.Center)) -
    ↪  (Key2.Radius * Key2.Radius);
    db Delta = B * B - 4.0 * A * C;
    if (Sgn(Delta) < 0) return false;
    db X1 = (-B - sqrt(Delta)) / (2.0 * A), X2 = (-B + sqrt(Delta)) / (2.0 *
    ↪  A);
    if (Cmp(X1, X2) > 0) swap(X1, X2);
    if (Sgn(X1) <= 0) return false;
    Dis = X1;
    return true;
    }

    void Reflect(Ray &Key1, Sphere Key2, db Dis) {
        Point Pos = Key1.Origin + (Key1.Dir * Dis);
        Vector Temp = Key2.Center + (((Pos - Key2.Center) * ((Pos - Key2.Center) *
        ↪  (Key1.Origin - Key2.Center))) / GetLen2(Pos - Key2.Center));
        Key1.Dir = Temp * 2.0 - Key1.Origin - Pos; Key1.Origin = Pos;
    }
};
using namespace Geometry3D;
```

7 Others

7.1 Factorial

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

void Factorial() {
    int res[10010];
    int Book = 1;
    int BaoFour = 0;
    res[Book] = 1;
    int n;
    scanf("%d", &n);
    // 乘法计算
    for (int i = 1;i <= n;++i) {
        BaoFour = 0;
        for (int j = 1;j <= Book;++j) {
            res[j] = res[j] * i + BaoFour;
            BaoFour = res[j] / 10000;
            res[j] = res[j] % 10000;
        }
        if (BaoFour > 0) {
            res[++Book] += BaoFour;
        }
    }
    printf("%d", res[Book]);
    // 补零输出
    for (int i = Book - 1;i > 0;--i) {
        if (res[i] >= 1000) {
            printf("%d", res[i]);
        }
        else if (res[i] >= 100) {
            printf("0%d",res[i]);
        }
        else if (res[i] >= 10) {
            printf("00%d",res[i]);
        }
        else {
            printf("000%d",res[i]);
        }
    }
    putchar('\n');
}
```

7.2 FastIO

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

// 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,10000000000000
            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

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

inline bool Leep(int Year) {
    return (!(Year % 4) && (Year % 100)) || !(Year % 400);
}
```

7.4 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
```