ACM/ICPC Template Manaual

Mango Gao

March 15, 2016

Contents

1 头文件模板 **4**

2 数学 **5**
- 2.1 素数 . . . 5
  - 2.1.1 埃氏筛 . . . 5
  - 2.1.2 欧拉筛 . . . 5
  - 2.1.3 随机素数判定 . . . 5
  - 2.1.4 分解质因数 . . . 6
- 2.2 欧拉函数 . . . 6
  - 2.2.1 求一个数的欧拉函数 . . . 6
  - 2.2.2 筛法求欧拉函数 . . . 7
- 2.3 扩展欧几里得-乘法逆元 . . . 7
  - 2.3.1 扩展欧几里得 . . . 7
  - 2.3.2 求 ax+by=c 的解 . . . 7
  - 2.3.3 乘法逆元 . . . 8
- 2.4 模线性方程组 . . . 8
  - 2.4.1 中国剩余定理 . . . 8
  - 2.4.2 一般模线性方程组 . . . 8
- 2.5 组合数学 . . . 9
  - 2.5.1 一般组合数 . . . 9
  - 2.5.2 Lucas 定理 . . . 9
  - 2.5.3 大组合数 . . . 10
  - 2.5.4 Polya 定理 . . . 11
- 2.6 快速乘-快速幂 . . . 11
- 2.7 莫比乌斯反演 . . . 11
  - 2.7.1 莫比乌斯 . . . 11
  - 2.7.2 n 个数中互质数对数 . . . 12
  - 2.7.3 VisibleTrees . . . 13
- 2.8 其他 . . . 13
  - 2.8.1 Josephus 问题 . . . 13
  - 2.8.2 数位问题 . . . 14
- 2.9 相关公式 . . . 14

3 字符串 **15**
- 3.1 KMP . . . 15
- 3.2 Manacher 最长回文子串 . . . 15

4 数据结构 **16**
- 4.1 树状数组 . . . 16
- 4.2 线段树 . . . 16
  - 4.2.1 声明 . . . 16
  - 4.2.2 单点更新-区间查询 . . . 16
  - 4.2.3 区间更新-区间查询 . . . 17

5 图论 **19**
5.1 并查集 . . . 19
5.2 最小生成树 . . . 19
5.2.1 Kruskal . . . 19
5.2.2 Prim . . . 20
5.3 最短路 . . . 20
5.3.1 Dijkstra-邻接矩阵 . . . 20
5.3.2 Dijkstra-邻接表数组 . . . 21
5.3.3 Dijkstra-邻接表向量 . . . 22
5.3.4 Dijkstra-优先队列 . . . 23
5.3.5 Bellman-Ford(可判负环) . . . 24
5.3.6 SPFA . . . 25
5.3.7 Floyd 算法 . . . 26
5.4 拓扑排序 . . . 26
5.4.1 邻接矩阵 . . . 26
5.4.2 邻接表 . . . 27
5.5 欧拉回路 . . . 28
5.5.1 判定 . . . 28
5.5.2 求解 . . . 28

6 计算几何 **30**
6.1 基本函数 . . . 30
6.2 位置关系 . . . 31
6.2.1 两点间距离 . . . 31
6.2.2 直线与直线的交点 . . . 31
6.2.3 判断线段与线段相交 . . . 31
6.2.4 判断线段与直线相交 . . . 31
6.2.5 点到直线距离 . . . 32
6.2.6 点到线段距离 . . . 32
6.2.7 点在线段上 . . . 32
6.3 多边形 . . . 32
6.3.1 多边形面积 . . . 32
6.3.2 点在凸多边形内 . . . 33
6.3.3 点在任意多边形内 . . . 33
6.3.4 判断凸多边形 . . . 34
6.3.5 小结 . . . 34
6.4 整数点问题 . . . 37
6.4.1 线段上整点个数 . . . 37
6.4.2 多边形边上整点个数 . . . 37
6.4.3 多边形内整点个数 . . . 37
6.5 圆 . . . 38
6.5.1 过三点求圆心 . . . 38

7 动态规划 **39**
7.1 子序列 . . . 39
7.1.1 最大子序列和 . . . 39
7.1.2 最长上升子序列 LIS . . . 39

7.1.3 最长公共上升子序列 LCIS 40

8 其他 **41**
8.1 矩阵 41
8.2 高精度 42

1 头文件模板

```
#include <bits/stdc++.h> // c++0x only
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <string>
#include <vector>
#include <queue>
#include <stack>
#include <set>
#include <map>
#include <cmath>
#include <iomanip>
#include <functional>
#include <cstdlib>
#include <climits>
#include <cctype>
using namespace std;
#define REP(i,x) for(int i = 0; i < (x); i++)
#define DEP(i,x) for(int i = (x) - 1; i >= 0; i--)
#define FOR(i,x) for(__typeof(x.begin())i=x.begin(); i!=x.end();
   i++)
#define CLR(a,x) memset(a, x, sizeof(a))
#define MO(a,b) (((a)%(b)+(b))%(b))
#define ALL(x) (x).begin(), (x).end()
#define SZ(v) ((int)v.size())
#define UNIQUE(v) sort(ALL(v)); v.erase(unique(ALL(v)), v.end())
#define out(x) cout << #x << ": " << x << endl;
#define fastcin ios_base::sync_with_stdio(0);cin.tie(0);
typedef long long ll;
typedef unsigned long long ull;
typedef pair<int, int> PII;
typedef vector<int> VI;
#define INF 0x3f3f3f3f
#define MOD 1000000007
#define EPS 1e-8
#define MP(x,y) make_pair(x,y)
#define MT(x,y...) make_tuple(x,y) // c++0x only
#define PB(x) push_back(x)
#define IT iterator
#define X first
#define Y second
```

2 数学

2.1 素数

2.1.1 埃氏筛

$O(n\log\log n)$ 筛出 MAXN 内所有素数 $notprime[i]=0/1$ 0 为素数 1 为非素数

```
const int MAXN = 1000100;
bool notprime[MAXN] = {1, 1}; // 0 && 1 为非素数
void GetPrime() {
   for (int i = 2; i < MAXN; i++)
      if (!notprime[i] && i <= MAXN / i) // 筛到√n为止
         for (int j = i * i; j < MAXN; j += i)
            notprime[j] = 1;
}
```

2.1.2 欧拉筛

$O(n)$ 得到欧拉函数 $phi[]$、素数表 $prime[]$、素数个数 tot 传入的 n 为函数定义域上界

```
const int MAXN = 100010;
bool vis[MAXN];
int tot, phi[MAXN], prime[MAXN];
void CalPhi(int n) {
   set(vis, 0); phi[1] = 1; tot = 0;
   for (int i = 2; i < n; i++) {
      if (!vis[i]) {
         prime[tot++] = i;
         phi[i] = i - 1;
      }
      for (int j = 0; j < tot; j++) {
         if (i * prime[j] > n) break;
         vis[i * prime[j]] = 1;
         if (i % prime[j] == 0) {
            phi[i * prime[j]] = phi[i] * prime[j];
            break;
         }
         else phi[i * prime[j]] = phi[i] * (prime[j] - 1);
      }
   }
}
```

2.1.3 随机素数判定

$O(s\log n)$ 内判定 2^{63} 内的数是不是素数，s 为测定次数

```
bool Miller_Rabin(ll n, int s) {
   if (n == 2) return 1;
```

```
    if (n < 2 || !(n & 1)) return 0;
    int t = 0; ll x, y, u = n - 1;
    while ((u & 1) == 0) t++, u >>= 1;
    for (int i = 0; i < s; i++) {
        ll a = rand() % (n - 1) + 1;
        ll x = Pow(a, u, n);
        for (int j = 0; j < t; j++) {
            ll y = Mul(x, x, n);
            if (y == 1 && x != 1 && x != n - 1) return 0;
            x = y;
        }
        if (x != 1) return 0;
    }
    return 1;
}
```

2.1.4 分解质因数

函数返回素因数个数数组以 $fact[i][0]^{fact[i][1]}$ 的形式保存第 i 个素因数

```
ll fact[100][2];
int getFactors(ll x) {
    int cnt = 0;
    for (int i = 0; prime[i] <= x / prime[i]; i++) {
        fact[cnt][1] = 0;
        if (x % prime[i] == 0 ) {
            fact[cnt][0] = prime[i];
            while (x % prime[i] == 0) {
                fact[cnt][1]++;
                x /= prime[i];
            }
            cnt++;
        }
    }
    if (x != 1) {
        fact[cnt][0] = x;
        fact[cnt++][1] = 1;
    }
    return cnt;
}
```

2.2 欧拉函数

2.2.1 求一个数的欧拉函数

```
long long Euler(long long n) {
    long long rt = n;
```

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

2.2.2 筛法求欧拉函数

```
const int MAXN = 10001;
int phi[MAXN] = {0, 1};
void CalEuler() {
    for (int i = 2; i < MAXN; i++)
        if (!phi[i]) for (int j = i; j < MAXN; j += i) {
                if (!phi[j]) phi[j] = j;
                phi[j] = phi[j] / i * (i - 1);
            }
}
```

2.3 扩展欧几里得-乘法逆元

2.3.1 扩展欧几里得

```
void exgcd(ll a, ll b, ll &d, ll &x, ll &y) {
    if (!b) {d = a; x = 1; y = 0;}
    else {exgcd(b, a % b, d, y, x); y -= x * (a / b);}
}
```

2.3.2 求 ax+by=c 的解

```
// 引用返回通解: X = x + k * dx, Y = y - k * dy
// 引用返回的x是最小非负整数解，方程无解函数返回0
#define Mod(a,b) (((a)%(b)+(b))%(b))
bool solve(ll a, ll b, ll c, ll &x, ll &y, ll &dx, ll &dy) {
    if (a == 0 && b == 0) return 0;
    ll d, x0, y0; exgcd(a, b, d, x0, y0);
    if (c % d != 0) return 0;
    dx = b / d; dy = a / d;
    x = Mod(x0 * c / d, dx); y = (c - a * x) / b;
// y = Mod(y0 * c / d, dy); x = (c - b * y) / a;
    return 1;
}
```

2.3.3 乘法逆元

```
// 利用exgcd求a在模m下的逆元，需要保证gcd(a, m) == 1.
ll inv(ll a, ll m) {
   ll x, y, d; exgcd(a, m, d, x, y);
   return d == 1 ? (x + m) % m : -1;
}
// a < m 且 m为素数时，有以下两种求法
ll inv(ll a, ll m) {
   return a == 1 ? 1 : inv(m % a, m) * (m - m / a) % m;
}
ll inv(ll a, ll m) {
   return Pow(a, m - 2, m);
}
```

2.4 模线性方程组

2.4.1 中国剩余定理

```
// X = r[i] (mod m[i]); 要求m[i]两两互质
// 引用返回通解X = re + k * mo;
void crt(ll r[], ll m[], ll n, ll &re, ll &mo) {
   mo = 1, re = 0;
   for (int i = 0; i < n; i++) mo *= m[i];
   for (int i = 0; i < n; i++) {
      ll x, y, d, tm = mo / m[i];
      exgcd(tm, m[i], d, x, y);
      re = (re + tm * x * r[i]) % mo;
   } re = (re + mo) % mo;
}
```

2.4.2 一般模线性方程组

```
// X = r[i] (mod m[i]); m[i]可以不两两互质
// 引用返回通解X = re + k * mo; 函数返回是否有解
bool excrt(ll r[], ll m[], ll n, ll &re, ll &mo) {
   ll x, y, d; mo = m[0], re = r[0];
   for (int i = 1; i < n; i++) {
      exgcd(mo, m[i], d, x, y);
      if ((r[i] - re) % d != 0) return 0;
      x = (r[i] - re) / d * x % (m[i] / d);
      re += x * mo;
      mo = mo / d * m[i];
      re %= mo;
   } re = (re + mo) % mo;
   return 1;
}
```

2.5 组合数学

2.5.1 一般组合数

```
// 0 <= m <= n <= 1000
const int maxn = 1010;
ll C[maxn][maxn];
void CalComb() {
   C[0][0] = 1;
   for (int i = 1; i < maxn; i++) {
      C[i][0] = 1;
      for (int j = 1; j <= i; j++)
         C[i][j] = (C[i - 1][j - 1] + C[i - 1][j]) % mod;
   }
}

// 0 <= m <= n <= 105, 模p为素数
const int maxn = 100010;
ll f[maxn];
void CalFact() {
   f[0] = 1;
   for (int i = 1; i < maxn; i++)
      f[i] = (f[i - 1] * i) % mod;
}
ll C(int n, int m) {
   return f[n] * inv(f[m] * f[n - m] % mod, mod) % mod;
}
```

2.5.2 Lucas 定理

```
// 1 <= n, m <= 1000000000, 1 < p < 100000, p是素数
const int maxp = 100010;
ll f[maxp];
void CalFact(ll p) {
   f[0] = 1;
   for (int i = 1; i <= p; i++)
      f[i] = (f[i - 1] * i) % p;
}
ll Lucas(ll n, ll m, ll p) {
   ll ret = 1;
   while (n && m) {
      ll a = n % p, b = m % p;
      if (a < b) return 0;
      ret = (ret * f[a] * Pow(f[b] * f[a - b] % p, p - 2, p)) % p
         ;
      n /= p; m /= p;
   }
```

```
    return ret;
}
```

2.5.3 大组合数

```
// 0 <= n <= 109, 0 <= m <= 104, 1 <= k <= 109+7
vector<int> v;
int dp[110];
ll Cal(int l, int r, int k, int dis) {
    ll res = 1;
    for (int i = l; i <= r; i++) {
        int t = i;
        for (int j = 0; j < v.size(); j++) {
            int y = v[j];
            while (t % y == 0) {
                dp[j] += dis;
                t /= y;
            }
        }
        res = res * (ll)t % k;
    }
    return res;
}
ll Comb(int n, int m, int k) {
    set(dp, 0); v.clear(); int tmp = k;
    for (int i = 2; i * i <= tmp; i++) {
        if (tmp % i == 0) {
            int num = 0;
            while (tmp % i == 0) {
                tmp /= i;
                num++;
            }
            v.pb(i);
        }
    } if (tmp != 1) v.pb(tmp);
    ll ans = Cal(n - m + 1, n, k, 1);
    for (int j = 0; j < v.size(); j++) {
        ans = ans * Pow(v[j], dp[j], k) % k;
    }
    ans = ans * inv(Cal(2, m, k, -1), k) % k;
    return ans;
}
```

2.5.4 Polya 定理

推论：一共 n 个置换，第 i 个置换的循环节个数为 $gcd(i,n)$ $N*N$ 的正方形格子，$c^{n^2}+2c^{\frac{n^2+3}{4}}+c^{\frac{n^2+1}{2}}+2c^{n\frac{n+1}{2}}+2c^{\frac{n(n+1)}{2}}$ 正六面体，$\frac{m^8+17m^4+6m^2}{24}$ 正四面体，$\frac{m^4+11m^2}{12}$

```
// 长度为n的项链串用c种颜色染
ll solve(int c, int n) {
   if (n == 0) return 0;
   ll ans = 0;
   for (int i = 1; i <= n; i++)
      ans += Pow(c, __gcd(i, n));
   if (n & 1)
      ans += n * Pow(c, n + 1 >> 1);
   else
      ans += n / 2 * (1 + c) * Pow(c, n >> 1);
   return ans / n / 2;
}
```

2.6 快速乘-快速幂

```
ll Mul(ll a, ll b, ll mod) {
   ll t = 0;
   for (; b; b >>= 1, a = (a << 1) % mod)
      if (b & 1) t = (t + a) % mod;
   return t;
}
ll Pow(ll a, ll n, ll mod) {
   ll t = 1;
   for (; n; n >>= 1, a = (a * a % mod))
      if (n & 1) t = (t * a % mod);
   return t;
}
```

2.7 莫比乌斯反演

2.7.1 莫比乌斯

$F(n)=\sum_{d|n}f(d)\Rightarrow f(n)=\sum_{d|n}\mu(d)F(\frac{n}{d})$ $F(n)=\sum_{n|d}f(d)\Rightarrow f(n)=\sum_{n|d}\mu(\frac{d}{n})F(d)$

```
long long ans;
const int MAXN = 1e5 + 1;
int n, x, prime[MAXN], tot, mu[MAXN];
bool check[MAXN];
void calmu() {
   mu[1] = 1;
   for (int i = 2; i < MAXN; i++) {
      if (!check[i]) {
         prime[tot++] = i;
```

```
            mu[i] = -1;
        }
        for (int j = 0; j < tot; j++) {
            if (i * prime[j] >= MAXN) break;
            check[i * prime[j]] = true;
            if (i % prime[j] == 0) {
                mu[i * prime[j]] = 0;
                break;
            } else {
                mu[i * prime[j]] = -mu[i];
            }
        }
    }
}
```

2.7.2 n 个数中互质数对数

```
// 有n个数（n<=100000），问这n个数中互质的数的对数
#include <cstdio>
#include <cstring>
#include <cstdlib>
using namespace std;
long long ans;
const int MAXN = 1e5 + 1;
int n, x, prime[MAXN], _max, b[MAXN], tot, mu[MAXN];
bool check[MAXN];
void calmu() {
    mu[1] = 1;
    for (int i = 2; i < MAXN; i++) {
        if (!check[i]) {
            prime[tot++] = i;
            mu[i] = -1;
        }
        for (int j = 0; j < tot; j++) {
            if (i * prime[j] >= MAXN) break;
            check[i * prime[j]] = true;
            if (i % prime[j] == 0) {
                mu[i * prime[j]] = 0;
                break;
            } else {
                mu[i * prime[j]] = -mu[i];
            }
        }
    }
}
int main() {
```

```
    calmu();
    while (scanf("%d", &n) == 1) {
        memset(b, 0, sizeof(b));
        _max = 0; ans = 0;
        for (int i = 0; i < n; i++) {
            scanf("%d", &x);
            if (x > _max) _max = x;
            b[x]++;
        }
        int cnt;
        for (int i = 1; i <= _max; i++) {
            cnt = 0;
            for (long long j = i; j <= _max; j += i)
                cnt += b[j];
            ans += 1LL * mu[i] * cnt * cnt;
        }
        printf("%lld\n", (ans - b[1]) / 2);
    }
    return 0;
}
```

2.7.3 VisibleTrees

```
// gcd(x,y)==1的对数 x<=n, y<=m
int main() {
    calmu();
    int n, m;
    scanf("%d %d", &n, &m);
    if (n < m) swap(n, m);
    ll ans = 0;
    for (int i = 1; i <= m; ++i) {
        ans += (ll)mu[i] * (n / i) * (m / i);
    }
    printf("%lld\n", ans);
    return 0;
}
```

2.8 其他

2.8.1 Josephus 问题

```
#include <iostream>
using namespace std;
int main() {
    int num, m, r
    while (cin >> num >> m) {
```

```
        r = 0;
        for (int k = 1; k <= num; ++k)
            r = (r + m) % k;
        cout << r + 1 << endl;
    }
    return 0;
}
```

2.8.2 数位问题

```
// n^n最左边一位数
int leftmost(int n) {
    double m = n * log10((double)n);
    double g = m - (long long)m;
    g = pow(10.0, g);
    return (int)g;
}

// n!位数
int count(ll n) {
    return n == 1 ? 1 : (int)ceil(0.5 * log10(2 * M_PI * n) + n *
        log10(n) - n * log10(M_E));
}
```

2.9 相关公式

约数定理：若 $n = \prod_{i=1}^{k} p_i^{a_i}$，则 1. 约数个数 $f(n) = \prod_{i=1}^{k}(a_i + 1)$ 2. 约数和 $g(n) = \prod_{i=1}^{k}(\sum_{j=0}^{a_i} p_i^j)$ 错排公式：$D(n) = (n-1)(D(n-2)+D(n-1)) = \sum_{i=2}^{n} \frac{(-1)^k n!}{k!} = [\frac{n!}{e}+0.5]$ 威尔逊定理：$p\ is\ prime\ \Rightarrow (p-1)! \equiv -1(mod\ p)$ 欧拉定理：$gcd(a,n) = 1 \Rightarrow a^{\varphi(n)} \equiv 1(mod\ n)$ 欧拉定理推广：$gcd(n,p) = 1 \Rightarrow a^n \equiv a^{n\%\varphi(p)}(mod\ p)$ 素数定理：对于不大于 n 的素数个数 $\pi(n)$，$\lim_{n\to\infty} \pi(n) = \frac{n}{\ln n}$ 位数公式：正整数 x 的位数 $N = log10(n) + 1$ 斯特灵公式 $n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$ 设 $a > 1, m, n > 0$, 则 $gcd(a^m-1, a^n-1) = a^{gcd(m,n)}-1$ 设 $a > b, gcd(a,b) = 1$, 则 $gcd(a^m - b^m, a^n - b^n) = a^{gcd(m,n)} - b^{gcd(m,n)}$

$$G = gcd(C_n^1, C_n^2, ..., C_n^{n-1}) = \begin{cases} n, & n \text{ is prime} \\ 1, & n \text{ has multy prime factors} \\ p, & n \text{ has single prime factor } p \end{cases}$$

$gcd(Fib(m), Fib(n)) = Fib(gcd(m,n))$ 若 $gcd(m,n) = 1$, 则: 1. 最大不能组合的数为 $m*n-m-n$ 2. 不能组合数个数 $N = \frac{(m-1)(n-1)}{2}$ $(n+1)lcm(C_n^0, C_n^1, ..., C_n^{n-1}, C_n^n) = lcm(1, 2, ..., n+1)$ 若 p 为素数,则 $(x+y+...+w)^p \equiv x^p+y^p+...+w^p(mod\ p)$ 卡特兰数：1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012 $h(0) = h(1) = 1, h(n) = \frac{(4n-2)h(n-1)}{n+1} = \frac{C_{2n}^n}{n+1} = C_{2n}^n - C_{2n}^{n-1}$

3 字符串

3.1 KMP

```
// 返回y中x的个数
int next[10010];
void kmp_pre(char x[], int m, int next[]) {
   int i, j; j = next[0] = -1; i = 0;
   while (i < m) {
      while (j != -1 && x[i] != x[j])
         j = next[j];
      next[++i] = ++j;
   }
}
int KMP (char x[], int m, char y[], int n) {
   int i, j, ans; i = j = ans = 0;
   kmp_pre (x, m, next);
   while (i < n) {
      while (j != -1 && y[i] != x[j]) j = next[j];
      i++; j++;
      if (j >= m) {
         ans++; j = next[j];
      }
   }
   return ans;
}
```

3.2 Manacher 最长回文子串

```
const int MAXN = 110010;
char Ma[MAXN * 2];
int Mp[MAXN * 2];
void Manacher(char s[], int len) {
   int l = 0; Ma[l++] = '$'; Ma[l++] = '#';
   for (int i = 0; i < len; i++) {
      Ma[l++] = s[i]; Ma[l++] = '#';
   }
   Ma[l] = 0; int mx = 0, id = 0;
   for (int i = 0; i < l; i++) {
      Mp[i] = mx > i ? min(Mp[2 * i d - i], mx - i) : 1;
      while (Ma[i + Mp[i]] == Ma[i - Mp[i]]) Mp[i]++;
      if (i + Mp[i] > mx) {
         mx = i + Mp[i]; id = i;
      }
   }
}
```

4 数据结构

4.1 树状数组

$O(\log n)$ 查询和修改数组的前缀和

```
// 注意下标应从1开始
#define lowbit(i) (i&(-i))
int bit[maxn], n;
int query(int i){
   int s = 0;
   while(i){
      s += bit[i];
      i -= lowbit(i);
   }
   return s;
}
void add(int i, int x){
   while(i<=n){
      bit[i] += x;
      i += lowbit(i);
   }
}
```

4.2 线段树

4.2.1 声明

```
#define lson rt<<1   // 左儿子
#define rson rt<<1|1   // 右儿子
#define Lson l,m,lson  // 左子树
#define Rson m+1,r,rson  // 右子树
void PushUp(int rt);   // 用lson和rson更新rt
void PushDown(int rt[, int m]); // rt的标记下移，m为区间长度（若与标记
   有关）
void build(int l, int r, int rt); // 以rt为根节点，对区间[l, r]建立线
   段树
void update([...,] int l, int r, int rt) // rt[l, r]内寻找目标并更新
int query(int L, int R, int l, int r, int rt) // rt-[l, r]内查询[L
   , R]
```

4.2.2 单点更新-区间查询

```
const int maxn = 50010;
int sum[maxn << 2];
void PushUp(int rt) {
   sum[rt] = sum[lson] + sum[rson];
```

```
}
void build(int l, int r, int rt) {
   if (l == r) {scanf("%d", &sum[rt]); return;} // 建立的时候直接输
      入叶节点
   int m = (l + r) >> 1;
   build(Lson); build(Rson);
   PushUp(rt);
}
void update(int p, int add, int l, int r, int rt) {
   if (l == r) {sum[rt] += add; return;}
   int m = (l + r) >> 1;
   if (p <= m) update(p, add, Lson);
   else update(p, add, Rson);
   PushUp(rt);
}
int query(int L, int R, int l, int r, int rt) {
   if (L <= l && r <= R) {return sum[rt];}
   int m = (l + r) >> 1, s = 0;
   if (L <= m) s += query(L, R, Lson);
   if (m < R) s += query(L, R, Rson);
   return s;
}
```

4.2.3 区间更新-区间查询

```
// seg[rt]用于存放懒惰标记，注意PushDown时标记的传递
const int maxn = 101010;
int seg[maxn << 2], sum[maxn << 2];
void PushUp(int rt) {
   sum[rt] = sum[lson] + sum[rson];
}
void PushDown(int rt, int m) {
   if (seg[rt] == 0) return;
   seg[lson] += seg[rt];
   seg[rson] += seg[rt];
   sum[lson] += seg[rt] * (m - (m >> 1));
   sum[rson] += seg[rt] * (m >> 1);
   seg[rt] = 0;
}
void build(int l, int r, int rt) {
   seg[rt] = 0;
   if (l == r) {scanf("%lld", &sum[rt]); return;}
   int m = (l + r) >> 1;
   build(Lson); build(Rson);
   PushUp(rt);
}
```

```cpp
void update(int L, int R, int add, int l, int r, int rt) {
   if (L <= l && r <= R) {
      seg[rt] += add;
      sum[rt] += add * (r - l + 1);
      return;
   }
   PushDown(rt, r - l + 1);
   int m = (l + r) >> 1;
   if (L <= m) update(L, R, add, Lson);
   if (m < R) update(L, R, add, Rson);
   PushUp(rt);
}
int query(int L, int R, int l, int r, int rt) {
   if (L <= l && r <= R) return sum[rt];
   PushDown(rt, r - l + 1);
   int m = (l + r) >> 1, ret = 0;
   if (L <= m) ret += query(L, R, Lson);
   if (m < R) ret += query(L, R, Rson);
   return ret;
}
```

5 图论

5.1 并查集

```
const int MAXN = 128;
int fa[MAXN], ra[MAXN];
void init(int n) {
   for (int i = 0; i <= n; i++) {
      fa[i] = i; ra[i] = 0;
   }
}
int find(int x) {
   if (fa[x] != x) fa[x] = find(fa[x]);
   return fa[x];
}
void unite(int x, int y) {
   x = find(x); y = find(y); if (x == y) return;
   if (ra[x] < ra[y]) fa[x] = y;
   else {
      fa[y] = x; if (ra[x] == ra[y]) ra[x]++;
   }
}
bool same(int x, int y) {
   return find(x) == find(y);
}
```

5.2 最小生成树

5.2.1 Kruskal

```
vector<pair<int, PII> > G;
void add_edge(int u, int v, int d) {
   G.pb(mp(d, mp(u, v)));
}
int Kruskal(int n) {
   init(n);
   sort(G.begin(), G.end());
   int m = G.size();
   int num = 0, ret = 0;
   for (int i = 0; i < m; i++) {
      pair<int, PII> p = G[i];
      int x = p.Y.X;
      int y = p.Y.Y;
      int d = p.X;
      if (!same(x, y)) {
         unite(x, y);
         num++;
```

```
            ret += d;
        }
        if (num == n - 1) break;
    }
    return ret;
}
```

5.2.2 Prim

```
// 耗费矩阵cost[][],标号从0开始,0~n-1
// 返回最小生成树的权值,返回-1表示原图不连通
const int INF = 0x3f3f3f3f;
const int MAXN = 110;
bool vis[MAXN];
int lowc[MAXN];
int Prim(int cost[][MAXN], int n) {
    int ans = 0;
    set(vis, 0);
    vis[0] = 1;
    for (int i = 1; i < n; i++)
        lowc[i] = cost[0][i];
    for (int i = 1; i < n; i++) {
        int minc = INF;
        int p = -1;
        for (int j = 0; j < n; j++)
            if (!vis[j] && minc > lowc[j]) {
                minc = lowc[j];
                p = j;
            }
        if (minc == INF) return -1;
        vis[p] = 1;
        for (int j = 0; j < n; j++)
            if (!vis[j] && lowc[j] > cost[p][j]) lowc[j] = cost[p][j
                ];
    }
    return ans;
}
```

5.3 最短路

5.3.1 Dijkstra-邻接矩阵

```
// MAXN为点数最大值 求s到所有点的最短路
// 要求边权值为非负数 模板为有向边
// dis[x]为起点到点x的最短路 inf表示无法走到
// 记得初始化
```

```
const int MAXN = 100; // 点数最大值
const int INF = 0x3f3f3f3f;
int G[MAXN][MAXN], dis[MAXN];
bool vis[MAXN];
void init(int n) {
   set(G, 0x3f);
}
void add_edge(int u, int v, int w) {
   G[u][v] = min(G[u][v], w);
}
void Dijkstra(int s, int n) {
   set(vis, 0);
   set(dis, 0x3f);
   dis[s] = 0;
   for (int i = 0; i < n; i++) {
      int x, min_dis = INF;
      for (int j = 0; j < n; j++) {
         if (!vis[j] && dis[j] <= min_dis) {
            x = j;
            min_dis = dis[j];
         }
      }
      vis[x] = 1;
      for (int j = 0; j < n; j++)
         dis[j] = min(dis[j], dis[x] + G[x][j]);
   }
}
```

5.3.2 Dijkstra-邻接表数组

```
// 点最大值: MAX_N 边最大值: MAX_E
// 求起点s到每个点x的最短路dis[x]
const int MAX_N = "Edit"; // 点数最大值
const int MAX_E = "Edit";
const int INF = 0x3F3F3F3F;
int tot;
int Head[MAX_N], vis[MAX_N], dis[MAX_N];
int Next[MAX_E], To[MAX_E], W[MAX_E];
void init() {
   tot = 0;
   memset(Head, -1, sizeof(Head));
}
void add_edge(int u, int v, int d) {
   W[tot] = d;
   To[tot] = v;
   Next[tot] = Head[u];
```

```
    Head[u] = tot++;
}
void Dijkstra(int s, int n) {
    memset(vis, 0, sizeof(vis));
    memset(dis, 0x3F, sizeof(dis));
    dis[s] = 0;
    for (int i = 0; i < n; i++) {
        int x, min_dis = INF;
        for (int j = 0; j < n; j++) {
            if (!vis[j] && dis[j] <= min_dis) {
                x = j;
                min_dis = dis[j];
            }
        }
        vis[x] = 1;
        for (int j = Head[x]; j != -1; j = Next[j]) {
            int y = To[j];
            dis[y] = min(dis[y], dis[x] + W[j]);
        }
    }
}
```

5.3.3 Dijkstra-邻接表向量

```
// MAXN: 点数最大值
// 求起点s到所有点x的最短路dis[x]
// 记得初始化
const int MAXN = "Edit";
const int INF = 0x3F3F3F3F;
vector<int> G[MAXN];
vector<int> GW[MAXN];
bool vis[MAXN];
int dis[MAXN];
void init(int n) {
    for (int i = 0; i < n; i++) {
        G[i].clear();
        GW[i].clear();
    }
}
void add_edge(int u, int v, int w) {
    G[u].push_back(v);
    GW[u].push_back(w);
}
void Dijkstra(int s, int n) {
    memset(vis, false, sizeof(vis));
    memset(dis, 0x3F, sizeof(dis));
```

```
    dis[s] = 0;
    for (int i = 0; i < n; i++) {
        int x;
        int min_dis = INF;
        for (int j = 0; j < n; j++) {
            if (!vis[j] && dis[j] <= min_dis) {
                x = j;
                min_dis = dis[j];
            }
        }
        vis[x] = true;
        for (int j = 0; j < (int)G[x].size(); j++) {
            int y = G[x][j];
            int w = GW[x][j];
            dis[y] = min(dis[y], dis[x] + w);
        }
    }
}
```

5.3.4 Dijkstra-优先队列

```
// pair<权值, 点>
// 记得初始化
const int MAXN = "Edit";
const int INF = 0x3F3F3F3F;
typedef pair<int, int> PII;
typedef vector<PII> VII;
VII G[MAXN];
int vis[MAXN], dis[MAXN];
void init(int n) {
    for (int i = 0; i < n; i++)
        G[i].clear();
}
void add_edge(int u, int v, int w) {
    G[u].push_back(make_pair(w, v));
}
void Dijkstra(int s, int n) {
    memset(vis, 0, sizeof(vis));
    memset(dis, 0x3F, sizeof(dis));
    dis[s] = 0;
    priority_queue<PII, VII, greater<PII> > PQ;
    PQ.push(make_pair(dis[s], s));
    while (!PQ.empty()) {
        PII t = PQ.top();
        int x = t.second;
        PQ.pop();
```

```
        if (vis[x]) continue;
        vis[x] = 1;
        for (int i = 0; i < (int)G[x].size(); i++) {
            int y = G[x][i].second;
            int w = G[x][i].first;
            if (!vis[y] && dis[y] > dis[x] + w) {
                dis[y] = dis[x] + w;
                PQ.push(make_pair(dis[y], y));
            }
        }
    }
}
```

5.3.5 Bellman-Ford(可判负环)

```
// 求出起点s到每个点x的最短路dis[x]
// 存在负环返回1 否则返回0
// 记得初始化
const int MAX_N = "Edit"; // 点数最大值
const int MAX_E = "Edit"; // 边数最大值
const int INF = 0x3F3F3F3F;
int From[MAX_E], To[MAX_E], W[MAX_E];
int dis[MAX_N], tot;
void init() {tot = 0;}
void add_edge(int u, int v, int d) {
    From[tot] = u;
    To[tot] = v;
    W[tot++] = d;
}
bool Bellman_Ford(int s, int n) {
    memset(dis, 0x3F, sizeof(dis));
    dis[s] = 0;
    for (int k = 0; k < n - 1; k++) {
        bool relaxed = 0;
        for (int i = 0; i < tot; i++) {
            int x = From[i], y = To[i];
            if (dis[y] > dis[x] + W[i]) {
                dis[y] = dis[x] + W[i];
                relaxed = 1;
            }
        }
        if (!relaxed) break;
    }
    for (int i = 0; i < tot; i++)
        if (dis[To[i]] > dis[From[i]] + W[i])
            return 1;
```

```
    return 0;
}
```

5.3.6 SPFA

```
// G[u] = mp(v, w)
// SPFA()返回0表示存在负环
const int MAXN = "Edit";
const int INF = 0x3F3F3F3F;
vector<pair<int, int> > G[MAXN];
bool vis[MAXN];
int dis[MAXN];
int inqueue[MAXN];
void init(int n) {
    for (int i = 0; i < n; i++)
        G[i].clear();
}
void add_edge(int u, int v, int w) {
    G[u].push_back(make_pair(v, w));
}
bool SPFA(int s, int n) {
    memset(vis, 0, sizeof(vis));
    memset(dis, 0x3F, sizeof(dis));
    memset(inqueue, 0, sizeof(inqueue));
    dis[s] = 0;
    queue<int> q; // 待优化的节点入队
    q.push(s);
    while (!q.empty()) {
        int x = q.front();
        q.pop();
        vis[x] = false;
        for (int i = 0; i < G[x].size(); i++) {
            int y = G[x][i].first;
            int w = G[x][i].second;
            if (dis[y] > dis[x] + w) {
                dis[y] = dis[x] + w;
                if (!vis[y]) {
                    q.push(y);
                    vis[y] = true;
                    if (++inqueue[y] >= n) return 0;
                }
            }
        }
    }
    return 1;
}
```

5.3.7 Floyd 算法

$O(n^3)$ 求出任意两点间最短路

```
const int MAXN = "Edit";
const int INF = 0x3F3F3F3F;
int G[MAXN][MAXN];
void init(int n) {
   memset(G, 0x3F, sizeof(G));
   for (int i = 0; i < n; i++)
      G[i][i] = 0;
}
void add_edge(int u, int v, int w) {
   G[u][v] = min(G[u][v], w);
}
void Floyd(int n) {
   for (int k = 0; k < n; k++)
      for (int i = 0; i < n; i++)
         for (int j = 0; j < n; j++)
            G[i][j] = min(G[i][j], G[i][k] + G[k][j]);
}
```

5.4 拓扑排序

5.4.1 邻接矩阵

```
// 存图前记得初始化
// Ans存放拓排结果，G为邻接矩阵，deg为入度信息
// 排序成功返回1，存在环返回0
const int MAXN = "Edit";
int Ans[MAXN];  // 存放拓扑排序结果
int G[MAXN][MAXN]; // 存放图信息
int deg[MAXN];  // 存放点入度信息
void init() {
   memset(G, 0, sizeof(G));
   memset(deg, 0, sizeof(deg));
   memset(Ans, 0, sizeof(Ans));
}
void add_edge(int u, int v) {
   if (G[u][v]) return;
   G[u][v] = 1;
   deg[v]++;
}
bool Toposort(int n) {
   int tot = 0;
   queue<int> que;
   for (int i = 0; i < n; ++i)
      if (deg[i] == 0) que.push(i);
```

```
    while (!que.empty()) {
        int v = que.front(); que.pop();
        Ans[tot++] = v;
        for (int i = 0; i < n; ++i)
            if (G[v][i] == 1)
                if (--deg[t] == 0) que.push(t);
    }
    if (tot < n) return false;
    return true;
}
```

5.4.2 邻接表

```
// 存图前记得初始化
// Ans排序结果，G邻接表，deg入度，map用于判断重边
// 排序成功返回1，存在环返回0
const int MAXN = "Edit";
typedef pair<int, int> PII;
int Ans[MAXN];
vector<int> G[MAXN];
int deg[MAXN];
map<PII, bool> S;
void init(int n) {
    S.clear();
    for (int i = 0; i < n; i++)G[i].clear();
    memset(deg, 0, sizeof(deg));
    memset(Ans, 0, sizeof(Ans));
}
void add_edge(int u, int v) {
    if (S[make_pair(u, v)]) return;
    G[u].push_back(v);
    S[make_pair(u, v)] = 1;
    deg[v]++;
}
bool Toposort(int n) {
    int tot = 0; queue<int> que;
    for (int i = 0; i < n; ++i)
        if (deg[i] == 0) que.push(i);
    while (!que.empty()) {
        int v = que.front(); que.pop();
        Ans[tot++] = v;
        for (int i = 0; i < G[v].size(); ++i) {
            int t = G[v][i];
            if (--deg[t] == 0) que.push(t);
        }
    }
```

```
    if (tot < n) return false;
    return true;
}
```

5.5 欧拉回路

5.5.1 判定

定理 1. *无向图 G 存在欧拉通路的充要条件是：G 为连通图，并且 G 仅有两个奇度结点或无奇度结点。*

推论 1. *(1) 当 G 是仅有两个奇度结点的连通图时，G 的欧拉通路必以此两个结点为端点。(2) 当 G 时无奇度结点的连通图时，G 必有欧拉回路。(3) G 为欧拉图（存在欧拉回路）的充要条件是 G 为无奇度结点的连通图。*

定理 2. *有向图 D 存在欧拉通路的充要条件是：D 为有向图，D 的基图连通，并且所有顶点的出度与入度都相等；或者除两个顶点外，其余顶点的出度与入度都相等，而这两个顶点中一个顶点的出度与入度只差为 1，另一个顶点的出度与入度之差为-1。*

推论 2. *(1) 当 D 除出、入度之差为 1，-1 的两个顶点之外，其余顶点的出度与入度都相等时，D 的有向欧拉通路必以出、入度之差为 1 的顶点作为始点，以出、入度之差为-1 的顶点作为终点。(2) 当 D 的所有顶点的出、入度都相等时，D 中存在有向欧拉回路。(3) 有向图 D 为有向欧拉图的充要条件是 D 的基图为连通图，并且所有顶点的出、入度都相等。*

5.5.2 求解

```
#define MAXN 200
struct stack {
    int top, node[MAXN];
} s;

int G[MAXN][MAXN]; // 邻接矩阵
int n; // 顶点个数

void dfs(int x) {
    int i;
    s.node[++s.top] = x;
    for (int i = 0; i < n; i++)
        if (G[i][x] > 0) {
            G[i][x] = G[x][i] = 0;
            dfs(i);
            break;
        }
}
void Fleury(int x) {
    int i, b;
    s.node[s.top = 0] = x;
    while (s.top >= 0) {
```

```c
        b = 0;
        for (int i = 0; i < n; i++)
            if (G[s.node[s.top]][i] > 0) {
                b = 1;
                break;
            }
        if (b == 0) {
            printf("%d ", s.node[s.top] + 1);
            s.top--;
        }
        else {
            s.top--;
            dfs(s.node[s.top + 1]);
        }
    }
    printf("\n");
}

int main() {
    int i, j;
    int m, s, t; // 边数，读入的边的起点和终点
    int degree, num, start; // 每个顶点的度、奇度顶点个数、欧拉回路的起点
    scanf("%d%d", &n, &m);
    set(G, 0);
    for (i = 0; i < m; i++) {
        scanf("%d%d", &s, &t)
        G[s - 1][t - 1] = G[t - 1][s - 1] = 1;
    }
    num = 0; start = 0;
    for (i = 0; i < n; i++) {
        degree = 0;
        for (j = 0; j < n; j++)
            degree += G[i][j];
        if (degree & 1) {
            start = i;
            num++;
        }
    }
    if (num == 0 || num == 2) Fleury(start);
    else puts("No Euler path");
    return 0;
}
```

6 计算几何

6.1 基本函数

```
#define eps 1e-8
#define pi M_PI
#define zero(x) ((fabs(x)<eps?1:0))
#define sgn(x) (fabs(x)<eps?0:((x)<0?-1:1))
#define mp make_pair
#define X first
#define Y second

struct point {
   double x, y;
   point(double a = 0, double b = 0) {x = a; y = b;}
   point operator - (const point& b) const {
      return point(x - b.x, y - b.y);
   }
   point operator + (const point &b) const {
      return point(x + b.x, y + b.y);
   }
   // 两点是否重合
   bool operator == (point& b) {
      return zero(x - b.x) && zero(y - b.y);
   }
   // 点积(以原点为基准)
   double operator * (const point &b) const {
      return x * b.x + y * b.y;
   }
   // 叉积(以原点为基准)
   double operator ^ (const point &b) const {
      return x * b.y - y * b.x;
   }
   // 绕P点逆时针旋转a弧度后的点
   point rotate(point b, double a) {
      double dx, dy; (*this - b).split(dx, dy);
      double tx = dx * cos(a) - dy * sin(a);
      double ty = dx * sin(a) + dy * cos(a);
      return point(tx, ty) + b;
   }
   // 点坐标分别赋值到a和b
   void split(double &a, double &b) {
      a = x; b = y;
   }
};

struct line {
```

```
    point s, e;
    line() {}
    line(point ss, point ee) {s = ss; e = ee;}
};
```

6.2 位置关系

6.2.1 两点间距离

```
double dist(point a, point b) {
    return sqrt((a - b) * (a - b));
}
```

6.2.2 直线与直线的交点

```
// <0, *> 表示重合; <1, *> 表示平行; <2, P> 表示交点是P;
pair<int, point> spoint(line l1, line l2) {
    point res = l1.s;
    if (sgn((l1.s - l1.e) ^ (l2.s - l2.e)) == 0)
        return mp(sgn((l1.s - l2.e) ^ (l2.s - l2.e)) != 0, res);
    double t = ((l1.s - l2.s) ^ (l2.s - l2.e)) / ((l1.s - l1.e) ^
        (l2.s - l2.e));
    res.x += (l1.e.x - l1.s.x) * t;
    res.y += (l1.e.y - l1.s.y) * t;
    return mp(2, res);
}
```

6.2.3 判断线段与线段相交

```
bool segxseg(line l1, line l2) {
    return
        max(l1.s.x, l1.e.x) >= min(l2.s.x, l2.e.x) &&
        max(l2.s.x, l2.e.x) >= min(l1.s.x, l1.e.x) &&
        max(l1.s.y, l1.e.y) >= min(l2.s.y, l2.e.y) &&
        max(l2.s.y, l2.e.y) >= min(l1.s.y, l1.e.y) &&
        sgn((l2.s - l1.e) ^ (l1.s - l1.e)) * sgn((l2.e-l1.e) ^ (l1.
            s - l1.e)) <= 0 &&
        sgn((l1.s - l2.e) ^ (l2.s - l2.e)) * sgn((l1.e-l2.e) ^ (l2.
            s - l2.e)) <= 0;
}
```

6.2.4 判断线段与直线相交

```
bool segxline(line l1, line l2) {
    return sgn((l2.s - l1.e) ^ (l1.s - l1.e)) * sgn((l2.e-l1.e) ^
        (l1.s - l1.e)) <= 0;
```

```
}
```

6.2.5 点到直线距离

```
point pointtoline(point P, line L) {
    point res;
    double t = ((P - L.s) * (L.e-L.s)) / ((L.e-L.s) * (L.e-L.s));
    res.x = L.s.x + (L.e.x - L.s.x) * t;
    res.y = L.s.y + (L.e.y - L.s.y) * t;
    return dist(P, res);
}
```

6.2.6 点到线段距离

```
point pointtosegment(point p, line l) {
    point res;
    double t = ((p - l.s) * (l.e-l.s)) / ((l.e-l.s) * (l.e-l.s));
    if (t >= 0 && t <= 1) {
        res.x = l.s.x + (l.e.x - l.s.x) * t;
        res.y = l.s.y + (l.e.y - l.s.y) * t;
    }
    else res = dist(p, l.s) < dist(p, l.e) ? l.s : l.e;
    return res;
}
```

6.2.7 点在线段上

```
bool PointOnSeg(point p, line l) {
    return
        sgn((l.s - p) ^ (l.e-p)) == 0 &&
        sgn((p.x - l.s.x) * (p.x - l.e.x)) <= 0 &&
        sgn((p.y - l.s.y) * (p.y - l.e.y)) <= 0;
}
```

6.3 多边形

6.3.1 多边形面积

```
double area(point p[], int n) {
    double res = 0;
    for (int i = 0; i < n; i++)
        res += (p[i] ^ p[(i + 1) % n]) / 2;
    return fabs(res);
}
```

6.3.2 点在凸多边形内

```
// 点形成一个凸包，而且按逆时针排序(如果是顺时针把里面的<0改为>0)
// 点的编号 : [0,n)
// -1 : 点在凸多边形外
// 0 : 点在凸多边形边界上
// 1 : 点在凸多边形内
int PointInConvex(point a, point p[], int n) {
   for (int i = 0; i < n; i++) {
      if (sgn((p[i] - a) ^ (p[(i + 1) % n] - a)) < 0)
         return -1;
      else if (PointOnSeg(a, line(p[i], p[(i + 1) % n])))
         return 0;
   }
   return 1;
}
```

6.3.3 点在任意多边形内

```
// 射线法,poly[]的顶点数要大于等于3,点的编号0~n-1
// -1 : 点在凸多边形外
// 0 : 点在凸多边形边界上
// 1 : 点在凸多边形内
int PointInPoly(point p, point poly[], int n) {
   int cnt;
   line ray, side;
   cnt = 0;
   ray.s = p;
   ray.e.y = p.y;
   ray.e.x = -100000000000.0; // -INF,注意取值防止越界
   for (int i = 0; i < n; i++) {
      side.s = poly[i];
      side.e = poly[(i + 1) % n];
      if (PointOnSeg(p, side))return 0;
      //如果平行轴则不考虑
      if (sgn(side.s.y - side.e.y) == 0)
         continue;
      if (PointOnSeg(sid e.s, r ay)) {
         if (sgn(side.s.y - side.e.y) > 0) cnt++;
      }
      else if (PointOnSeg(side.e, ray)) {
         if (sgn(side.e.y - side.s.y) > 0) cnt++;
      }
      else if (segxseg(ray, side)) cnt++;
   }
   return cnt % 2 == 1 ? 1 : -1;
```

```
}
```

6.3.4 判断凸多边形

```
//点可以是顺时针给出也可以是逆时针给出
//点的编号1~n-1
bool isconvex(point poly[], int n) {
   bool s[3];
   set(s, 0);
   for (int i = 0; i < n; i++) {
      s[sgn((poly[(i + 1) % n] - poly[i]) ^ (poly[(i + 2) % n] -
         poly[i]) ) + 1] = 1;
      if (s[0] && s[2]) return 0;
   }
   return 1;
}
```

6.3.5 小结

```
#include <stdlib.h>
#include <math.h>
#define MAXN 1000
#define offset 10000
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
#define _sign(x) ((x)>eps?1:((x)<-eps?2:0))
struct point{double x,y;};
struct line{point a,b;};

double xmult(point p1,point p2,point p0){
   return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}

//判定凸多边形,顶点按顺时针或逆时针给出,允许相邻边共线
int is_convex(int n,point* p){
   int i,s[3]={1,1,1};
   for (i=0;i<n&&s[1]|s[2];i++)
      s[_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;
   return s[1]|s[2];
}

//判定凸多边形,顶点按顺时针或逆时针给出,不允许相邻边共线
int is_convex_v2(int n,point* p){
   int i,s[3]={1,1,1};
   for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
      s[_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;
```

```
    return s[0]&&s[1]|s[2];
}

//判点在凸多边形内或多边形边上,顶点按顺时针或逆时针给出
int inside_convex(point q,int n,point* p){
    int i,s[3]={1,1,1};
    for (i=0;i<n&&s[1]|s[2];i++)
        s[_sign(xmult(p[(i+1)%n],q,p[i]))]=0;
    return s[1]|s[2];
}

//判点在凸多边形内,顶点按顺时针或逆时针给出,在多边形边上返回0
int inside_convex_v2(point q,int n,point* p){
    int i,s[3]={1,1,1};
    for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
        s[_sign(xmult(p[(i+1)%n],q,p[i]))]=0;
    return s[0]&&s[1]|s[2];
}

//判点在任意多边形内,顶点按顺时针或逆时针给出
//on_edge表示点在多边形边上时的返回值,offset为多边形坐标上限
int inside_polygon(point q,int n,point* p,int on_edge=1){
    point q2;
    int i=0,count;
    while (i<n)
        for (count=i=0,q2.x=rand()+offset,q2.y=rand()+offset;i<n;i
            ++)
            if (zero(xmult(q,p[i],p[(i+1)%n]))&&(p[i].x-q.x)*(p[(i
                +1)%n].x-q.x)<eps&&(p[i].y-q.y)*(p[(i+1)%n].y-q.y)<
                eps)
                return on_edge;
            else if (zero(xmult(q,q2,p[i])))
                break;
            else if (xmult(q,p[i],q2)*xmult(q,p[(i+1)%n],q2)<-eps&&
                xmult(p[i],q,p[(i+1)%n])*xmult(p[i],q2,p[(i+1)%n])<-
                eps)
                count++;
    return count&1;
}

inline int opposite_side(point p1,point p2,point l1,point l2){
    return xmult(l1,p1,l2)*xmult(l1,p2,l2)<-eps;
}

inline int dot_online_in(point p,point l1,point l2){
```

```
	return zero(xmult(p,l1,l2))&&(l1.x-p.x)*(l2.x-p.x)<eps&&(l1.y-
	   p.y)*(l2.y-p.y)<eps;
}

//判线段在任意多边形内,顶点按顺时针或逆时针给出,与边界相交返回1
int inside_polygon(point l1,point l2,int n,point* p){
	point t[MAXN],tt;
	int i,j,k=0;
	if (!inside_polygon(l1,n,p)||!inside_polygon(l2,n,p))
		return 0;
	for (i=0;i<n;i++)
		if (opposite_side(l1,l2,p[i],p[(i+1)%n])&&opposite_side(p[i
		   ],p[(i+1)%n],l1,l2))
			return 0;
		else if (dot_online_in(l1,p[i],p[(i+1)%n]))
			t[k++]=l1;
		else if (dot_online_in(l2,p[i],p[(i+1)%n]))
			t[k++]=l2;
		else if (dot_online_in(p[i],l1,l2))
			t[k++]=p[i];
	for (i=0;i<k;i++)
		for (j=i+1;j<k;j++){
			tt.x=(t[i].x+t[j].x)/2;
			tt.y=(t[i].y+t[j].y)/2;
			if (!inside_polygon(tt,n,p))
				return 0;
		}
	return 1;
}

point intersection(line u,line v){
	point ret=u.a;
	double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b
	   .x))
		/((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x
		   ));
	ret.x+=(u.b.x-u.a.x)*t;
	ret.y+=(u.b.y-u.a.y)*t;
	return ret;
}

point barycenter(point a,point b,point c){
	line u,v;
	u.a.x=(a.x+b.x)/2;
	u.a.y=(a.y+b.y)/2;
	u.b=c;
```

```
    v.a.x=(a.x+c.x)/2;
    v.a.y=(a.y+c.y)/2;
    v.b=b;
    return intersection(u,v);
}

//多边形重心
point barycenter(int n,point* p){
    point ret,t;
    double t1=0,t2;
    int i;
    ret.x=ret.y=0;
    for (i=1;i<n-1;i++)
        if (fabs(t2=xmult(p[0],p[i],p[i+1]))>eps){
            t=barycenter(p[0],p[i],p[i+1]);
            ret.x+=t.x*t2;
            ret.y+=t.y*t2;
            t1+=t2;
        }
    if (fabs(t1)>eps)
        ret.x/=t1,ret.y/=t1;
    return ret;
}
```

6.4 整数点问题

6.4.1 线段上整点个数

```
int OnSegment(line l) {
    return __gcd(fabs(l.s.x - l.e.x), fabs(l.s.y - l.e.y)) + 1;
}
```

6.4.2 多边形边上整点个数

```
int OnEdge(point p[], int n) {
    int i, ret = 0;
    for (i = 0; i < n; i++)
        ret += __gcd(fabs(p[i].x - p[(i + 1) % n].x), fabs(p[i].y -
             p[(i + 1) % n].y));
    return ret;
}
```

6.4.3 多边形内整点个数

```
int InSide(point p[], int n) {
    int i, area = 0;
```

```
    for (i = 0; i < n; i++)
        area += p[(i + 1) % n].y * (p[i].x - p[(i + 2) % n].x);
    return (fabs(area) - OnEdge(n, p)) / 2 + 1;
}
```

6.5 圆

6.5.1 过三点求圆心

```
point waixin(point a, point b, point c) {
    double a1 = b.x - a.x, b1 = b.y - a.y, c1 = (a1 * a1 + b1 * b1
        ) / 2;
    double a2 = c.x - a.x, b2 = c.y - a.y, c2 = (a2 * a2 + b2 * b2
        ) / 2;
    double d = a1 * b2 - a2 * b1;
    return point(a.x + (c1 * b2 - c2 * b1) / d, a.y + (a1 * c2 -
        a2 * c1) / d);
}
```

7 动态规划

7.1 子序列

7.1.1 最大子序列和

```
// 传入序列a和长度n, 返回最大子序列和
// 限制最短长度：用cnt记录长度, rt更新时判断
int MaxSeqSum(int a[], int n) {
   int rt = 0, cur = 0;
   for (int i = 0; i < n; i++) {
      cur += a[i];
      rt = rt < cur ? cur : rt;
      cur = cur < 0 ? 0 : cur;
   }
   return rt;
}
```

7.1.2 最长上升子序列 LIS

```
// 序列下标从1开始, LIS()返回长度, 序列存在lis[]中
#define N 100100
int n, len, a[N], b[N], f[N];
int Find(int p, int l, int r) {
   int mid;
   while (l <= r) {
      mid = (l + r) >> 1;
      if (a[p] > b[mid]) l = mid + 1;
      else r = mid - 1;
   }
   return f[p] = l;
}
int LIS(int lis[]) {
int len = 1;
f[1] = 1;
b[1] = a[1];
   for (int i = 2; i <= n; i++) {
      if (a[i] > b[len]) b[++len] = a[i], f[i] = len;
      else b[Find(i, 1, len)] = a[i];
   }
   for (int i = n, t = len; i >= 1 && t >= 1; i--)
      if (f[i] == t)
         lis[--t] = a[i];
   return len;
}
```

7.1.3 最长公共上升子序列 LCIS

```cpp
// 序列下标从1开始
int LCIS(int a[], int b[], int n, int m) {
   set(dp, 0);
   for (int i = 1; i <= n; i++) {
      int ma = 0;
      for (int j = 1; j <= m; j++) {
         dp[i][j] = dp[i - 1][j];
         if (a[i] > b[j]) ma = max(ma, dp[i - 1][j]);
         if (a[i] == b[j]) dp[i][j] = ma + 1;
      }
   }
   return *max_element(dp[n] + 1, dp[n] + 1 + m);
}
```

8 其他

8.1 矩阵

```
typedef long long ll;
#define REP(i,n) for(int i=0;i<n;i++)
const ll mod = 1000000007;
class Matrix {
private:
   int r, c;
   ll **m;
public:
   Matrix(int R, int C): r(R), c(C) {
      m = new ll*[r];
      REP(i, r) m[i] = new ll[c];
      REP(i, r) REP(j, c) m[i][j] = 0;
   }
   Matrix(const Matrix& B) {
      r = B.r; c = B.c; m = new ll*[r];
      REP(i, r) m[i] = new ll[c];
      REP(i, r) REP(j, c) m[i][j] = B.m[i][j];
   }
   ~Matrix() {REP(i, r) delete[] m[i]; delete[] m;}
   void e() {REP(i, r) REP(j, c) m[i][j] = !(i ^ j);}
   ll* operator [] (int p) {return m[p];}
   ll* operator [] (int p) const {return m[p];}
   Matrix& operator = (const Matrix& B) {
      r = B.r; c = B.c; this->~Matrix(); m = new ll*[r];
      REP(i, r) m[i] = new ll[c];
      REP(i, r) REP(j, c) m[i][j] = B[i][j];
      return *this;
   }
   Matrix operator * (const Matrix& B) const {
      Matrix rt(r, B.c);
      REP(i, r) REP(k, c) if (m[i][k] != 0) REP(j, B.c)
         rt[i][j] = (rt[i][j] + m[i][k] * B[k][j] % mod) % mod;
      return rt;
   }
   Matrix operator ^ (ll n) {
      Matrix rt(r, c); rt.e();
      Matrix ba(*this);
      while (n) {
         if (n & 1) rt = rt * ba;
         ba = ba * ba;
         n >>= 1;
      }
      return rt;
```

```
    }
    void out() {
        REP(i, r) REP(j, c)
        cout << m[i][j] << (j == c - 1 ? '\n' : ' ');
        cout << endl;
    }
};
```

8.2 高精度

```
// 加法 乘法 小于号 输出
struct bint {
    int l; short int w[100];
    bint(int x = 0) {
        l = x == 0; memset(w, 0, sizeof(w));
        while (x != 0) {w[l++] = x % 10; x /= 10;}
    }
    bool operator < (const bint& x) const {
        if (l != x.l) return l < x.l;
        int i = l - 1;
        while (i >= 0 && w[i] == x.w[i]) i--;
        return (i >= 0 && w[i] < x.w[i]);
    }
    bint operator + (const bint& x) const {
        bint ans; ans.l = l > x.l ? l : x.l;
        for (int i = 0; i < ans.l; i++) {
            ans.w[i] += w[i] + x.w[i];
            ans.w[i + 1] += ans.w[i] / 10;
            ans.w[i] = ans.w[i] % 10;
        }
        if (ans.w[ans.l] != 0) ans.l++;
        return ans;
    }
    bint operator * (const bint& x) const {
        bint res; int up, tmp;
        for (int i = 0; i < l; i++) {
            up = 0;
            for (int j = 0; j < x.l; j++) {
                tmp = w[i] * x.w[j] + res.w[i + j] + up;
                res.w[i + j] = tmp % 10;
                up = tmp / 10;
            }
            if (up != 0) res.w[i + x.l] = up;
        } res.l = l + x.l;
        while (res.w[res.l - 1] == 0 && res.l > 1) res.l--;
        return res;
```

```
    }
    void print() {
        for (int i = l - 1; i >= 0 ; i--)
printf("%d", w[i]);
        printf("\n");
    }
};
```