**目录 Dynamic No Programming**

[**宏定义&常量 3**](#_Toc511749319)

[**数学 3**](#_Toc511749320)

[**斐波那契 3**](#_Toc511749321)

[**快速幂 4**](#_Toc511749322)

[**素数 4**](#_Toc511749323)

[**欧几里得算法 6**](#_Toc511749324)

[**乘法逆元 6**](#_Toc511749325)

[**线性同余方程组 7**](#_Toc511749326)

[**BSGS算法 8**](#_Toc511749327)

[**欧拉函数 8**](#_Toc511749328)

[**莫比乌斯函数 9**](#_Toc511749329)

[**矩阵 10**](#_Toc511749330)

[**组合数学 10**](#_Toc511749331)

[**博弈 11**](#_Toc511749332)

[**计算几何 13**](#_Toc511749333)

[**二维 13**](#_Toc511749334)

[**凸包 16**](#_Toc511749335)

[**三维 16**](#_Toc511749336)

[**字符串 17**](#_Toc511749337)

[**KMP 17**](#_Toc511749338)

[**Manacher算法 18**](#_Toc511749339)

[**动态规划 20**](#_Toc511749340)

[**背包 20**](#_Toc511749341)

[**最长公共子序列 LCS 20**](#_Toc511749342)

[**最长递增子序列 LIS 21**](#_Toc511749343)

[**最长递增公共子序列 LICS 21**](#_Toc511749344)

[**0-1分数规划 21**](#_Toc511749345)

[**整数拆分 22**](#_Toc511749346)

[**多线程 23**](#_Toc511749347)

[**数据结构 23**](#_Toc511749348)

[**Tire树 23**](#_Toc511749349)

[**图论 25**](#_Toc511749350)

[**并查集 25**](#_Toc511749351)

[**最小生成树 25**](#_Toc511749352)

[**次小生成树 25**](#_Toc511749353)

[**强连通分量 26**](#_Toc511749354)

[**最短路径 27**](#_Toc511749355)

[**拓扑排序 31**](#_Toc511749356)

[**欧拉回路 31**](#_Toc511749357)

[**二分图染色 32**](#_Toc511749358)

[**数据结构 33**](#_Toc511749359)

[**树状数组 33**](#_Toc511749360)

[**STL模板库 35**](#_Toc511749361)

[**算法库 35**](#_Toc511749362)

[**Are you get it? 37**](#_Toc511749363)

宏定义&常量

**#define F first**

**#define S second**

**#define mp make\_pair**

**#define pf push\_front**

**#define pb push\_back**

**#define lson rt<<1,l,m**

**#define rson rt<<1|1,m+1,r**

**#define rep(i,s,e) for (int i = s; i < e; i++)**

**#define rev(i,s,e) for (int i = e-1; i >= s; i--)**

**#define lowbit(x) x&-x -树状数组**

**#define lson rt<<1,l,m -线段树左区间**

**#define rson rt<<1|1,m+1,r -线段树右区间**

**const int INF = 0x3f3f3f3f;**

**const int MAXN = 1010; //题目要求数组**

**const double e = 2.718281828;**

**const double eps = 1e-8;**

**const double PI = 3.1415926535;**

数学

**/\*\***

**\* Stirling公式**

**\* 求n!的近似值 n! ≈ sqrt(2PIn)\*(n/e)^n**

**\* n的位数 log10(n) + 1**

**\*/**

**int Stirling(int n)**

**{**

**return (log(2\*PI\*n)/2 + n\*log(n) - n) / log(10) + 1;**

**}**

斐波那契

|  |  |
| --- | --- |
| **F(0) = 0, F(1) = F(2) = 1. F(n) = F(n-1) + F(n-1) (n>=2)** | |
| **F(1) + F(2) + F(3) + …… + F(n) = F(n+2) – 1** | **证明只需加个F(2),最后减去,即减1** |
| **F(1)^2 + F(2)^2 + …… + F(n)^2 = F(n)\* F(n+1)** | **矩形面积** |
| **F(1) + F(3) + F(5) + …… + F(2\*n-1) = F(2n)** | **把F(1)当F(2)推导** |
| **F(2) + F(4) + F(6) + …… + F(2\*n) = F(2n+1) - 1** | **加个F(1)推导** |
| **F(n-1) + F(n+1) = F(2n) / F(n)** |  |
| **F(n-1) \* F(n+1) = F(n)^2 + (-1)^n** |  |

## 快速幂

**LL mul\_mod(LL a, LL b, LL m)**

**{//a\*b%m**

**LL res = 0;**

**while (b)**

**{**

**if (b & 1) res = (res + a) % m;**

**a = (a + a) % m;**

**b >>= 1;**

**}**

**return res;**

**}**

**LL pow\_mod(LL a, LL p, LL m)**

**{//a^p%m**

**LL res = 1;**

**while (p)**

**{**

**if (p & 1) res = mul\_mod(res, a, m);**

**a = mul\_mod(a, a, m);**

**p >>= 1;**

**}**

**return res;**

**}**

## 素数

**/\*\***

**\* 埃氏筛法 复杂度 O(nloglogn)**

**\*/**

**bool vis[MAXN];**

**void Eratosthenes\_sieve(int n)**

**{//-1代表是素数 0代表不是素数**

**memset(vis, -1, sizeof (vis));**

**vis[0] = vis[1] = false;**

**int tmp = (int)sqrt(n+0.5);**

**for (int i = 2; i <= tmp; ++i)**

**if(vis[i])**

**for (int j = i \* i; j <= n; j += i)**

**vis[j] = false;**

**}**

**/\*\***

**\* 欧拉筛法 复杂度 O(n)**

**\*/**

**int prime[MAXN];**

**void Euler\_sieve(int n)**

**{//prime[0]是素数个数**

**memset(prime, 0, sizeof prime);**

**for (int i = 2; i <= n; ++i)**

**{**

**if (!prime[i]) prime[++prime[0]] = i;**

**for (int j = 1; j <= prime[0] && i \* prime[j] <= n; ++j)**

**{**

**prime[i \* prime[j]] = 1;**

**if (i % prime[j] == 0) break;**

**}**

**}**

**}**

**/\*\***

**\* Miller\_Rabin算法素数检测**

**\* 若m是素数 a^(m-1) % m = 1**

**\* p = m-1 = 2^r\*s**

**\* a^s % m == 1 || a^(2^r\*s) % m == n-1通过检测**

**\* 2,7,61 (2^31) 2,3,5,7,11 (2^63)**

**\*/**

**bool Miller\_Rabbin(LL a, LL m)**

**{//p=m-1=2^r\*s a^s%m==1 || a^(2^r\*s)%m==n-1**

**LL r = 0, s = m - 1;**

**while (~s & 1) //找到奇数**

**{**

**s >>= 1;**

**r++;**

**}**

**LL tmp = pow\_mod(a, s, m);**

**if (tmp == 1) return true; //a^s%n==1通过素数检测**

**for (int i = 0; i < r; ++i, tmp = mul\_mod(tmp, tmp, m))**

**if (tmp == m-1) //a^s^2^r%n==n-1通过检测**

**return true;**

**return false;**

**}**

**bool isprime(LL n)**

**{**

**if (n < 2) return false;**

**int prime[5] = {2, 3, 5, 7, 11};**

**for (int i = 0; i < 5; ++i)**

**{**

**if (prime[i] == n) return true; //素数表元素不能通过测试**

**if (n % prime[i] == 0) return false;**

**if (!Miller\_Rabbin(prime[i], n)) return false;**

**}**

**return true;**

**}**

欧几里得算法

**/\*\***

**\* 欧几里得算法 复杂度 O(log max(a,b))**

**\*/**

**LL gcd(LL a, LL b)**

**{**

**return b ? gcd(b, a%b) : a;**

**}**

**/\*\***

**\* 扩展欧几里得算法 复杂度 O(log max(a,b))**

**\* 返回ax + by = gcd(a,b)的一组解 x0,y0**

**\*通解：**

**\* x = x0 + (b/d) \*t 等同于增加(b/d) \*t**

**\* y = y0 – (a/d) \*t 等同于减少(a/d) \*t**

**\* 注：d=gcd(a,b)**

**\* x = x0 + k \* b / d 等同于增加k\*a\*b/d**

**\* y = y0 – k \* a / d 等同于减少k\*a\*b/d**

**\*/**

**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 -= a/b\*x;**

**}**

**/\*\***

**\*助思考式：**

**\*x=y1;**

**\*y=x1-a/b\*y1;**

**\*/**

乘法逆元

**/\*\***

**\* 对于正整数a和m,如am互质 ax ≡ 1(mod m)**

**\* X的最小正整数解叫做a模m的逆元，(a,m为负不影响，只要保证X>0)**

**\* ax ≡ 1(mod m) => x ≡ 1/a(mod m)**

**\* 如am互质 a^(φ(m)-1) ≡ 1(mod m)**

**\* 如果m是素数,可用费马小定理求逆元为a^(m-2) mod m**

**\*/**

**LL inv(LL a, LL m)**

**{//不存在逆,返回-1**

**LL d, x, y;**

**exgcd(a, m, d, x, y);**

**return d == 1 ? (x + m) % m : -1;**

**}**

**LL inv[MAXN];**

**void getinv()**

**{//M 模数**

**inv[1] = 1;**

**for (int i = 2; i < MAXN; i++)**

**inv[i] = (M - M / i) \* inv[M % i] % M;**

**}**

线性同余方程组

**x ≡ a1 (mod m1)**

**x ≡ a2 (mod m2)**

**x ≡ a3 (mod m3)**

**⁝**

**/\*\***

**\* 中国剩余定理**

**\* m1,m2,m3,……,mr 两两互质**

**\* M = m1\*m2\*m3\*……\*mn, Mi = M/mi**

**\* Mi\*Pi ≡ 1 (mod mi) (Mi\*Pi只模mi不等于零)**

**\* a1M1P1 + a2M2P2 + …… + arMrPn就是方程组的解**

**\*/**

**LL china(int n)**

**{**

**LL ans = 0, M = 1, Mi, d, x, y;**

**for(int i = 0; i < n; ++i) M \*= m[i];**

**for(int i = 0; i < n; ++i)**

**{**

**Mi = M / m[i];**

**exgcd(Mi, m[i], d, x, y);**

**ans += a[i] \* Mi \* x;**

**}**

**return (ans % M + M) % M;**

**}**

**x ≡ b1 (mod m1)，x ≡ b2 (mod m2)**

**令m = [m1,m2],d = (m1,m2)**

**则最终解 X ≡ b1 (mod m1)，X ≡ b2 (mod m2)**

**X ≡ x (mod m1), X ≡ x (mod m2) 这样需要满组两个方程**

**合并方程 X ≡ x (mod m) 满足此方程等同于满足上面两个方程**

**m1 \* y1 ≡ (b2 – b1) (mod m2)**

**若d | (b2 – b1)有解，否则无解。**

**求出 m1 \* y1 ≡ d (mod m2) 的一个解y1 => 原式解 y1 =(b2 – b1) / d \* y1**

**最小非负整数解(y1 % (m2/d) + (m2/d)) % (m2/d)，方程x = m1 \* y1 + b1**

**最终解X = x (mod m)**

**int n;**

**ll a1,r1,a2,r2,d,x,y;**

**cin>>n;**

**cin>>a1>>r1;**

**bool flag = 1;**

**for (int i = 1; i < n; ++i)**

**{**

**cin>>a2>>r2;**

**ll a = a1,b = a2,c = r2 - r1;**

**ex\_gcd(a,b,d,x,y);**

**if (c%d) {flag = 0;continue;}**

**ll t = b/d;**

**x = (x \* (c/d)%t + t) %t;**

**r1 = a1 \* x + r1;**

**a1 = a1 \* (a2/d);**

**}**

**if (flag) cout<<r1<<endl;**

**else cout<<-1<<endl;**

BSGS 算法

**/\*\***

**\* BSGS算法 a^x % m = b 复杂度O(n^0.5logn)**

**\* 先枚举a^0, a^1, a^2, … , a^m-1**

**\* 这些项乘a^m => a^m, a^m+1, a^m+2, … , a^2m-1**

**\* 左右同乘 a^-m => ba^-m, 2m,3m,4m, … 同理**

**\*/**

**LL BSGS(LL a, LL b, LL m)**

**{**

**map<LL,LL> x;**

**map<LL,bool> y;**

**LL e = 1, r = ceil(sqrt(m)), v = pow\_mod(a, m-r-1, m);**

**for (int i = 0; i < r; ++i, e = e \* a % m)**

**if (!y[e]) x[e] = i, y[e] = 1;**

**for (int i = 0; i <= r; ++i, b = b \* v % m)**

**if (y[b]) return i \* r + x[b];**

**return -1;**

**}**

## 欧拉函数

**/\*\***

**\* 欧拉函数 小于n的正整数中与n互质的数的数目 φ(1) = 1**

**\* φ(n) = n \* (1 - 1/p1) \* (1 - 1/p2) \* ……**

**\*/**

**LL phi(LL n)**

**{**

**LL res = n;**

**for (int i = 2; i \* i <= n; ++i)**

**if (n % i == 0)**

**{**

**res -= res/i;**

**while (n % i == 0) n /= i;**

**}**

**if (n > 1) res -= res/n;**

**return res;**

**}**

**int phi[MAXN];**

**void get\_phi(int n)**

**{**

**memset(phi, 0, sizeof phi);**

**phi[1] = 1;**

**for (int i = 2; i <= n; ++i) if (!phi[i])**

**for (int j = i; j <= n; j += i)**

**{**

**if (!phi[j]) phi[j] = j;**

**phi[j] -= phi[j] / i;**

**}**

**}**

## 莫比乌斯函数

**/\*\***

**\***

**\*/**

**int mu[MAXN];**

**int prime[MAXN];**

**void get\_mu(int n)**

**{**

**memset(prime, 0, sizeof prime);**

**mu[1] = 1;**

**for (int i = 2; i <= n; ++i)**

**{**

**if (!prime[i]) prime[++prime[0]] = i, mu[i] = -1;**

**for (int j = 1; j <= prime[0] && i \* prime[j] <= n; ++j)**

**{**

**prime[i \* prime[j]] = 1;**

**if (i % prime[j] == 0)**

**{**

**mu[i \* prime[j]] = 0;**

**break;**

**}**

**else**

**mu[i \* prime[j]] = -mu[i];**

**}**

**}**

**}**

## 矩阵

**struct mat**

**{**

**int a[MAXN][MAXN];**

**mat() { memset(a, 0, sizeof a); }**

**mat operator \* (const mat &b)**

**{**

**mat res;**

**for (int i = 0; i < sz; ++i)**

**for (int j = 0; j < sz; ++j) if (a[i][j])**

**for (int k = 0; k < sz; ++k)**

**res.a[i][k] = (res.a[i][k] + a[i][j] \* b.a[j][k]) % M;**

**return res;**

**}**

**};**

**mat mat\_pow(mat &a, LL p)**

**{**

**mat res;**

**for (int i = 0; i < sz; ++i)**

**res.a[i][i] = 1;**

**while (p)**

**{**

**if (p & 1) res = res \* a;**

**a = a \* a;**

**p >>= 1;**

**}**

**return res;**

**}**

**矩阵和**

**A + A^2 + A^3 + …… + A^k**

|  |  |  |  |
| --- | --- | --- | --- |
| **A^k** | **A** | **0** | **A^(k-1)** |
| **S(k-1)** | **I** | **I** | **S(k-2)** |

**S(k) = I + A + A^2 + A^3 + …… + A^k**

**a×b = 0 两向量平行**

**a·b = 0 垂直 <= 0**

## 组合数学

1. **A(n,r) = n! / (n-r)!**
2. **C(n,r) = n! / (n-r)! / r!**
3. **C(n,r) = C(n,n-r)**
4. **C(n,r) = C(n-1,r) + C(n-1,r-1)**
5. **C(n+r+1,r) = C(n+r,r) + C(n+r-1,r-1) + …… + C(n,0)**

**= C(n+r,n) + C(n+r-1,n) + …… + C(n,n)**

1. **C(n,k) \* C(k,r) = C(n,r) \* C(n-r,k-r) (k >= r)**
2. **Catalan数**

**h(n) = C(n2,n)-C(2n,n-1)=h(0)\*h(n-1)+h(1)+h(n-2)+……+h(n-1)h(0)**

**= (4\*n-2)/(n+1) \*h (n-1)**

**/\*\* `**

**\* 组合数C(n,r)**

**\*/**

**ULL com(int n,int r)**

**{**

**if (r > n || r < 0) return 0;**

**ULL res = 1;**

**if (r > n - r) r = n - r;**

**for (int i = n, j = 1; j <= r; --i,++j)**

**res = res \* i / j;**

**return res;**

**}**

## 博弈

**/\*\***

**\* 巴什博弈**

**\* 一堆n个物品,两个人轮流从这堆物品中取物,每次至少取一个,最多取m个.**

**\*/**

**if (n % (m+1) != 0) //拿走最后一个先手胜**

**if (n % (m+1) != 1) //拿走最后一个先手输**

**/\*\***

**\* 威佐夫博弈**

**\* 有两堆物品，两个人轮流从某一堆或同时从两堆中取同样多的物品，每次至少取一个**

**\* bn=an+n是奇异局势 (自己取时必输)**

**\* ０ １　３　４　６　　８　　９　　１１　１２**

**\* ０　２　５　７　１０　１３　１５　１８　２０**

**\* 根据Betty定理**

**\* 设an=[αn],bn=[βn], 有an+n = [(α+1)n] = [βn]**

**\* 解方程 1/α + 1/(α+1) = 1, 解得α = (sqrt5+1) / 2**

**\*/**

**if (a > b) a ^= b ^= a ^= b;**

**int c = (sqrt(5)+1) \* (b-a) / 2;**

**if (a != c) //bn - an = n会输**

**/\*\***

**\* 尼姆博弈**

**\* 有三堆物品，两个人轮流从某一堆取任意多的物品，规定每次至少取一个。**

**\* (a,b,c)表示某种局势,则a^b^c=0是奇异局势**

**\* 可扩展为n堆**

**\*/**

**/\*\***

**\* 斐波那契博弈**

**\* 有一堆物品,两人轮流取物品,先手最少取一个,至多无上限，但不能把物品取完**

**\* 之后每次取的物品数不能超过上一次取的物品数的二倍且至少为一件**

**\* 取走最后一件物品的人获胜**

**\* n不是斐波那契数先手胜**

**\*/**

**运用SG定理（Sprague-Grundy定理（SG定理）： 游戏和的SG函数等于各个游戏SG函数的Nim和。）然后将n个游戏的SG值XOR，如果结果为零那么先手必败，否则先手必胜。**

**int grundy(int x)**

**{**

**SET = {};**

**for(int j=1.2....k)**

**{**

**if(a\_j <= x)**

**将grundy(x - a\_j)加入SET；**

**}**

**return 最小的不属于SET的非负整数;**

**}**

**SG函数模板**

**void grundy(int n)**

**{**

**memset(SG,0,sizeof(SG));**

**for(int i=1;i<=n;i++)**

**{**

**set<int> S;//每次都要初始化后继状态的集合**

**for(int j=1;f[j]<=i && j<=16;j++)**

**S.insert(SG[i-f[j]]);**

**int l = 0;**

**while(S.count(l)) l++;//找到第一个不存在集合的最小的数**

**SG[i] = l;**

**}**

**}**

计算几何

## 二维

**int sgn(double x)**

**{**

**if (fabs(x) < eps) return 0;**

**if (x < 0) return -1;**

**return 1;**

**}**

**typedef struct point**

**{**

**double x, y;**

**point (double x = 0, double y = 0): x(x), y(y) {}**

**bool operator < (const point &b) const**

**{**

**return x < b.x || (x == b.x && y < b.y);**

**}**

**point operator + (const point &b)**

**{//向量+向量=向量 点+向量=点**

**return point(x + b.x, y + b.y);**

**}**

**point operator - (const point &b)**

**{//点-点=向量ab (!注意a-b=ab 不是ba)**

**return point(b.x - x, b.y - y);**

**}**

**point operator \* (double v)**

**{//向量\*数=向量**

**return point(x \* v, y \* v);**

**}**

**point operator / (double v)**

**{//向量/数=向量**

**return point(x / v, y / v);**

**}**

**double operator \* (const point &b)**

**{//点乘a·b = axbx + ayby = |a||b|cos<a,b> 等于零时垂直**

**return x \* b.x + y \* b.y;**

**}**

**double operator ^ (const point &b)**

**{//叉乘a×b = axby – aybx = |a||b|sin<a,b> 等于零时平行**

**return x \* b.y - y \* b.x;**

**}**

**void trans(double B)**

**{//逆时针旋转角度B (弧度值)**

**double tx = x, ty = y;**

**x = tx \* cos(B) - ty \* sin(B);**

**y = tx \* sin(B) + ty \* cos(B);**

**}**

**friend double dist(point a)**

**{//向量长度 or 两点间距离(点-点)**

**return sqrt(a \* a);**

**}**

**} P;**

**/\*\***

**\* 判断点是否在线段上**

**\*/**

**bool on\_seg(P &a, P &b1, P &b2)**

**{**

**return**

**sgn((a-b1) ^ (a-b2)) == 0 &&**

**sgn((a.x - b1.x) \* (a.x - b2.x)) <= 0 &&**

**sgn((a.y - b1.y) \* (a.y - b2.y)) <= 0;**

**}**

**/\*\***

**\* 返回点到线段最近的点**

**\* 去掉两个if语句为点到直线最近的点**

**\*/**

**P closest\_point(P &a, P &b1, P &b2)**

**{**

**double t = ((b1-a) \* (b1-b2)) / ((b1-b2) \* (b1-b2));**

**if (t < 0) return b1;**

**if (t > 1) return b2;**

**return b1 + ((b1-b2) \* t);**

**}**

**/\*\***

**\* 判断线段是否相交**

**\* 快速排斥 与 跨立实验**

**\*/**

**bool seg\_inter(P &a1, P &a2, P &b1, P &b2)**

**{**

**return min(a1.x, a2.x) < max(b1.x, b2.x) &&**

**min(a1.y, a2.y) < max(b1.y, b2.y) &&**

**min(b1.x, b2.x) < max(a1.x, a2.x) &&**

**min(b1.y, b2.y) < max(a1.y, a2.y) &&**

**sgn((b1-a1) ^ (b1-b2)) \* sgn((b1-a2) ^ (b1-b2)) <= 0 &&**

**sgn((a1-b1) ^ (a1-a2)) \* sgn((a1-b2) ^ (a1-a2)) <= 0;**

**}**

**/\*\***

**\* 判断线段a1a2与直线b1b2是否相交**

**\*/**

**bool seg\_inter\_line(P &a1, P &a2, P &b1, P &b2)**

**{**

**return sgn((b1-a1) ^ (b1-b2)) \* sgn((b1-a2) ^ (b1-b2)) <= 0;**

**}**

**/\*\***

**\* 两直线交点 0重合，1平行，2是返回交点**

**\*/**

**int line\_inter(P &a1, P &a2, P &b1, P &b2, P &res)**

**{**

**if (sgn((a1-a2) ^ (b1-b2)) == 0)**

**{**

**if (sgn((a1-a2) ^ (a1-b2)) == 0) return 0;**

**return 1;**

**}**

**double t = ((a1-a2) ^ (b1-a1)) / ((a1-a2) ^ (b1-b2));**

**res = b1 + ((b1-b2) \* t);**

**return 2;**

**}**

**/\*\***

**\* 计算多边形面积 按点的编号依次相连**

**\*/**

**double poly\_area(P p[], int n)**

**{**

**double res = 0;**

**p[n] = p[0];**

**for (int i = 0; i < n; ++i)**

**res += p[i] ^ p[i+1];**

**return fabs(res/2);**

**}**

**/\*\***

**\* 判断点在凸多边形内**

**\* 点形成凸包，逆时针排序(顺时针把里面的<0改为>0)**

**\* -1：在凸多边形外 0：在边界上 1：在凸多边形内**

**\*/**

**int in\_convex\_poly(P &a, P p[], int n)**

**{**

**p[n] = p[0];**

**for (int i = 0; i < n; ++i)**

**{**

**if (sgn((a-p[i]) ^ (a-p[i+1])) < 0) return -1;**

**if (on\_seg(a, p[i], p[i+1])) return 0;**

**}**

**return 1;**

**}**

## 凸包

**/\*\***

**\* Graham扫描线求凸包 O(nlogn)**

**\* 水平排序**

**\*/**

**int Graham(P p[], int n, P res[])**

**{**

**sort(p, p+n);**

**int len = 0;**

**for (int i = 0; i < n; ++i)**

**{**

**while (len > 1 && sgn((res[len-2]-res[len-1]) ^ (res[len-2]-p[i])) <= 0) len--;**

**res[len++] = p[i];**

**}**

**int tmp = len;**

**for (int i = n-2; i >= 0; --i)**

**{**

**while (len > tmp && sgn((res[len-2]-res[len-1]) ^ (res[len-2]-p[i])) <= 0) len--;**

**res[len++] = p[i];**

**}**

**if (n > 1) len--;**

**return len;**

**}**

## 三维

**typedef struct point**

**{**

**double x, y, z;**

**point (double x = 0, double y = 0, double z=0):**

**x(x), y(y), z(z) {}**

**point operator - (const point &b)**

**{**

**return point(b.x-x, b.y-y, b.z-z);**

**}**

**double operator \* (const point &b)**

**{**

**return x\*b.x + y\*b.y + z\*b.z;**

**}**

**point operator ^ (const point &b)**

**{**

**return point(y\*b.z - z\*b.y, z\*b.x - x\*b.z, x\*b.y - y\*b.x);**

**}**

**} P;**

**/\*\***

**\* 四点共面 混合积(体积)为零**

**\*/**

**bool surface(P &a, P &b, P &c, P &d)**

**{**

**double ans = ((b-a) ^ (c-a)) \* (d-a);**

**return !sgn(ans);**

**}**

字符串

KMP

**void getnext(char t[], int next[])**

**{**

**int j = 0, k = next[0] = -1;**

**while (t[j + 1])**

**{**

**while (k ^ -1 && t[j] ^ t[k])**

**k = next[k];**

**next[++j] = ++k;**

**}**

**}**

**void getnextv(char t[], int nextv[])**

**{**

**int j = 0, k = nextv[0] = -1;**

**while (t[j + 1])**

**{**

**while (k ^ -1 && t[j] ^ t[k])**

**k = nextv[k];**

**if (t[++j] == t[++k])**

**nextv[j] = nextv[k];**

**else**

**nextv[j] = k;**

**}**

**}**

**bool kmp(char s[], char t[])**

**{**

**int i = 0, j = 0, next[N];**

**getnext(t, next);**

**while (s[i] && t[j])**

**{**

**While (j ^ -1 && s[i] ^ t[j])**

**j = next[j];**

**++i, ++j;**

**}**

**return !t[j];//匹配成功**

**}**

**len - next[len];//周期**

**len % (len-next[len]) //完整周期差的数**

Manacher算法

**/\*\***

**\* Manacher算法 复杂度O(n)**

**\* 求最长回文子串**

**\* ababa => $#a#b#a#b#a#**

**\*/**

int **len[MAXN], ans = 0;**

char **str[MAXN];**

void **getstr(**char **\*s)**

**{**

int **k = 0;**

**str[k++] = '$';**

for **(**int **i = 0; s[i]; ++i)**

**str[k++] = '#', str[k++] = s[i];**

**str[k++] = '#';**

**str[k] = '\0';**

**}**

void **Manacher(**char **\*s)**

**{**

**getstr(s);**

int **mx = 0, id = 0;**

for **(**int **i = 1; str[i]; ++i)**

**{**

**len[i] = i < mx ?** min**(len[2\*id - i], mx - i) : 1;**

while **(str[i + len[i]] == str[i - len[i]]) ++len[i];**

if **(i + len[i] > mx) mx = i + len[i], id = i;**

if **(len[i]-1 > ans) ans = len[i] - 1;**

**}**

**}**

动态规划

背包

**C费用 W价值 M数量 V最大体积**

**void ZeroOnePack(int c, int w)**

**{**

**for (int v = V; v >= c; --v)**

**dp[v] = max(dp[v], dp[v - c] + w);**

**}**

**void CompletePack(int c, int w)**

**{**

**for (int v = c; v <= V; ++v)**

**dp[v] = max(dp[v], dp[v - c] + w);**

**}**

**void MultiplePack(int c, int w, int m)**

**{**

**if (c \* m >= V)**

**{**

**CompletePack(c, w);**

**return;**

**}**

**for (int k = 1; k < m; m -= k, k <<= 1)**

**ZeroOnePack(k \* c, k \* w);**

**ZeroOnePack(m \* c, m \* w);**

**}**

**/\*\***

**\* 单调队列优化多重背包算法 复杂度 O(nV)**

**\*/**

**void MultiplePack()**

**{//不等于-1的是可以组成的**

**memset(dp, -1, sizeof dp); dp[0] = 0;**

**for (int i = 0; i < n; ++i)**

**for (int v = 0; v <= V; ++v)**

**{**

**if (dp[v] >= 0) dp[v] = m[i];**

**else if (v >= c[i]) dp[v] = max(dp[v-c[i]] - 1, -1);**

**}**

**}**

## 最长公共子序列 LCS

**void LCS(char s[], char t[])**

**{**

**int len1 = strlen(s) ,len2 = strlen(t);**

**for (int i = 0; i < len1; ++i)**

**for (int j = 0; j < len2; ++j)**

**{**

**if(s[i] == t[j])**

**dp[i + 1][j + 1] = dp[i][j] + 1;**

**else**

**dp[i + 1][j + 1] = max(dp[i][j + 1], dp[i + 1][j]);**

**}**

**cout<<dp[len1][len2];**

**}**

## 最长递增子序列 LIS

**//复杂度 O(nlogn)**

**const int INF = 0x3f3f3f3f;**

**void LIS()**

**{//最长上升子序列**

**memset(dp, 0x3f, sizeof dp);**

**for (int i = 0; i < n; ++i)**

**\*lower\_bound(dp, dp+n, a[i]) = a[i];**

**cout<<lower\_bound(dp, dp+n, INF) - dp;**

**}**

**void LNDS()**

**{//最长不下降子序列**

**memset(dp, INF, sizeof dp);**

**for (int i = 0; i < n; ++i)**

**\*upper\_bound(dp, dp+n, a[i]) = a[i];**

**cout<<lower\_bound(dp, dp+n, INF) - dp;**

**}**

## 最长递增公共子序列 LICS

## 0-1分数规划

**/\***

**\* 0-1 分数规划**

**\* a1 \* x1 + a2 \* x2 + ... + an \* xn**

**\* R = ---------------------------------**

**\* b1 \* x1 + b2 \* x2 + ... + bn \* xn**

**\* a[i]表示选取i的收益,b[i]表示选取i的代价 x[i]代表是否选取i(0 or 1)**

**\* 从n个里选k个使R最大**

**\* 先设F(R) = Σ((a[i] - R \* b[i]) \* x[i])**

**\* F[R] >= 0 说有存在比R大的解**

**\*/**

**int n, k;**

**int a[MAXN], b[MAXN];**

**double d[MAXN];**

**bool check(double R)**

**{**

**for (int i = 0; i < n; i++)**

**d[i] = a[i] - R \* b[i];**

sort**(d, d+n);//从小到大排序，再反向选取**

**double res = 0;**

**for (int i = k; i < n; i++)**

**sum += d[i];**

**return sum >= 0;**

**}**

**double slove()**

**{**

**double l = 0, r = 1;**

**while(r - l > esp)**

**{**

**double m = (l + r) / 2;**

**if(check(m)) l = m;**

**else r = m;**

**}**

**return r;**

**}**

## 整数拆分

**拆分：把正整数分解成若干正整数的和。**

**拆分数：不同拆分法的总数。**

**定理：重复次数不超过k次的拆分数 等于 分解数中无k+1倍数的拆分数。**

**Ferrers图像**

**定理：拆分成k个数的拆分数 等于 拆分成最大数为k的拆分数。**

**定理：重复次数不超过k次的拆分数 等于 拆分成最大数不超过k的拆分数。**

**定理：拆分互不相同若干奇数的拆分数 等于 拆分成自共轭的Ferrers图像的拆分数。**

**定理：N拆分成不超过k个数的拆分数 等于 N+k拆分成k个数的拆分数。**

**void solve()**

**{**

**dp[0][0] = 1;//j的i划分数**

**for (int i = 1; i <= m; ++i)**

**for (int j = 0; j <= n; ++j)**

**{**

**if (j < i)// 1的(2.3.4……)划分数 = 1的1划分数**

**dp[i][j] = dp[i-1][j];**

**else//最后的+1和 每个+1**

**dp[i][j] = dp[i-1][j] + dp[i][j-i];**

**}**

**}**

**1.整数N拆分成k个数的拆分数**

## 多线程

**int n, m;**

**int a[MAXN][MAXN], dp[2][MAXN][MAXN];**

**void solve()**

**{**

**int cnt = 0;**

**for (int k = 2; k <= n+m; ++k)**

**{**

**cnt ^= 1;**

**for (int i = 1; i <= n && i < k; ++i)**

**for (int j = i; j <= n && j < k; ++j)**

**{**

**dp[cnt][i][j] =**

**max( max(dp[cnt^1][i][j], dp[cnt^1][i-1][j-1]) ,**

**max(dp[cnt^1][i-1][j], dp[cnt^1][i][j-1]) );**

**if (i == j)**

**dp[cnt][i][j] += a[i][k-i];**

**else**

**dp[cnt][i][j] += a[i][k-i] + a[j][k-j];**

**}**

**}**

**cout<<dp[cnt][n][n];**

**}**

数据结构

## Tire树

**struct tire**

**{**

**//date**

**tire \*n[26];**

**tire() { memset(n, 0, sizeof n); //date初始化 }**

**} \*rt = new tire;**

**void insert(tire \*p, char s[])**

**{**

**for (int i = 0; s[i]; ++i)**

**{**

**int ch = s[i] - 'a';**

**if (p->n[ch] == 0) p->n[ch] = new tire;**

**p = p->n[ch];**

**}**

**//修改date**

**}**

**void query(tire \*p, char s[])**

**{**

**for (int i = 0; s[i]; ++i)**

**{**

**int ch = s[i] - 'a';**

**if (p->n[ch] == 0) return;**

**p = p->n[ch];**

**}**

**//返回date**

**}**

图论

## 并查集

**/\*\***

**\* 并查集算法 复杂度 O(< logn)**

**\*/**

**const int MAXN = 550;**

**int father[MAXN];//初始化为-1**

**int Find(int x)**

**{**

**return father[x] == -1 ? x : father[x] = Find(father[x]);**

**}**

**bool Unite(int x, int y)**

**{**

**x = Find(x), y = Find(y);**

**if(x == y) return false;**

**father[y] = x;**

**return true;**

**}**

## 最小生成树

**Prim算法 -> Dijkstar算法**

**Kruskal算法 -> 并查集**

**Or -> 次小生成树**

## 次小生成树

**/\*\***

**\* 次小生成树**

**\* 先求最小生成树**

**\* 然后枚举所有不在MST中的边,加入到MST中会形成回路**

**\* 替换掉回路中最大边权的边,更新答案**

**\*/**

**int pre[MAXN]; /前驱**

**int dis[MAXN];**

**bool vis[MAXN];**

**bool use[MAXN][MAXN];**

**int maxv[MAXN][MAXN];**

**int cost[MAXN][MAXN];**

**int Prim(int s)**

**{//s:起点**

**int ans = 0;**

**clr(dis, INF); clr(vis, 0); clr(use, 0); clr(maxv, 0);**

**dis[s] = 0; fill(pre, pre + MAXN, s);**

**for (int i = 0; i < n; ++i)**

**{**

**int u = -1;**

**for (int j = 0; j < n; ++j)**

**if (!vis[j] && (u == -1 || dis[u] > dis[j]))**

**u = j;**

**if (dis[u] == INF) return -1; //不连通**

**ans += dis[u];**

**use[u][pre[u]] = use[pre[u]][u] = vis[u] = 1;**

**for (int v = s; v < n; ++v)**

**{**

**if (vis[v])**

**maxv[u][v] = maxv[v][u] = max(maxv[v][pre[u]], dis[u]);**

**if(!vis[v] && dis[v] > cost[u][v])**

**{**

**dis[v] = cost[u][v];**

**pre[v] = u;**

**}**

**}**

**}**

**return ans;**

**}**

**int smst(int s, int ans)**

**{//s:起点 ans:最小生成树的值**

**int minv = INF;**

**for (int i = 0; i < n; ++i)**

**for (int j = i+1; j < n; ++j)**

**if (cost[i][j] != INF && !use[i][j])**

**minv = min(minv, ans + cost[i][j] - maxv[i][j]);**

**if(minv == INF) return -1;//不存在**

**return minv;**

**}**

## 强连通分量

**/\*\***

**\* 强连通分量Tarjan算法 复杂度 O(V+E)**

**\*/**

**stack<int>sta;**

**vector<int>E[MAXN];**

**int idx; //时间戳,搜索的顶点个数**

**int num; //强连通分量个数**

**int dfn[MAXN]; //被搜索次序**

**int low[MAXN]; //追溯到的最早次序**

**bool vis[MAXN];**

**void init()**

**{**

**clr(dfn, 0); clr(low, 0); clr(vis, 0);**

**idx = sum = 0;**

**for(int i = 0; i <= n; ++i) E[i].**clear**();**

**while(!sta.empty()) sta.pop();**

**}**

**void Tarjan(int u)**

**{**

**dfn[u] = low[u] = ++idx;**

**sta.push(u); vis[u]=1;**

**for (int i = 0; i < (int)E[u].size(); ++i)**

**{**

**int v = E[u][i];**

**if (!dfn[v])**

**Tarjan(v), low[u] = min(low[u], low[v]);**

**else if (vis[v])**

**low[u] = min(low[u], dfn[v]);**

**}**

**if (low[u] == dfn[u])**

**{**

**num++;**

**while (!sta.empty())**

**{**

**int v = sta.top();**

**sta.pop(); vis[v]=0;**

**if(u == v) break;**

**}**

**}**

**}**

## 最短路径

**/\*\***

**\* 单源最短路Dijkstar算法 复杂度 O(V^2)**

**\* 权值必须非负**

**\*/**

**struct Edge { int to, cost; };**

**vector<Edge> E[MAXN];**

**int dis[MAXN];**

**bool vis[MAXN];**

**void Dijkstra(int s, int n)**

**{//s:起点 以0和1为起点的时候不用修改代码**

**memset(vis, 0, sizeof vis);**

**memset(dis, 0x3f, sizeof dis);**

**dis[s] = 0;**

**for (int i = s; i <= n; ++i)**

**{**

**int u = -1;**

**for (int j = s; j <= n; ++j)**

**if (!vis[j] && (u == -1 || dis[u] > dis[j]))**

**u = j;**

**if (dis[u] == INF) return;**

**vis[u] = 1;**

**for (int j = 0; j < (int)E[u].size(); ++j)**

**{**

**int v = E[u][j].to;**

**if (dis[v] > dis[u] + E[u][j].cost)**

**dis[v] = dis[u] + E[u][j].cost;**

**}**

**}**

**}**

**/\*\***

**\* 优先队列优化Dijkstar算法 复杂度 O(ElogE)**

**\* 权值必须非负**

**\* 注意对vector<Edge> E[MAXN]初始化后加边**

**\*/**

**struct Edge**

**{**

**int to, cost;**

**Edge(int \_to = 0, int \_cost = 0): to(\_to), cost(\_cost) {}**

**bool operator < (const Edge &x) const { return cost > x.cost; }**

**};**

**vector<Edge> E[MAXN];**

**int dis[MAXN];**

**void Dijkstar(int s)**

**{//s:起点**

**memset(dis, 0x3f, sizeof dis);**

**dis[s] = 0;**

**priority\_queue<Edge> que;**

**que.push(Edge(s, 0));**

**while (!que.empty())**

**{**

**int u = que.top().to, val = que.top().cost;**

**que.pop();**

**if (dis[u] < val) continue;**

**for (int i = 0; i < (int)E[u].size(); ++i)**

**{**

**int v = E[u][i].to;**

**if (dis[v] > dis[u] + E[u][i].cost)**

**{**

**dis[v] = dis[u] + E[u][i].cost;**

**que.push(Edge(v, dis[v]));**

**}**

**}**

**}**

**}**

**/\*\***

**\* 单源最短路Bellman\_Ford算法 复杂度 O(VE)**

**\* 可以处理负边权**

**\* 可以判断是否存在负环回路**

**\* 最短路径最多经过n-1个点,只需n-1次循环.**

**\*/**

**struct Edge { int u, v, cost; } E[MAXN];**

**int dis[MAXN];//顶点数**

**bool Bellman\_Ford(int s, int n, int m)**

**{//s:起点 n:顶点 m:边**

**memset(dis, 0x3f, sizeof dis);**

**dis[s] = 0;**

**for (int i = 1; i <= n; ++i)**

**{**

**bool update = false;**

**for (int j = 0; j < m; ++j)**

**if (dis[E[j].v] > dis[E[j].u] + E[j].cost)**

**{**

**dis[E[j].v] = dis[E[j].u] + E[j].cost;**

**update = true;**

**}**

**if (!update) return true;//没有负环回路,能求出最短路**

**}**

**return false;//循环n次还能更新,存在负环回路**

**}**

**/\*\***

**\* 单源最短路SPFA算法 复杂度 O(kE)**

**\* k是所有顶点平均加入队列次数,一般k<=2**

**\* SLF优化(常用) 设要加入的节点是j，队首元素为i**

**\* dis(j) > dis(i) 插入队尾 否则 插入队首**

**\*/**

**struct Edge { int to, cost; };**

**vector<Edge> E[MAXN];**

**int cnt[MAXN]; //每个点的入队列次数 负数不在队列中**

**int dis[MAXN];**

**bool SPFA(int s)**

**{//s:起点**

**memset(dis, 0x3f, sizeof dis);**

**memset(cnt, 0x3f, sizeof cnt);**

**dis[s] = 0; cnt[s] = 1;**

**deque<int> que; que.push\_back(s);**

**while (!que.empty())**

**{**

**int u = que.front(); que.pop\_front();**

**cnt[u] = -cnt[u];**

**for (int i = 0; i < (int)E[u].size(); ++i)**

**{**

**int v = E[u][i].to;**

**if (dis[v] > dis[u] + E[u][i].cost)**

**{**

**dis[v] = dis[u] + E[u][i].cost;**

**if (cnt[v] <= 0)**

**{**

**cnt[v] = 1 - cnt[v];**

**if(que.empty() || dis[v] > dis[que.front()])**

**que.push\_back(v);**

**else**

**que.push\_front (v);**

**if (cnt[v] > n)**

**return false;//入队列次数超过n,存在负环回路**

**}**

**}**

**}**

**}**

**return true;**

**}**

**/\*\***

**\* 任意两点间最短路Floyd\_Warshall算法 复杂度 O(V^3)**

**\* 可以处理负边权**

**\* 顶点编号从0开始**

**\*/**

**int cost[MAXN][MAXN];**

**int dist[MAXN][MAXN];**

**void Floyd\_Warshall(int s, int n)**

**{**

**memcpy(dist, cost, sizeof cost);**

**for (int k = s; k <= n; ++k) //以第k个顶点为桥梁, k必须在最外层**

**for (int i = s; i <= n; ++i)**

**for (int j = s; j <= n; ++j)**

**{**

**int tmp = dist[i][k] + dist[k][j];**

**if (dist[i][j] > tmp) dist[i][j] = tmp;**

**}**

**}**

## 拓扑排序

**/\*\***

**\* 拓扑排序TopoSort算法 复杂度 O(V+E)**

**\* 顶点从1开始**

**\*/**

**struct Edge { int to, cost; };**

**vector<Edge> E[MAXN];**

**int cnt[MAXN]; //顶点入度**

**int top[MAXN]; //拓扑序列**

**bool TopoSort(int n)**

**{**

**int k = 0;**

**for (int i = 1; i <= n; ++i)**

**if(!cnt[i])**

**top[k++] = i;**

**for (int i = 0; i < k; ++i)**

**for (int j = 0, u = top[i]; j < (int)E[u].size(); ++j)**

**if(--cnt[ E[u][j].to ] == 0)**

**top[k++] = cnt[ E[u][j].to ];**

**if(k == n) return true; //有合法的拓扑排序**

**return false;**

**}**

## 欧拉回路

**/\*\***

**\* 欧拉图 必须连通**

**\* 无向图 顶点数度为偶数**

**\* 有向图 顶点入度=出度**

**\* 混合图**

**\* 欧拉通路**

**\* 无向图 仅有零个或两个奇数度顶点**

**\* 有向图 除两个顶点(入度=出度±1)外,其余顶点入度=出度**

**\* 混合图**

**\* \*\*\*\*\*\*\*\***

**\*/**

**//无向图 复杂度O(V^2) 下标从1开始**

**const int MAXN=1010;**

**int dis[MAXN][MAXN];**

**int cnt[MAXN]; //树的度**

**int n, m;**

**void dfs(int u, int &num)**

**{**

**for (int v = 1; v <= n; ++v)**

**if (dis[u][v])**

**{**

**dis[u][v] = dis[v][u] = 0;**

**dfs(v, ++num);**

**}**

**}**

**bool Euler\_path()**

**{**

**for (int i = 1; i <= n; ++i)**

**if (cnt[i] & 1)**

**return false;**

**int num = 0;**

**dfs(1, num);**

**if(num == m) return true;**

**return false;**

**}**

## 二分图染色

**int n,color[N];**

**vector<int> E[N];**

**bool dfs(int u, int c)**

**{//把顶点v染色成c**

**color[u] = c;**

**for (int v = 0; v < (int)E[u].size(); ++v)**

**{**

**if (color[E[u][v]] == c)**

**//相邻顶点颜色相同**

**return false;**

**if (color[E[u][v]] == 0 && !dfs(G[u][v], -c))**

**//相邻顶点没被染色，染成-c**

**return false;**

**}**

**return true;**

**}**

**bool solve()**

**{**

**for (int i = 0; i < n; ++i)**

**if (color[i] == 0) //没被染色,染成1**

**if(!dfs(i, 1))**

**return false;**

**return true;**

**}**

数**据结构**

## 树状数组

**void update(int p,int v)**

**{**

**while (p <= n)**

**a[p] += v, p += lowbit(p);**

**}**

**int query(int p)**

**{**

**int sum = 0;**

**while (p)**

**sum += a[p], p -= lowbit(p);**

**return sum;**

**}**

**//原数组a[i],差分数组c[i]=a[i]-a[i-1],数组d[i]=(i-1)\*c[i]**

**//a[1]+a[2]+...+a[n]**

**//=n\*(c[1]+c[2]+...+c[n]) - (0\*c[1]+1\*c[2]+...+(n-1)\*c[n])**

**//=n\*sigma(c,n) - sigma(d,n)**

**int n;**

**int c[MAXN]; //差分数组 a[p]=sigma(c,p)**

**int d[MAXN];**

**void add(int \*r, int p, int v)**

**{**

**for(; p <= n; p += lowbit(p)) r[p] += v;**

**}**

**int sigma(int \*r, int p)**

**{//前i项和**

**int sum = 0;**

**for(; p; p -= lowbit(p)) sum += r[p];**

**return sum;**

**}**

**void solve()**

**{**

**//初始化**

**cin>>num[i]**

**add(c, i, num[i] - num[i-1]);**

**add(d, i, (i - 1) \* (num[i] - num[i-1]));**

**//区间修改 a[x]~a[y] += v**

**cin>>x>>y>>v;**

**add(c, x, v), add(c, y + 1, -v);**

**add(d, x, v \* (x - 1)), add(d, y + 1, -v \* y);**

**//单点查询**

**cin>>p;**

**sigma(c, p);**

**//区间查询 a[x]+……+a[y]**

**cin>>x>>y;**

**sum1 = (x - 1) \* sigma(c, x - 1) - sigma(d, x - 1);**

**sum2 = y \* sigma(c, y) - sigma(d, y);**

**}**

STL模板库

**String**

|  |  |
| --- | --- |
| **c\_str** | **返回字符数组指针,用strcpy函数接收数据** |
| **copy(char\* s,count,pos=0)** | **复制子串[pos,pos+count)到字符数组s** |
| **substr(pos=0,count=size)** | **返回子串[pos,pos+count)** |
| **erase(pos=0,count=size)** | **删除子串[pos,pos+count)** |
| **insert(index,count,ch)** | **在index插入count个字符ch.** |
| **insert(index,char\* s,count=size)** | **在index插入字符数组s的前count个字符** |
| **insert(index,string& s)** | **在index插入字符串s** |
| **append(count,ch)** | **字符串尾插入count个字符ch** |
| **append(char\* s,count=size)** | **字符串尾插入字符数组s中前count个字符** |
| **find(ch,pos=0)** | **返回字符ch的下标,没查找到返回-1** |
| **find(char\* s,pos=0,count=size)** | **返回字符数组[pos,pos+count)的开始下标** |
| **find(string& s,pos=0)** | **返回子串[pos,size)的开始下标** |

**Set <key,cmp> & Map<key,T,cmp>**

|  |  |
| --- | --- |
| **insert(value)** | **返回pair<iterator,bool>** |
| **insert(iterator,iterator)** |  |
| **erase(iterator)** |  |
| **erase(iterator,iterator)** |  |
| **erase(key)** |  |
| **count(value)** | **返回值为value的个数** |
| **find(value)** | **返回iterator** |
| **equal\_range(key)** | **返回pair<iterator, iterator>** |
| **lower\_bound(key)** | **返回iterator** |
| **upper\_bound(key)** | **返回iterator** |
|  |  |
| **insert(make\_pair(key,T))** | **返回pair<iterator,bool>** |
| **insert(iterator,iterator)** |  |
| **Map\_value[key]=T** |  |

## 算法库

|  |  |
| --- | --- |
| **sort**  **upper\_bound** |  |
| **ceil** |  |
| **lower\_bound（array+i,array+n,number）-array** | **返回>=number第一个数的位置** |
| **upper\_bound(array+i,array+n,number)-array** | **返回>number第一个数的位置,不存在则返回n** |
| **lower\_bound（array+i,array+n,number）-array-1** | **返回<number第最大的位置** |
|  |  |
|  |  |
|  |  |

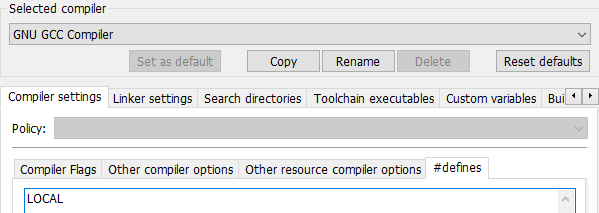
**圆周上有n个点，两两相连之后能把圆面分成多少部分？**

**(n^4 – 6n^3 + 23n^2 – 18n +24)/24;**

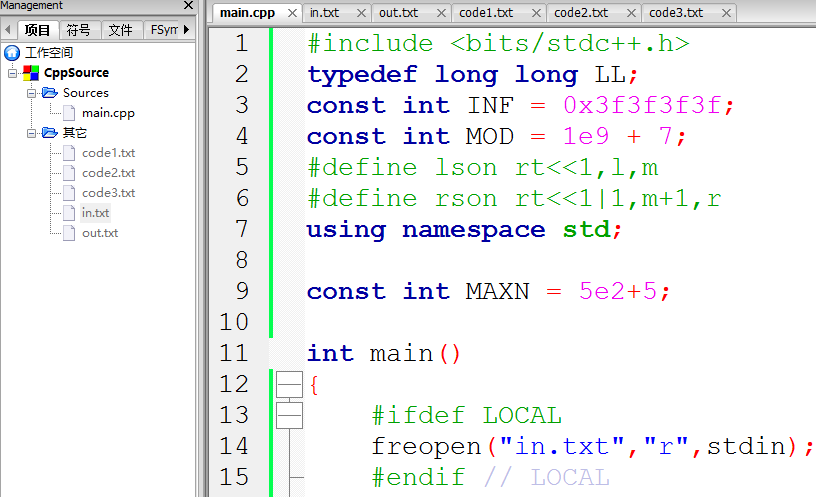
**Σ(n^2) = n(n+1)(2n+1)/6;**

# Are you get it?

**Tips 1**

****

**Codebolcks 编译器加个宏定义LOCAL，然后再添加几个文件**

**可以不删除这三行直接提交代码。**

**#ifdef LOCAL**

**freopen("in.txt","r",stdin);**

**#endif**

**Tips 2**

**ios\_base::sync\_with\_stdio(false);**

**cin.tie(NULL),cout.tie(NULL);**

**加速C++输入输出速度，不能同时使用scanf printf和cin cout，具体自行百度。**

**Tips 3**

**const int INF=0x3f3f3f3f;**

**定义无穷大值不适合定义为0x7fffffff，因为这个数再加一个数就变负数了，不符合无穷大加一个数还是无穷大，而0x3f3f3f3f和0x7fffffff是一个数量级（10^9）的，二倍也不超int数据范围。还可以使用memset(a, 0x3f, sizeof a)对数组初始化。**

**Tips 4**

**集合的正数表示**

**集合{0,1,2,……,n-1} 表示成 Σ2^i**

|  |  |
| --- | --- |
| **空集** | **0** |
| **只含第i个元素** | **1<<i** |
| **全集** | **(1<<n)-1** |
| **判断第i个元素是否属于集合S** | **if (S&1<<i)** |
| **向集合加入第i个元素** | **S|1<<i** |
| **从集合去除第i个元素** | **S&~(1<<i)** |
| **集合S与T的并集** | **S|T** |
| **集合S与T的交集** | **S&T** |
| **枚举子集** | **S0 = S0-1&S** |

**Tips 5**

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

**包含所有c++头文件，很多OJ都支持，POJ不支持。**

**Tips 6**

**解决爆栈**

**#pragma comment(linker, "/STACK:1024000000,1024000000")**

如果代码里有 递归函数 频繁调用， 用 C++ 提交代码， 很可能就会 出现

 Runtime Error(ACCESS\_VIOLATION) 但是用G++提交，如果数据量很多的话，又会出现Time Limit Exceeded

**Tips 7**

[**https://oeis.org/**](https://oeis.org/) **数学规律**

**Notice 1**

**数据误差。包括整数溢出（变负数）和浮点误差（比如1不等于1.0）。**