Standard Code Library

ZJGSU Ginray

目录

[一、基本算法： 7](#_Toc448357156)

[1 、线段树 7](#_Toc448357157)

[2、并查集 16](#_Toc448357158)

[3、快速幂、矩阵快速幂以及快速乘法 20](#_Toc448357159)

[4、块状数组 24](#_Toc448357160)

[5、树状数组 27](#_Toc448357161)

[6、二叉堆排序 30](#_Toc448357162)

[7、归并排序 32](#_Toc448357163)

[二、字符串处理 33](#_Toc448357164)

[1、KMP算法 33](#_Toc448357165)

[2、O(n)最长公共子序列 35](#_Toc448357166)

[3、O(n)回文子串（Manacher）算法 37](#_Toc448357167)

[4、后缀数组 39](#_Toc448357168)

[5、sunday算法 字符串匹配 比KMP快 42](#_Toc448357169)

[6、LIS 最长上升子序列 （nlogn） 45](#_Toc448357170)

[7、ST算法 （RMQ问题） 48](#_Toc448357171)

[8、二维RMQ 50](#_Toc448357172)

[9、AC自动机 52](#_Toc448357173)

[10、树链剖分 56](#_Toc448357174)

[11、字符串循环同构的最小表示法 60](#_Toc448357175)

[三、STL使用 62](#_Toc448357176)

[1.vector 62](#_Toc448357177)

[2.list 63](#_Toc448357178)

[3、priority queue 64](#_Toc448357179)

[4.set 65](#_Toc448357180)

[5.map 66](#_Toc448357181)

[6.总结 67](#_Toc448357182)

[四、图论 68](#_Toc448357183)

[1、邻接表的简易实现 68](#_Toc448357184)

[2、SPFA算法模板 （邻接矩阵）（求单源最短路径） 69](#_Toc448357185)

[3、SPFA(邻接表实现) 71](#_Toc448357186)

[4、迪杰斯特拉算法 76](#_Toc448357187)

[5、最小生成树模板 prime 和kruskal 79](#_Toc448357188)

[prime算法： 79](#_Toc448357189)

[kruskal算法： 80](#_Toc448357190)

[6、求树的直径 83](#_Toc448357191)

[7、DINIC 最大流 模板 85](#_Toc448357192)

[8、ISAP最大流模板 88](#_Toc448357193)

[9、LCA 模板 91](#_Toc448357194)

[10、匈牙利算法 96](#_Toc448357195)

[11、KM算法模板——带权二分图的最大匹配 98](#_Toc448357196)

[12、带花树开花算法 101](#_Toc448357197)

[13、最小比例生成树 105](#_Toc448357198)

[14、左偏树模板 107](#_Toc448357199)

[15、伸展树模板 112](#_Toc448357200)

[16、求图的割点 115](#_Toc448357201)

[五、数论 118](#_Toc448357202)

[1、欧几里得和扩展欧几里得 118](#_Toc448357203)

[2、欧拉函数 120](#_Toc448357204)

[3、欧拉判别式a^((p-1)2)=(ap)mod p 122](#_Toc448357205)

[4、求解模方程a^x=b(mod n) 124](#_Toc448357206)

[5、中国剩余定理 127](#_Toc448357207)

[6、判断素数 130](#_Toc448357208)

[7、Pollard\_rho大整数分解 模板 132](#_Toc448357209)

[8、 组合公式 136](#_Toc448357210)

[1)、C(m,n) 136](#_Toc448357211)

[C(m,n)=C(m-1,n)+C(m-1,n-1) 136](#_Toc448357212)

[2）、求和公式,k = 1..n 140](#_Toc448357213)

[六、计算几何 142](#_Toc448357214)

[0、注意 142](#_Toc448357215)

[1、凸包 143](#_Toc448357216)

[2、求凸包的直径 147](#_Toc448357217)

[3、三角形 151](#_Toc448357218)

[4、四边形 152](#_Toc448357219)

[5、正n边形: 152](#_Toc448357220)

[6、圆: 152](#_Toc448357221)

[7、棱柱 152](#_Toc448357222)

[8、棱锥 152](#_Toc448357223)

[9、棱台 153](#_Toc448357224)

[10、圆柱: 153](#_Toc448357225)

[11、圆锥: 153](#_Toc448357226)

[12、圆台: 153](#_Toc448357227)

[13、球: 153](#_Toc448357228)

[14、球台: 153](#_Toc448357229)

[15、球扇形 153](#_Toc448357230)

[16、多边形 154](#_Toc448357231)

[1)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 154](#_Toc448357232)

[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 154](#_Toc448357233)

[3)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 154](#_Toc448357234)

[4)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0 155](#_Toc448357235)

[5)判点在任意多边形内,顶点按顺时针或逆时针给出 155](#_Toc448357236)

[6)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1 155](#_Toc448357237)

[7)多边形重心 157](#_Toc448357238)

[17、浮点函数 158](#_Toc448357239)

[1) 浮点几何函数库 158](#_Toc448357240)

[2) 计算cross product (P1-P0)x(P2-P0) 158](#_Toc448357241)

[3) 计算dot product (P1-P0).(P2-P0) 158](#_Toc448357242)

[4) 两点距离 158](#_Toc448357243)

[5) 判三点共线 158](#_Toc448357244)

[6) 判点是否在线段上,包括端点 159](#_Toc448357245)

[7) 判点是否在线段上,不包括端点 159](#_Toc448357246)

[8) 判两点在线段同侧,点在线段上返回0 159](#_Toc448357247)

[9) 判两点在线段异侧,点在线段上返回0 160](#_Toc448357248)

[10) 判两直线平行 160](#_Toc448357249)

[11) 判两直线垂直 160](#_Toc448357250)

[12) 判两线段相交,包括端点和部分重合 160](#_Toc448357251)

[13) 判两线段相交,不包括端点和部分重合 161](#_Toc448357252)

[14) 计算两直线交点,注意事先判断直线是否平行! 161](#_Toc448357253)

[15) 点到直线上的最近点 161](#_Toc448357254)

[16) 点到直线距离 162](#_Toc448357255)

[17) 点到线段上的最近点 162](#_Toc448357256)

[18) 点到线段距离 162](#_Toc448357257)

[19) 矢量V以P为顶点逆时针旋转angle并放大scale倍 163](#_Toc448357258)

[18、面积 164](#_Toc448357259)

[1) 计算cross product (P1-P0)x(P2-P0) 164](#_Toc448357260)

[2) 三角形面积,输入三顶点 164](#_Toc448357261)

[3) 计算三角形面积,输入三边长 164](#_Toc448357262)

[4) 计算多边形面积,顶点按顺时针或逆时针给出 164](#_Toc448357263)

[19、三角形详细 165](#_Toc448357264)

[1) 外心 165](#_Toc448357265)

[2) 内心 165](#_Toc448357266)

[3) 垂心 166](#_Toc448357267)

[4) 重心 166](#_Toc448357268)

[5) 到三角形三顶点距离之和最小的点 166](#_Toc448357269)

[20、圆详细 168](#_Toc448357270)

[1) 判直线和圆相交,包括相切 168](#_Toc448357271)

[2) 判线段和圆相交,包括端点和相切 168](#_Toc448357272)

[3) 判圆和圆相交,包括相切 169](#_Toc448357273)

[4) 计算圆上到点p最近点,如p与圆心重合,返回p本身 169](#_Toc448357274)

[5) 计算直线与圆的交点,保证直线与圆有交点 169](#_Toc448357275)

[6) 计算圆与圆的交点,保证圆与圆有交点,圆心不重合 170](#_Toc448357276)

1. 基本算法：

1 、线段树

//单点替换、单点增减、区间求和、区间最值

#include <cstdio>

#include <algorithm>

using namespace std;

#define lson l , m , rt << 1

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

const int maxn = 10000005; //!!!!!!!!!!!注意最大值

int MAX[maxn<<2];

int MIN[maxn<<2];

int SUM[maxn<<2];

int max(int a,int b){if(a>b)return a;else return b;}

int min(int a,int b){if(a<b)return a;else return b;}

void PushUP(int rt)

{

MAX[rt] = max(MAX[rt<<1] , MAX[rt<<1|1]);

MIN[rt] = min(MIN[rt<<1] , MIN[rt<<1|1]);

SUM[rt] = SUM[rt<<1] + SUM[rt<<1|1];

}

void build(int l,int r,int rt) {

if (l == r)

{

scanf("%d",&MAX[rt]);

MIN[rt] = MAX[rt];

SUM[rt] = MAX[rt];

//printf("mi = %d\n",MIN[rt]);

// printf("ma = %d\n",MAX[rt]);

return ;

}

int m = (l + r) >> 1;

build(lson);

build(rson);

PushUP(rt);

}

void update(int p,int tihuan,int l,int r,int rt)

{

if (l == r) {

MAX[rt] = tihuan;

MIN[rt] = tihuan;

SUM[rt] = tihuan;

return ;

}

int m = (l + r) >> 1;

if (p <= m) update(p , tihuan ,lson);

else update(p , tihuan , rson);

PushUP(rt);

}

void update1(int p,int add,int l,int r,int rt)

{

if (l == r) {

MAX[rt] = SUM[rt]+add;

MIN[rt] = SUM[rt]+add;

SUM[rt] = SUM[rt] + add;

return ;

}

int m = (l + r) >> 1;

if (p <= m) update1(p , add ,lson);

else update1(p , add , rson);

PushUP(rt);

}

int query(int L,int R,int l,int r,int rt)

{

if (L <= l && r <= R)

{

return MAX[rt];

}

int m = (l + r) >> 1;

int ret = -1;

if (L <= m) ret = max(ret , query(L , R , lson));

if (R > m) ret = max(ret , query(L , R , rson));

return ret;

}

int query1(int L,int R,int l,int r,int rt)

{

if (L <= l && r <= R)

{

return MIN[rt];

}

int m = (l + r) >> 1;

int ret = 99999999999; //!!!!!!!!!!!!!!!!!!!!!!注意最大值

if (L <= m) ret = min(ret , query1(L , R , lson));

if (R > m) ret = min(ret , query1(L , R , rson));

return ret;

}

int queryhe(int L,int R,int l,int r,int rt)

{

if (L <= l && r <= R)

{

return SUM[rt];

}

int m = (l + r) >> 1;

int ret = 0;

if (L <= m) ret += queryhe(L , R , lson);

if (R > m) ret += queryhe(L , R , rson);

return ret;

}

int main()

{

int n , m;

while (~scanf("%d%d",&n,&m))

{

build(1 , n , 1);

//此处应输入n个数

while (m --) {

char op[2];

int a , b;

scanf("%s%d%d",op,&a,&b);

if (op[0] == 'Q') //区间求最大

{

/\* for(int i = 1;i<=10;i++)

printf("%d ",MAX[i]);

puts("");

\*/

printf("%d\n",query(a , b , 1 , n , 1));

}

else if(op[0]=='U') //单点替换

update(a , b , 1 , n , 1);

else if(op[0]=='M')//区间求最小

{

/\*for(int i = 1;i<=10;i++)

printf("%d ",MIN[i]);

puts("");\*/

printf("%d\n",query1(a , b , 1 , n , 1));

}

else if(op[0]=='H')//区间求和

{

printf("%d\n",queryhe(a , b , 1 , n , 1));

}

else if(op[0]=='S')//单点增加

{

//scanf("%d%d",&a,&b);

update1(a , b , 1 , n , 1);

}

else if(op[0]=='E')//单点减少

{

//scanf("%d%d",&a,&b);

update1(a , -b , 1 , n , 1);

}

}

}

return 0;

}

//区间替换

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <algorithm>

#define max(a,b) (a>b)?a:b

#define min(a,b) (a>b)?b:a

#define lson l , m , rt << 1

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

#define LL long long

const int maxn = 10000005; //注意最大值

using namespace std;

int lazy[maxn<<2];

int sum[maxn<<2];

void PushUp(int rt)//由左孩子、右孩子向上更新父节点

{

sum[rt] = sum[rt<<1] + sum[rt<<1|1];

}

void PushDown(int rt,int m) //向下更新

{

if (lazy[rt]) //懒惰标记

{

lazy[rt<<1] = lazy[rt<<1|1] = lazy[rt];

sum[rt<<1] = (m - (m >> 1)) \* lazy[rt];

sum[rt<<1|1] = ((m >> 1)) \* lazy[rt];

lazy[rt] = 0;

}

}

void build(int l,int r,int rt)//建树

{

lazy[rt] = 0;

if (l== r)

{

scanf("%d",&sum[rt]);

return ;

}

int m = (l + r) >> 1;

build(lson);

build(rson);

PushUp(rt);

}

void update(int L,int R,int c,int l,int r,int rt)//更新

{

//if(L>l||R>r) return;

if (L <= l && r <= R)

{

lazy[rt] = c;

sum[rt] = c \* (r - l + 1);

//printf("%d %d %d %d %d\n", rt, sum[rt], c, l, r);

return ;

}

PushDown(rt , r - l + 1);

int m = (l + r) >> 1;

if (L <= m) update(L , R , c , lson);

if (R > m) update(L , R , c , rson);

PushUp(rt);

}

LL query(int L,int R,int l,int r,int rt)

{

if (L <= l && r <= R)

{

//printf("%d\n", sum[rt]);

return sum[rt];

}

PushDown(rt , r - l + 1);

int m = (l + r) >> 1;

LL ret = 0;

if (L <= m) ret += query(L , R , lson);

if (m < R) ret += query(L , R , rson);

return ret;

}

int main()

{

int n , m;

char str[5];

while(scanf("%d%d",&n,&m))

{

build(1 , n , 1);

while (m--)

{

scanf("%s",str);

int a , b , c;

if(str[0]=='T')

{

scanf("%d%d%d",&a,&b,&c);

update(a , b , c , 1 , n , 1);

}

else if(str[0]=='Q')

{

scanf("%d%d",&a,&b);

cout<<query(a,b,1,n,1)<<endl;

}

}

}

return 0;

}

//区间增减

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <algorithm>

#define max(a,b) (a>b)?a:b

#define min(a,b) (a>b)?b:a

#define lson l , m , rt << 1

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

#define LL \_\_int64

const int maxn = 10000005; //注意最大值!!!!!!!

using namespace std;

LL lazy[maxn<<2];

LL sum[maxn<<2];

void putup(int rt)

{

sum[rt] = sum[rt<<1] + sum[rt<<1|1];

}

void putdown(int rt,int m)

{

if (lazy[rt])

{

lazy[rt<<1] += lazy[rt];

lazy[rt<<1|1] += lazy[rt];

sum[rt<<1] += lazy[rt] \* (m - (m >> 1));

sum[rt<<1|1] += lazy[rt] \* (m >> 1);

lazy[rt] = 0;

}

}

void build(int l,int r,int rt) {

lazy[rt] = 0;

if (l == r)

{

scanf("%I64d",&sum[rt]);

return ;

}

int m = (l + r) >> 1;

build(lson);

build(rson);

putup(rt);

}

void update(int L,int R,int c,int l,int r,int rt)

{

if (L <= l && r <= R)

{

lazy[rt] += c;

sum[rt] += (LL)c \* (r - l + 1);

return ;

}

putdown(rt , r - l + 1);

int m = (l + r) >> 1;

if (L <= m) update(L , R , c , lson);

if (m < R) update(L , R , c , rson);

putup(rt);

}

LL query(int L,int R,int l,int r,int rt)

{

if (L <= l && r <= R)

{

return sum[rt];

}

putdown(rt , r - l + 1);

int m = (l + r) >> 1;

LL ret = 0;

if (L <= m) ret += query(L , R , lson);

if (m < R) ret += query(L , R , rson);

return ret;

}

int main()

{

int n , m;int a , b , c;

char str[5];

scanf("%d%d",&n,&m);

build(1 , n , 1);

while (m--)

{

scanf("%s",str);

if (str[0] == 'Q')

{

scanf("%d%d",&a,&b);

printf("%I64d\n",query(a , b , 1 , n , 1));

}

else if(str[0]=='C')

{

scanf("%d%d%d",&a,&b,&c);

update(a , b , c , 1 , n , 1);

}

}

return 0;

}

2、并查集

#include<stdio.h>

#include<stdlib.h>

int pre[4005];

const int MAX = 2000;

int flag;

int find (int x)

{

int r =x;

if(pre[r]!=r)

{

pre[r]=find(pre[r]);

}

return pre[r];

}

void join (int x,int y)

{

int fx=find(x);

int fy=find(y);

if(fx==fy)

{

flag=1;

return;

}

// fy = find(y);

// if(fx!=fy)

pre[fx]=pre[fy];

return ;

}

void init(int n)

{

int i;

for(i=1;i<=2000+n;++i)pre[i] = i;

flag = 0;

}

int main ()

{

int t;

scanf("%d",&t);

while(t--)

{

int i,j;

int m,n,a,b;

int fa,fb;

scanf("%d%d",&m,&n);

init(m);

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

{

scanf("%d%d",&a,&b);

if(flag==0)

join(a,b);

}

if(flag)

{

printf("有相同祖先\n");

}

else

printf("没有相同祖先\n");

}

return 0;

}

大数相乘模板（大数模板代码量太多，手敲不现实）

#include<stdio.h>

#include<string.h>

#define MAX\_LEN 200

unsigned an1[ MAX\_LEN + 5 ];

unsigned an2[ MAX\_LEN + 5 ];

unsigned aResult[ MAX\_LEN \* 2 + 10 ];

char szLine1[ MAX\_LEN + 5 ];

char szLine2[ MAX\_LEN + 5 ];

int main()

{

gets( szLine1 );

gets( szLine2 );

int i, j;

memset( an1, 0, sizeof( an1 ) );

memset( an2, 0, sizeof( an2 ) );

memset( aResult, 0, sizeof( aResult ) );

int nLen1 = strlen( szLine1 ); //szLine1处理

j = 0;

for( i = nLen1 - 1; i >= 0; i-- )

an1[ j++ ] = szLine1[ i ] - '0';

int nLen2 = strlen( szLine2 ); //szLine2处理

j = 0;

for( i = nLen2 - 1; i >= 0; i-- )

an2[ j++ ] = szLine2[ i ] - '0';

for( i = 0; i < nLen2; i++ )

{

for( j = 0; j < nLen1; j++ )

aResult[ i + j ] += an2[ i ] \* an1[ j ];// an1[ j ] \* an2[ i ] 结果累加到第 i + j 位 ；；；；先不要进位处理

}

for( i = 0; i < MAX\_LEN \* 2; i++ )

{

if( aResult[ i ] >= 10 ) //进位处理

{

aResult[ i + 1 ] += aResult[ i ] / 10;

aResult[ i ] %= 10;

}

}

int flag = 0;

for( i = MAX\_LEN \* 2; i >= 0; i-- )

if( flag )

printf( "%d", aResult[ i ] );//输出 同大整数加法

else if( aResult[ i ] )

{

printf( "%d", aResult[ i ] );

flag = 1;

}

if( flag == 0 ) //都为 0 的情况

printf( "0" );

return 0;

}

3、快速幂、矩阵快速幂以及快速乘法

/\*矩阵快速幂真是非常nice

附上计算快速幂的方法

// m^n % k

int quickpow(int m,int n,int k)

{

int b = 1;

while (n > 0)

{

if (n & 1)

b = (b\*m)%k;

n = n >> 1 ;

m = (m\*m)%k;

}

return b;

}

\*/

//快速乘法，基本思路和快速幂是一样的

#include <iostream>

#include <cstdio>

#include <cmath>

#include <cstring>

using namespace std;

long long q\_mul( long long a, long long b, long long mod ) //快速计算 (a\*b) % mod

{

long long ans = 0; // 初始化

while(b) //根据b的每一位看加不加当前a

{

if(b & 1) //如果当前位为1

{

b--;

ans =(ans+ a)%mod; //ans+=a

}

b /= 2; //b向前移位

a = (a + a) % mod; //更新a

}

return ans;

}

int main( )

{

long long a, b, n;

while(cin >> a >> b >> n)

{

cout << "a\*b%n = " << q\_mul( a, b, n ) << endl;

}

return 0;

}

#include <stdio.h>

#include <string.h>

const int NN=32;

typedef struct node{

int matrix[NN][NN];

}Matrix;

Matrix a,sa,unit;

int n,m,k;

Matrix Add(Matrix a,Matrix b) //矩阵加法(%m)

{

Matrix c;

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

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

{

c.matrix[i][j]=a.matrix[i][j]+b.matrix[i][j];

c.matrix[i][j]%=m;

}

return c;

}

Matrix Mul(Matrix a,Matrix b) //矩阵乘法(%m)

{

Matrix c;

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

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

{

c.matrix[i][j]=0;

for (k=0; k<n; k++)

c.matrix[i][j]+=a.matrix[i][k]\*b.matrix[k][j];

c.matrix[i][j]%=m;

}

return c;

}

Matrix Cal(int exp) //矩阵快速幂

{

Matrix p,q;

p=a;

q=unit;

while (exp!=1)

{

if (exp&1)

{

exp--;

q=Mul(p,q);

}

else

{

exp>>=1;

p=Mul(p,p);

}

}

return Mul(p,q);

}

int main()

{

scanf("%d%d%d",&n,&k,&m);

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

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

{

scanf("%d",&a.matrix[i][j]);

a.matrix[i][j]%=m;

unit.matrix[i][j]=(i==j);

}

sa=Cal(k);

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

{

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

printf("%d ",sa.matrix[i][j]);

printf("\n");

}

return 0;

}

4、块状数组

#include<stdio.h>

#include <string.h>

/\*用块状链表来实现类似文本编辑器的作用，来处理大量数据\*/

const int N=2000, LEN=2000;

struct Block\_List {

struct Node {

char buff[LEN];

int next, Size;

void init() {

memset(buff, 0, sizeof buff);

next=-1, Size=0;

}

} List[N];

int head, tot;

void init(char S[]) {

head=tot=0;

List[tot++].init();

for(int i=0, cur=head; S[i]; cur=List[cur].next) {

for(int j=0; j<LEN && S[i]; j++, i++) {

List[cur].buff[j]=S[i];

List[cur].Size++;

}

if(S[i]) {

List[tot].init();

List[cur].next=tot++;

}

}

for(int cur=head; cur!=-1; cur=List[cur].next)

if(List[cur].Size==LEN) Split(cur);

}

void Split(int id) {

List[tot].init();

for(int i=LEN/2; i<LEN; i++) {

List[tot].buff[i-LEN/2]=List[id].buff[i];

List[tot].Size++;

List[id].buff[i]=0;

List[id].Size--;

}

List[tot].next=List[id].next;

List[id].next=tot++;

}

void Insert(int pos, char val) {

int cur=head;

while(pos>List[cur].Size && List[cur].next!=-1) {

pos-=List[cur].Size;

cur=List[cur].next;

}

if(pos>=List[cur].Size) List[cur].buff[List[cur].Size]=val;

else {

for(int i=List[cur].Size; i>pos; i--) List[cur].buff[i]=List[cur].buff[i-1];

List[cur].buff[pos]=val;

}

List[cur].Size++;

if(List[cur].Size==LEN) Split(cur);

}

char Find(int pos) {

int cur=head;

while(pos>List[cur].Size) pos-=List[cur].Size, cur=List[cur].next;

return List[cur].buff[pos-1];

}

};

Block\_List hehe;

char buff[1000006], S[2];

int n;

int main() {

// freopen("in", "r", stdin);

scanf("%s", buff);

hehe.init(buff);

scanf("%d", &n);

for(int i=0, pos; i<n; i++) {

scanf("%s", buff);

if(buff[0]=='I') {

scanf("%s%d", S, &pos);

hehe.Insert(pos-1, S[0]);

}

else {

scanf("%d", &pos);

printf("%c\n", hehe.Find(pos));

}

}

return 0;

}

5、树状数组

/\*求出任意一个子矩阵内的所有元素之和，即sum(x2, y2) - sum(x1-1, y2) - sum(x2, y1-1) + sum(x1-1, y1-1) \*/

#include<stdio.h>

#include<string.h>

/\*一维\*/

#define size 50005

int c[size],n;

int Lowbit(int k)

{

return (k&-k);

}

void update(int pos,int num)

{

while(pos<=size)//注意这里

{

c[pos]+=num;

pos+=Lowbit(pos);

}

}

int sum(int pos)

{

int s=0;

while(pos>0)

{

s+=c[pos];

pos-=Lowbit(pos);

}

return s;

}

int main()

{

int i,j,s;

int T;

int ccase=1;

scanf("%d",&T);

while(T--)

{

int n;

scanf("%d",&n);

int x;

memset(c,0,sizeof(c));

for(x=1;x<=n;x++)

{

int fad;

scanf("%d",&fad); //注意怎么添加原始值

update(x,fad);

}

printf("Case %d:\n",ccase++);

while(1)

{

char f[10];

scanf("%s",f);

int a,b;

if(f[0]=='A') //增加值

{

scanf("%d%d",&a,&b);

update(a,b);

}

else if(f[0]=='S') //减少值

{

scanf("%d%d",&a,&b);

update(a,-b);

}

else if(f[0]=='Q') //查询区间的和

{

scanf("%d%d",&a,&b);

int ans=sum(b)-sum(a-1);

printf("%d\n",ans);

}

else if(f[0]=='E') //退出

break;

}

}

return 0;

}

/\*二维\*/

const int N=10000;

int c[111][111];

int lowbit(int x)

{

return x&(-x);

}

void modify(int x,int y,int delta )

{

int i, j;

for(i=x;i<=N;i+=lowbit(i))

{

for(j=y; j<=N; j+=lowbit(j))

{

c[i][j] += delta;

}

}

}

int sum( int x, int y )

{

int res=0,i,j;

for(i=x;i>0;i-=lowbit(i))

{

for(j=y; j>0; j-=lowbit(j))

{

res += c[i][j];

}

}

return res;

}

6、二叉堆排序

/\*堆排序(大顶堆) 2011.9.14\*/

#include<stdio.h>

#include<string.h>

void swap(int &x,int &y)

{

int a;

a=x;

x=y;

y=a;

}

void HeapAdjust(int \*a,int i,int size) //调整堆

{

int lchild=2\*i; //i的左孩子节点序号

int rchild=2\*i+1; //i的右孩子节点序号

int max=i; //临时变量

if(i<=size/2) //如果i是叶节点就不用进行调整

{

if(lchild<=size&&a[lchild]>a[max])

{

max=lchild;

}

if(rchild<=size&&a[rchild]>a[max])

{

max=rchild;

}

if(max!=i)

{

swap(a[i],a[max]);

HeapAdjust(a,max,size); //避免调整之后以max为父节点的子树不是堆

}

}

}

void BuildHeap(int \*a,int size) //建立堆

{

int i;

for(i=size/2;i>=1;i--) //非叶节点最大序号值为size/2

{

HeapAdjust(a,i,size);

}

}

void HeapSort(int \*a,int size) //堆排序

{

int i;

BuildHeap(a,size);

for(i=size;i>=1;i--)

{

//cout<<a[1]<<" ";

swap(a[1],a[i]); //交换堆顶和最后一个元素，即每次将剩余元素中的最大者放到最后面

//BuildHeap(a,i-1); //将余下元素重新建立为大顶堆

HeapAdjust(a,1,i-1); //重新调整堆顶节点成为大顶堆

}

}

int main(int argc, char \*argv[])

{

//int a[]={0,16,20,3,11,17,8};

int a[100];

int size;

while(scanf("%d",&size)!=EOF&&size>0)

{

int i;

for(i=1;i<=size;i++)

scanf("%d",&a[i]);

HeapSort(a,size);

for(i=1;i<=size;i++)

printf("%d,",a[i]);

printf("\n");

}

return 0;

}

7、归并排序

#include <iostream>

using namespace std;

void mergearray(int a[], int first, int mid, int last, int temp[])

{

int i = first, j = mid + 1;

int m = mid, n = last;

int k = 0;

while (i <= m && j <= n)

{

temp[k++] = a[i] > a[j] ? a[j++] : a[i++];

}

while (i <= m)

temp[k++] = a[i++];

while (j <= n)

temp[k++] = a[j++];

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

a[first + i] = temp[i];

}

void mergeSort(int data[], int start, int end ,int temp[]){

if(start < end){

mergeSort(data,start ,(start+end)/2,temp);

mergeSort(data,(start+end)/2+1 ,end,temp);

mergearray(data,start,(start+end)/2,end,temp);

}

}

int main(){

int input[9]={0,1,1,5,6,71,3,4,9};

int output[9];

mergeSort( input ,0,8,output);

// mergeSort(input,0,8);

for(int i=0;i<9 ;i++){

cout << output[i] <<" ";

}

// cout<< BinarySearch(input,0,8,8);

return 0; }

二、字符串处理

1、KMP算法

#include <cstdio>

#include <cstring>

int next[1005000];

char s[1005000]; char t[1005000];

inline void calnext(char s[],int next[]) {

int i,j;

int len = strlen(s);

next[0]=-1;

j=-1;

for(i=1;i<len;i++) {

while(j>=0&&s[i]!=s[j+1])

j=next[j];

if(s[j+1]==s[i])//上一个循环可能因为 j==-1 而不做，此时不能知道s[i]与s[j+1]的关系。故需要此条件

j++;

next[i]=j;

}

}

int KMP(char t[],char s[]) {

int ans=0;

int lent=strlen(t);

int lens=strlen(s);

if(lent<lens) return 0;

int i,j;

j=-1;

for(i=0;i<lent;i++) {

while(j>=0&&s[j+1]!=t[i])

j=next[j];

if(s[j+1]==t[i])

j++;

if(j==lens-1) {

ans++;

j=next[j];

}

}

return ans;

}

int main() {

int n;

scanf("%d",&n);

while(n--) {

scanf("%s%s",s,t); //s代表短串,t代表长串

int slen=strlen(s);

int tlen=strlen(t);

calnext(s,next);

//slen%(slen-1-next[slen-1])==0

//slen代表串长，(slen-1-next[slen-1])代表最小重复子串长度

if(slen%(slen-1-next[slen-1])==0)

printf("%d\n",slen/(slen-1-next[slen-1]));//求最多重复子串个数

else

printf("1\n");

printf("%d\n",KMP(t,s));//求s在t中的个数

}

return 0;

}

2、O(n)最长公共子序列

#include <cstdlib>

#include <cstring>

#include <cstdio>

#include <algorithm>

using namespace std;

const int N = 200010;

int seq [N],sa[N];

int n;

char str[N];

int wa[N], wb[N], ws[N], wv[N];

int rank[N], height[N];

bool cmp(int r[], int a, int b, int l) {

return r[a] == r[b] && r[a+l] == r[b+l];

}

void da(int r[], int sa[], int n, int m) {

int i, j, p, \*x = wa, \*y = wb;

for (i = 0; i < m; ++i) ws[i] = 0;

for (i = 0; i < n; ++i) ws[x[i]=r[i]]++;

for (i = 1; i < m; ++i) ws[i] += ws[i-1];

for (i = n-1; i >= 0; --i) sa[--ws[x[i]]] = i;

for (j = 1, p = 1; p < n; j \*= 2, m = p) {

for (p = 0, i = n - j; i < n; ++i) y[p++] = i;

for (i = 0; i < n; ++i) if (sa[i] >= j) y[p++] = sa[i] - j;

for (i = 0; i < n; ++i) wv[i] = x[y[i]];

for (i = 0; i < m; ++i) ws[i] = 0;

for (i = 0; i < n; ++i) ws[wv[i]]++;

for (i = 1; i < m; ++i) ws[i] += ws[i-1];

for (i = n-1; i >= 0; --i) sa[--ws[wv[i]]] = y[i];

for (swap(x, y), p = 1, x[sa[0]] = 0, i = 1; i < n; ++i)

x[sa[i]] = cmp(y, sa[i-1], sa[i], j) ? p-1 : p++;

}

}

void calheight(int r[], int sa[], int n) {

int i, j, k = 0;

for (i = 1; i <= n; ++i) rank[sa[i]] = i;

for (i = 0; i < n; height[rank[i++]] = k)

for (k?k--:0, j = sa[rank[i]-1]; r[i+k] == r[j+k]; k++);

}

int main ()

{

char s1[N],s2[N];

while(scanf("%s%s",s1,s2)!=EOF)

{

int len1=strlen(s1);

int len2=strlen(s2);

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

seq[i]=s1[i]-'a'+2;

seq[len1]=1;

for(int i=0;i<len2;i++)

seq[len1+i+1]=s2[i]-'a'+2;

seq[len1+len2+1]=0;

int len=len1+len2+1;

da(seq,sa,len+1,30);

calheight(seq,sa,len);

int maxn=0;

for(int i=2;i<=len;i++)

{

if(height[i]>maxn)

{

if(0<=sa[i-1]&&sa[i-1]<len1&&sa[i]>len1)

maxn=height[i];

if(0<=sa[i]&&sa[i]<len1&&sa[i-1]>len1)

maxn=height[i];

}

}

printf("%d\n",maxn);

}

return 0;

}

3、O(n)回文子串（Manacher）算法

#include <iostream>

#include <string>

#include <cstring>

using namespace std;

void findBMstr(string& str)

{

int \*p = new int[str.size() + 1];

memset(p, 0, sizeof(p));

int mx = 0, id = 0;

for(int i = 1; i <= str.size(); i++)

{

if(mx > i)

{

p[i] = (p[2\*id - i] < (mx - i) ? p[2\*id - i] : (mx - i));

}

else

{

p[i] = 1;

}

while(str[i - p[i]] == str[i + p[i]])

p[i]++;

if(i + p[i] > mx)

{

mx = i + p[i];

id = i;

}

}

int max = 0, ii;

for(int i = 1; i < str.size(); i++)

{

if(p[i] > max)

{

ii = i;

max = p[i];

}

}

max--;

int start = ii - max ;

int end = ii + max;

for(int i = start; i <= end; i++)

{

if(str[i] != '#')

{

cout << str[i];

}

}

cout << endl;

delete p;

}

int main()

{

string str = "12212321";

string str0;

str0 += "$#";

for(int i = 0; i < str.size(); i++)

{

str0 += str[i];

str0 += "#";

}

cout << str0 << endl;

findBMstr(str0);

return 0;

}

4、后缀数组

#include <cstdlib>

#include <cstring>

#include <cstdio>

#include <algorithm>

using namespace std;

const int N = 50005;

int seq[N], sa[N], rank[N], height[N];

int wa[N], wb[N], ws[N], wv[N];

char str[N];

int cmp(int \*r,int a,int b,int l)

{

return r[a]==r[b]&&r[a+l]==r[b+l];

}

void da(int \*r,int \*sa,int n,int m)

{

int i,j,p,\*x=wa,\*y=wb;

// 下面四行是对第一个字母的一个基数排序：基数排序其实就是记录前面有多少个位置被占据了

for(i=0;i<m;i++) ws[i]=0; // 将统计字符数量的数组清空

for(i=0;i<n;i++) ws[x[i]=r[i]]++; // 统计各种字符的个数

for(i=1;i<m;i++) ws[i]+=ws[i-1]; // 进行一个累加，因为前面的小字符集对后面字符的排位有位置贡献

for(i=n-1;i>=0;i--) sa[--ws[x[i]]]=i; // 根据位置来排序，sa[x] = i，表示i位置排在第x位

// wa[x[i]]就是字符集0-x[i]共有多少字符占据了位置，减去自己的一个位置剩下的就是自己的排名了，排名从0开始

// 排名过程中主要的过程是对于处于相同字符的字符的排序，因为改变wa[x[i]]值得只会是本身，小于该字符的贡献值

// 是不变的，对于第一个字符相同的依据是位置关系，在后面将看到通过第二个关键字来确定相同字符的先后关系

// 这以后的排序都是通过两个关键字来确定一个串的位置，也即倍增思想

// 通过将一个串分解成两部分，而这两部分的位置关系我们都已经计算出来

for(j=1,p=1;p<n;j\*=2,m=p)

{

for(p=0,i=n-j;i<n;i++) y[p++]=i; // 枚举的串是用于与i位置的串进行合并，由于i较大，因为匹配的串为空串

// 由于枚举的是长度为j的串，那么i位置开始的串将凑不出这个长度的串，因此第二关键字应该最小，这其中位置靠前的较小

for(i=0;i<n;i++) if(sa[i]>=j) y[p++]=sa[i]-j; // sa[i]-j开头的串作为第二关键字与编号为sa[i]的串匹配，sa[i]<j的串不用作为第二关键字来匹配

for(i=0;i<n;i++) wv[i]=x[y[i]]; // 取出这些位置的第一关键字

for(i=0;i<m;i++) ws[i]=0;

for(i=0;i<n;i++) ws[wv[i]]++;

for(i=1;i<m;i++) ws[i]+=ws[i-1];

for(i=n-1;i>=0;i--) sa[--ws[wv[i]]]=y[i]; // 按照第二关键字进行第一关键字的基数排序

for(swap(x,y),p=1,x[sa[0]]=0,i=1;i<n;i++) // 对排好序的sa数组进行一次字符集缩小、常数优化

x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;

}

return;

}

void calheight(int \*r,int \*sa,int n) // 这里的n是原串的本来长度，即不包括新增的0

{

int i,j,k=0;

for(i=1;i<=n;i++) rank[sa[i]]=i; // 有后缀数组得到名次数组，排名第0的后缀一定是添加的0

for(i=0;i<n;height[rank[i++]]=k) // 以 i 开始的后缀总能够从以 i-1 开始的后缀中继承 k-1 匹配项出来

for(k?k--:0,j=sa[rank[i]-1];r[i+k]==r[j+k];k++); // 进行一个暴力的匹配，但是整个算法的时间复杂度还是O(n)的

return;

}

int main()

{

int T;

scanf("%d", &T);

while (T--)

{

scanf("%s", str);

int len = strlen(str);

for (int i = 0; i < len; ++i)

{

seq[i] = str[i];

}

seq[len] = 0;

da(seq, sa, len+1, 128); //后两个是长度+1和字符的种类数目

calheight(seq, sa, len);

//写具体的实现方法

}

return 0;

}

5、sunday算法 字符串匹配 比KMP快

#include <iostream>

#include <cstring>

using namespace std;

int strStr(char \*haystack, char \*needle)

{

int len1 = strlen(haystack);

int len2 = strlen(needle);

int charStep[256];

for (int i = 0; i < 256; ++i)

charStep[i] = -1;

for (int i = 0; i < len2; ++i)

charStep[(int)needle[i]] = i;

for (int i = 0; i <= len1 - len2;)

{

int j = 0;

while (j < len2) {

if (haystack[i] == needle[j]) {

++i;

++j;

} else {

char\* p = haystack + i + len2 - j;

if (charStep[(int)\*p] == -1) {

i = p - haystack + 1;

} else {

i = p - charStep[(int)\*p] - haystack;

}

break;

}

}

if (j == len2) {

return i - len2;

}

}

return -1;

}

int main()

{

char src[]="abcdezzz";

char des[]="abcdezzz";

cout<<strStr(src,des)<<endl; //-1是匹配不成功 0是匹配成功

return 0;

}

Tire字典树

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

typedef struct node

{

int cnt;

struct node \*next[26];

}\*tree,t;

tree root;

void insert(char \*str)

{

tree p=root,newnode;

for(;\*str!='\0';str++)

{

if(p->next[\*str-'a']!=NULL)

{

p=p->next[\*str-'a'];

p->cnt++;

}

else

{

newnode=(tree)malloc(sizeof(t));

for(int i=0;i<26;i++)

newnode->next[i]=NULL;

newnode->cnt=1;

p->next[\*str-'a']=newnode;

p=newnode;

}

}

}

int find(char \*str)

{

tree p=root;

for(;\*str!='\0';str++)

{

if(p->next[\*str-'a']!=NULL)

p=p->next[\*str-'a'];

else return 0;

}

return p->cnt;

}

int main()

{

char str[16];

root=(tree)malloc(sizeof(t));

for(int i=0;i<26;i++)

root->next[i]=NULL;

root->cnt=0;//根节点初始化当然为0，不包含任何字母

while(gets(str))

{

if(strcmp(str,"")==0)

break;

insert(str);

}

while(gets(str))

{

//if(strcmp(str,"")==0)

//break;

printf("%d\n",find(str));

}

return 0;

}

## 6、LIS 最长上升子序列 （nlogn）

#define MAXN 100005

#include <iostream>

#include<stdio.h>

#include<algorithm>

using namespace std;

int a[MAXN];

int n;

int solve(int b[],int l)

{

int f[MAXN];//f[i]表示子序列长度为i+1的序列中，末尾元素最小的元素的值

int k=0;

f[k++]=b[0];

for(int i=1;i<l;i++)

{

if(b[i]>=f[k-1]) f[k++]=b[i];//上升子序列可以相等的情况 若不相等则改成=号

else

{

int pos=upper\_bound(f,f+k,a[i])-f;

f[pos]=a[i];

}

}

return k;

}

int main()

{

int l=0;

int num;

scanf("%d",&num);

while(num--)

{

int n;

scanf("%d",&n);

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

scanf("%d",&a[i]);

printf("Case #%d:\n%d\n",++l,n-(solve(a,n)));

}

}

//

/\*\*

最长递增子序列O(nlogn)算法：

状态转移方程：f[i] = max{f[i],f[j]+1},1<=j<i,a[j]<a[i].

分析：加入x<y,f[x]>=f[y],则x相对于y更有潜力。

首先根据f[]值分类，记录满足f[t]=k的最小的值a[t],记d[k]=min{a[t]},f[t]=k.

1.发现d[k]在计算过程中单调不上升

2.d[1]<d[2]<...<d[k] (反证) 1 2 3 8 4 7

解法：

1. 设当前最长递增子序列为len,考虑元素a[i];

2. 若d[len]<a[i],则len++，并将d[len]=a[i];

否则,在d[0-len]中二分查找,找到第一个比它小的元素d[k],并d[k+1]=a[i].()

\*/

#include <iostream>

#include <cstdio>

#include <cstring>

using namespace std;

const int N = 41000;

int a[N]; //a[i] 原始数据

int d[N]; //d[i] 长度为i的递增子序列的最小值

int BinSearch(int key, int\* d, int low, int high)

{

while(low<=high)

{

int mid = (low+high)>>1;

if(key>d[mid] && key<=d[mid+1])

return mid;

else if(key>d[mid])

low = mid+1;

else

high = mid-1;

}

return 0;

}

int LIS(int\* a, int n, int\* d)

{

int i,j;

d[1] = a[1];

int len = 1; //递增子序列长度

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

{

if(d[len]<a[i])

j = ++len;

else

j = BinSearch(a[i],d,1,len) + 1;

d[j] = a[i];

}

return len;

}

int main()

{

int t;

int p;

scanf("%d",&t);

while(t--)

{

scanf("%d",&p);

for(int i = 1; i <= p; i++)

scanf("%d",&a[i]);

printf("%d\n",LIS(a,p,d));

}

return 0;

}

7、ST算法 （RMQ问题）

#include <string.h>

#include <string.h>

#include <stdio.h>

#include<math.h>

#define MAXN 100005

#define MAXM 100005

#define max(a,b) (a>b)?a:b

#define min(a,b) (a<b)?a:b

int mi[MAXN][30], mx[MAXN][30], w[MAXN];

int n, q;

void rmqinit()

{

for(int i = 1; i <= n; i++) mi[i][0] = mx[i][0] = w[i];

int m = (int)(log(n \* 1.0) / log(2.0));

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

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

{

mx[j][i] = mx[j][i - 1];

if(j + (1 << (i - 1)) <= n) mx[j][i] = max(mx[j][i], mx[j + (1 << (i - 1))][i - 1]);

mi[j][i] = mi[j][i - 1];

if(j + (1 << (i - 1)) <= n) mi[j][i] = min(mi[j][i], mi[j + (1 << (i - 1))][i - 1]);

}

}

int rmqmin(int l,int r)

{

int m = (int)(log((r - l + 1) \* 1.0) / log(2.0));

return min(mi[l][m] , mi[r - (1 << m) + 1][m]);

}

int rmqmax(int l,int r)

{

int m = (int)(log((r - l + 1) \* 1.0) / log(2.0));

return max(mx[l][m] , mx[r - (1 << m) + 1][m]);

}

int main()

{

scanf("%d%d", &n, &q);

for(int i = 1; i <= n; i++) scanf("%d", &w[i]);

rmqinit();

int l, r;

while(q--)

{

scanf("%d%d", &l, &r);

printf("%d\n", rmqmax(l, r));

}

return 0;

}

8、二维RMQ

#include <stdio.h>

#include <algorithm>

#include <iostream>

#include <string.h>

using namespace std;

int val[310][310];

int dp[310][310][9][9];//最大值

int mm[310];//二进制位数减一，使用前初始化

void initRMQ(int n,int m)

{

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

for(int j = 1;j <= m;j++)

dp[i][j][0][0] = val[i][j];

for(int ii = 0; ii <= mm[n]; ii++)

for(int jj = 0; jj <= mm[m]; jj++)

if(ii+jj)

for(int i = 1; i + (1<<ii) - 1 <= n;i++)

for(int j = 1; j + (1<<jj) - 1 <= m;j++)

{

if(ii)dp[i][j][ii][jj] = max(dp[i][j][ii-1][jj],dp[i+(1<<(ii-1))][j][ii-1][jj]);

else dp[i][j][ii][jj] = max(dp[i][j][ii][jj-1],dp[i][j+(1<<(jj-1))][ii][jj-1]);

}

}

//查询矩形内的最大值(x1<=x2,y1<=y2)

int rmq(int x1,int y1,int x2,int y2)

{

int k1 = mm[x2-x1+1];

int k2 = mm[y2-y1+1];

x2 = x2 - (1<<k1) + 1;

y2 = y2 - (1<<k2) + 1;

return max(max(dp[x1][y1][k1][k2],dp[x1][y2][k1][k2]),max(dp[x2][y1][k1][k2],dp[x2][y2][k1][k2]));

}

int main()

{

//在外面对mm数组进行初始化

mm[0] = -1;

for(int i = 1;i <= 305;i++)

mm[i] = ((i&(i-1))==0)?mm[i-1]+1:mm[i-1];

int n,m;

int Q;

int r1,c1,r2,c2;

while(scanf("%d%d",&n,&m) == 2)

{

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

for(int j = 1;j <= m;j++)

scanf("%d",&val[i][j]);

initRMQ(n,m);

scanf("%d",&Q);

while(Q--)

{

scanf("%d%d%d%d",&r1,&c1,&r2,&c2);

int tmp = rmq(r1,c1,r2,c2);

printf("%d ",tmp);

}

}

return 0;

}

9、AC自动机

#include<stdio.h>

#include<string.h>

#include<malloc.h>

#include<queue>

#define KIND 26

//为可能的种类，假如是小写英文就是26 int pos=str[k]-'a';

//不可见字符就是 128 int pos=str[k]-' ';

//注意更改~!!

using namespace std;

char str[1000000+100];

struct node

{

int count;

struct node \*next[KIND];

struct node \*fail;

void init()

{

int i;

for(i=0;i<KIND;i++)

next[i]=NULL; //接下去要跳转的地方

count=0; //这里保存的是单词的个数 也可以保存单词的序号什么的

fail=NULL;

}

}\*root;

void insert()

{

int len,k;

node \*p=root;

len=strlen(str);

for(k=0;k<len;k++)

{

int pos=str[k]-'a'; //注意更改

if(p->next[pos]==NULL)

{

p->next[pos]=new node;

p->next[pos]->init();

p=p->next[pos];

}

else

p=p->next[pos];

}

p->count++;

}

void getfail()

{

int i;

node \*p=root,\*son,\*temp;

queue<struct node \*>que;

que.push(p);

while(!que.empty())

{

temp=que.front();

que.pop();

for(i=0;i<KIND;i++)

{

son=temp->next[i];

if(son!=NULL)

{

if(temp==root) {son->fail=root;}

else

{

p=temp->fail;

while(p)

{

if(p->next[i])

{

son->fail=p->next[i];

break;

}

p=p->fail;

}

if(!p) son->fail=root;

}

que.push(son);

}

}

}

}

void query()

{

int len,i,cnt=0;

//int flag[5500]; 记录单词是不是被访问过了

len=strlen(str);

node \*p,\*temp;

p=root;

for(i=0;i<len;i++)

{

int pos=str[i]-'a'; //注意更改

while(!p->next[pos]&&p!=root) p=p->fail;

p=p->next[pos];

if(!p) p=root;

temp=p;

while(temp!=root)

{

if(temp->count>=0) //这时候flag数组在这里判断一下就行了

{

cnt+=temp->count;

temp->count=-1; // 这里把原来的值清空了 假如一个AC自动机要多次被访问，可以用一个flag数组来保存是否被访问过

//假如可以计算重复的子串 就删除 temp->count=-1;这句。

}

temp=temp->fail;

}

}

printf("%d\n",cnt); //长串中有多少个AC自动机中的子串

}

int main()

{

int cas,n;

scanf("%d",&cas);

while(cas--)

{

root=new node;

root->init();

root->fail=NULL;

scanf("%d",&n);

int i;

getchar();

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

{

gets(str);

insert();

}

getfail();

gets(str); //假如有多个子串，记得添加flag数组

query();

}

return 0;

}

10、树链剖分

#include <cstdio>

#include <cstring>

#include <vector>

#include <algorithm>

using namespace std;

#define Del(a,b) memset(a,b,sizeof(a))

const int N = 10005;

int dep[N],siz[N],fa[N],id[N],son[N],val[N],top[N]; //top 最近的重链父节点

int num;

vector<int> v[N];

struct tree

{

int x,y,val;

void read(){

scanf("%d%d%d",&x,&y,&val);

}

};

tree e[N];

void dfs1(int u, int f, int d) {

dep[u] = d;

siz[u] = 1;

son[u] = 0;

fa[u] = f;

for (int i = 0; i < v[u].size(); i++) {

int ff = v[u][i];

if (ff == f) continue;

dfs1(ff, u, d + 1);

siz[u] += siz[ff];

if (siz[son[u]] < siz[ff])

son[u] = ff;

}

}

void dfs2(int u, int tp) {

top[u] = tp;

id[u] = ++num;

if (son[u]) dfs2(son[u], tp);

for (int i = 0; i < v[u].size(); i++) {

int ff = v[u][i];

if (ff == fa[u] || ff == son[u]) continue;

dfs2(ff, ff);

}

}

#define lson(x) ((x<<1))

#define rson(x) ((x<<1)+1)

struct Tree

{

int l,r,val;

};

Tree tree[4\*N];

void pushup(int x) {

tree[x].val = max(tree[lson(x)].val, tree[rson(x)].val);

}

void build(int l,int r,int v)

{

tree[v].l=l;

tree[v].r=r;

if(l==r)

{

tree[v].val = val[l];

return ;

}

int mid=(l+r)>>1;

build(l,mid,v\*2);

build(mid+1,r,v\*2+1);

pushup(v);

}

void update(int o,int v,int val) //log(n)

{

if(tree[o].l==tree[o].r)

{

tree[o].val = val;

return ;

}

int mid = (tree[o].l+tree[o].r)/2;

if(v<=mid)

update(o\*2,v,val);

else

update(o\*2+1,v,val);

pushup(o);

}

int query(int x,int l, int r)

{

if (tree[x].l >= l && tree[x].r <= r) {

return tree[x].val;

}

int mid = (tree[x].l + tree[x].r) / 2;

int ans = 0;

if (l <= mid) ans = max(ans, query(lson(x),l,r));

if (r > mid) ans = max(ans, query(rson(x),l,r));

return ans;

}

int Yougth(int u, int v) {

int tp1 = top[u], tp2 = top[v];

int ans = 0;

while (tp1 != tp2) {

//printf("YES\n");

if (dep[tp1] < dep[tp2]) {

swap(tp1, tp2);

swap(u, v);

}

ans = max(query(1,id[tp1], id[u]), ans);

u = fa[tp1];

tp1 = top[u];

}

if (u == v) return ans;

if (dep[u] > dep[v]) swap(u, v);

ans = max(query(1,id[son[u]], id[v]), ans);

return ans;

}

void Clear(int n)

{

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

v[i].clear();

}

int main()

{

//freopen("Input.txt","r",stdin);

int T;

scanf("%d",&T);

while(T--)

{

int n;

scanf("%d",&n);

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

{

e[i].read();

v[e[i].x].push\_back(e[i].y);

v[e[i].y].push\_back(e[i].x);

}

num = 0;

dfs1(1,0,1);

dfs2(1,1);

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

if (dep[e[i].x] < dep[e[i].y]) swap(e[i].x, e[i].y);

val[id[e[i].x]] = e[i].val;

}

build(1,num,1);

char s[200];

while(~scanf("%s",&s) && s[0]!='D')

{

int x,y;

scanf("%d%d",&x,&y);

if(s[0]=='Q')

printf("%d\n",Yougth(x,y));

if (s[0] == 'C')

update(1,id[e[x].x],y);

}

Clear(n);

}

return 0;

}

11、字符串循环同构的最小表示法

#include <iostream>

#include <cstdio>

#include <cstring>

using namespace std;

char data[10000];

int getminsub(char \*a)

{

int i=0,j=1,len=strlen(a),k=0; //取两个同构的字符串一个从下标0开始，一个从下标1开始

while(i<len&&j<len&&k<len) //这里并没有将字符串复制一份添加到后面

{

if(k==len) break; //说明找到了a的最小表示

if(i==j) j++;

int ni=i+k,nj = j+k;

if(ni>=len) ni-=len; //就是回到字符串的开始去

if(nj>=len) nj-=len;

if(data[ni]>data[nj])

{

i+=k+1;

k=0;

}

else if(data[ni]<data[nj])

{

j+=k+1;

k=0;

}

else k++;

}

return i; //返回从第i个字符开始时a的最小表示

}

//求最大的时候把大于号小于号换一下就行了

int getmaxsub(char \*a)

{

int i=0,j=1,len=strlen(a),k=0; //取两个同构的字符串一个从下标0开始，一个从下标1开始

while(i<len&&j<len&&k<len) //这里并没有将字符串复制一份添加到后面

{

if(k==len) break; //说明找到了a的最小表示

if(i==j) j++;

int ni=i+k,nj = j+k;

if(ni>=len) ni-=len; //就是回到字符串的开始去

if(nj>=len) nj-=len;

if(data[ni]<data[nj])

{

i+=k+1;

k=0;

}

else if(data[ni]>data[nj])

{

j+=k+1;

k=0;

}

else k++;

}

return i; //返回从第i个字符开始时a的最小表示

}

int main()

{

scanf("%s",data);

printf("%d\n",getminsub(data)+1);

printf("%d\n",getmaxsub(data)+1);

return 0;

}

三、STL使用

1.vector

 vector 底部实现为数组，支持快速的在底部添加数据 ，但是它的修改是n级别的

基本操作

(1)头文件#include<vector>.

(2)创建vector对象，vector<int> vec;

(3)尾部插入数字：vec.push\_back(a);

(4)使用下标访问元素，cout<<vec[0]<<endl;记住下标是从0开始的。

(5)使用迭代器访问元素.

vector<int>::iterator it;

for(it=vec.begin();it!=vec.end();it++)

cout<<\*it<<endl;

(6)插入元素：    vec.insert(vec.begin()+i,a);在第i+1个元素前面插入a;

(7)删除元素：    vec.erase(vec.begin()+2);删除第3个元素

vec.erase(vec.begin()+i,vec.end()+j);删除区间[i,j-1];区间从0开始

(8)向量大小:vec.size();

(9)清空:vec.clear();

2.list

Lists将元素按顺序储存在链表中. 与 向量(vectors)相比, 它允许快速的插入和删除，但是随机访问却比较慢.

基本操作

assign() 给list赋值   
back() 返回最后一个元素   
begin() 返回指向第一个元素的迭代器   
clear() 删除所有元素   
empty() 如果list是空的则返回true   
end() 返回末尾的迭代器   
erase() 删除一个元素   
front() 返回第一个元素   
get\_allocator() 返回list的配置器   
insert() 插入一个元素到list中   
max\_size() 返回list能容纳的最大元素数量   
merge() 合并两个list   
pop\_back() 删除最后一个元素   
pop\_front() 删除第一个元素   
push\_back() 在list的末尾添加一个元素   
push\_front() 在list的头部添加一个元素   
rbegin() 返回指向第一个元素的逆向迭代器   
remove() 从list删除元素   
remove\_if() 按指定条件删除元素   
rend() 指向list末尾的逆向迭代器   
resize() 改变list的大小   
reverse() 把list的元素倒转   
size() 返回list中的元素个数   
sort() 给list排序   
splice() 合并两个list   
swap() 交换两个list   
unique() 删除list中重复的元素

3、priority queue

优先队列：顾名思义，首先它是一个队列，但是它强调了“优先”二字，所以，已经不能算是一般意义上的队列了，它的“优先”意指取队首元素时，有一定的选择性，即根据元素的属性选择某一项值最优的出队~

百度百科上这样描述的：

　　优先级队列 是不同于先进先出队列的另一种队列。每次从队列中取出的是具有最高优先权的元素

　　优先队列的类定义

　　优先队列是0个或多个元素的集合,每个元素都有一个优先权或值,对优先队列执行的操作有1) 查找;2) 插入一个新元素;3) 删除.在最小优先队列(min priorityq u e u e)中,查找操作用来搜索优先权最小的元素,删除操作用来删除该元素;对于最大优先队列(max priority queue),查找操作用来搜索优先权最大的元素,删除操作用来删除该元素.优先权队列中的元素可以有相同的优先权,查找与删除操作可根据任意优先权进行.

优先队列，其构造及具体实现我们可以先不用深究，我们现在只需要了解其特性，及在做题中的用法，相信，看过之后你会收获不少。

使用优先队列，首先要包函STL头文件"queue"，

基本操作

#include<stdio.h>

#include<functional>

#include<queue>

#include<vector>

using namespace std;

//定义结构，使用运算符重载,自定义优先级1

struct cmp1{

bool operator ()(int &a,int &b){

return a>b;//最小值优先

}

};

struct cmp2{

bool operator ()(int &a,int &b){

return a<b;//最大值优先

}

};

4.set

0.set集合容器：实现了红黑树的平衡二叉检索树的数据结构，插入元素时，它会自动调整二叉树的排列，把元素放到适当的位置，以保证每个子树根节点键值大于左子树所有节点的键值，小于右子树所有节点的键值；另外，还得保证根节点左子树的高度与右子树高度相等。  
平衡二叉检索树使用中序遍历算法，检索效率高于vector、deque和list等容器，另外使用中序遍历可将键值按照从小到大遍历出来。  
构造set集合主要目的是为了快速检索，不可直接去修改键值。

1.基本操作

1.元素插入：insert()  
2.中序遍历：类似vector遍历（用迭代器）  
3.反向遍历：利用反向迭代器reverse\_iterator。  
    例：  
    set<int> s;  
    ......  
    set<int>::reverse\_iterator rit;  
    for(rit=s.rbegin();rit!=s.rend();rit++)  
4.元素删除：与插入一样，可以高效的删除，并自动调整使红黑树平衡。  
            set<int> s;  
            s.erase(2);        //删除键值为2的元素  
            s.clear();  
5.元素检索：find()，若找到，返回该键值迭代器的位置，否则，返回最后一个元素后面一个位置。  
            set<int> s;  
            set<int>::iterator it;  
            it=s.find(5);    //查找键值为5的元素  
            if(it!=s.end())    //找到  
                cout<<\*it<<endl;  
            else            //未找到  
                cout<<"未找到";

## 5.map

0.map是一类关联式容器，它是模板类。关联的本质在于元素的值与某个特定的键相关联，而并非通过元素在数组中的位置类获取。它的特点是增加和删除节点对迭代器的影响很小，除了操作节点，对其他的节点都没有什么影响。对于迭代器来说，不可以修改键值，只能修改其对应的实值。

1.基本操作

map的基本操作函数：  
      C++ Maps是一种关联式容器，包含“关键字/值”对  
      begin()          返回指向map头部的迭代器  
      clear(）         删除所有元素  
      count()          返回指定元素出现的次数  
      empty()          如果map为空则返回true  
      end()            返回指向map末尾的迭代器  
      equal\_range()    返回特殊条目的迭代器对  
      erase()          删除一个元素  
      find()           查找一个元素  
      get\_allocator()  返回map的配置器  
      insert()         插入元素  
      key\_comp()       返回比较元素key的函数  
      lower\_bound()    返回键值>=给定元素的第一个位置  
      max\_size()       返回可以容纳的最大元素个数  
      rbegin()         返回一个指向map尾部的逆向迭代器  
      rend()           返回一个指向map头部的逆向迭代器  
      size()           返回map中元素的个数  
      swap()            交换两个map  
      upper\_bound()     返回键值>给定元素的第一个位置  
      value\_comp()      返回比较元素value的函数

6.总结

C++ STL 的实现：

1.vector  底层数据结构为数组 ，支持快速随机访问

2.list    底层数据结构为双向链表，支持快速增删

3.deque   底层数据结构为一个中央控制器和多个缓冲区，详细见STL源码剖析P146，支持首尾（中间不能）快速增删，也支持随机访问

4.stack   底层一般用23实现，封闭头部即可，不用vector的原因应该是容量大小有限制，扩容耗时

5.queue   底层一般用23实现，封闭头部即可，不用vector的原因应该是容量大小有限制，扩容耗时

6.45是适配器,而不叫容器，因为是对容器的再封装

7.priority\_queue 的底层数据结构一般为vector为底层容器，堆heap为处理规则来管理底层容器实现

8.set       底层数据结构为红黑树，有序，不重复

9.multiset  底层数据结构为红黑树，有序，可重复

10.map      ﻿﻿﻿﻿底层数据结构为红黑树，有序，不重复

11.multimap 底层数据结构为红黑树，有序，可重复

12.hash\_set ﻿﻿﻿﻿底层数据结构为hash表，无序，不重复

13.hash\_multiset 底层数据结构为hash表，无序，可重复

14.hash\_map      ﻿﻿﻿﻿底层数据结构为hash表，无序，不重复

15.hash\_multimap 底层数据结构为hash表，无序，可重复

四、图论

1、邻接表的简易实现

//在有的oj上开next数组会报编译错误，因为有oj的库有重名 ，注意！！！！

int n,m,i;

//u、v和w的数组大小要根据实际情况来设置，要比m的最大值要大1

int u[6],v[6],w[6];

//first和next的数组大小要根据实际情况来设置，要比n的最大值要大1

int first[5],next[5];

scanf("%d %d",&n,&m);

//初始化first数组下标1~n的值为-1，表示1~n顶点暂时都没有边

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

first[i]=-1;

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

{

scanf("%d %d %d",&u[i],&v[i],&w[i]);//读入每一条边

//下面两句是关键啦

next[i]=first[u[i]];

first[u[i]]=i;

}

遍历每个顶点的所有边的代码如下。

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

{

k=first[i];

while(k!=-1)

{

printf("%d %d %d\n",u[k],v[k],w[k]);

k=next[k];

}

}

2、SPFA算法模板 （邻接矩阵）（求单源最短路径）

#include<iostream>

#include<cstdio>

#include<cstring>

#include<cstdlib>

#include<queue>

using namespace std;

#define Max 9999999;

int a[105][105];

int flag[105];//保存是否被访问过

int dis[105];//与节点0的最短路径的大小

int n;

queue<int> q;

void innt ()

{

memset(a,0,sizeof(a));

memset(flag,0,sizeof(flag));

dis[0]=0;

int i;

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

dis [i]=Max;

return ;

}

void SPFA (int s)//开始的节点

{

q.push(s);

while(!q.empty())

{

int x=q.front();

flag[x]=1;

int i;

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

{

if(a[i][x]!=0&&(dis[x]+a[i][x])<dis[i])

{

dis[i]=dis[x]+a[i][x];

if(!flag[i])

{

q.push(i);

flag[i]=1;

}

}

}

q.pop();

flag[x]=0;

}

return ;

}

int main ()

{

while(scanf("%d",&n)!= EOF)

{

int i,j;

int f;

innt();//用矩阵来储存,

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

{

for(j=0;j<i;j++)

{

scanf("%d",&f);

a[i][j]=f;

a[j][i]=f;

}

}

SPFA (0);

int minn=-1\*Max;

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

{

if(dis[i]>minn)

minn=dis[i];

}

printf("%d\n",minn);

}

return 0;

}

3、SPFA(邻接表实现)

#include <iostream>

#include<stdlib.h>

#include<string.h>

#include<stdio.h>

#include <queue>

using namespace std;

const long MAXN=10000;

const long lmax=0x7FFFFFFF;

typedef struct

{

long v;

long next;

long cost;

}Edge;

Edge e[MAXN];

long p[MAXN];

long Dis[MAXN];

bool vist[MAXN];

queue<long> q;

long m,n;//点,边

void init()

{

long i;

long eid=0;

memset(vist,0,sizeof(vist));

memset(p,-1,sizeof(p));

fill(Dis,Dis+MAXN,lmax);

while (!q.empty())

{

q.pop();

}

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

{

long from,to,cost;

scanf("%ld %ld %ld",&from,&to,&cost);

e[eid].next=p[from];

e[eid].v=to;

e[eid].cost=cost;

p[from]=eid++;

//以下适用于无向图

swap(from,to);

e[eid].next=p[from];

e[eid].v=to;

e[eid].cost=cost;

p[from]=eid++;

}

}

void print(long End)

{

//若为lmax 则不可达

printf("%ld\n",Dis[End]);

}

void SPF()

{

init();

long Start,End;

scanf("%ld %ld",&Start,&End);

Dis[Start]=0;

vist[Start]=true;

q.push(Start);

while (!q.empty())

{

long t=q.front();

q.pop();

vist[t]=false;

long j;

for (j=p[t];j!=-1;j=e[j].next)

{

long w=e[j].cost;

if (w+Dis[t]<Dis[e[j].v])

{

Dis[e[j].v]=w+Dis[t];

if (!vist[e[j].v])

{

vist[e[j].v]=true;

q.push(e[j].v);

}

}

}

}

print(End);

}

int main()

{

while (scanf("%ld %ld",&m,&n)!=EOF)

{

SPF();

}

return 0;

}

4、迪杰斯特拉算法

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int rode[1002][1002];

int flag[1002];//是否被访问过

int temp[1002];//到这个点的距离

void djs(int x,int y,int z) //要到y点 ，有x个数,有z条边

{

int i,j,now;

int min;

for(i=0;i<1002;i++)

{

flag[i]=0;

temp[i]=99999999;

}

for (i=1;i<=x;i++)

temp[i]=rode[y][i];

temp[y]=0;

for(i=1;i<=x;i++)

{

min =99999999;

for(j=1;j<=x;j++)

{

if(!flag[j]&&temp[j]<min)

{

min=temp[j];

now=j;

}

}

flag[now]=1;

for(j=1;j<=x;j++)

if(temp[j]>temp[now]+rode[now][j])

temp[j]=temp[now]+rode[now][j];

}

}

int main ()

{

int n,m,x;

while(~scanf("%d%d%d",&n,&m,&x))

{

int a,b,t;

int i,j,wtf;

int ans1[1002],ans2[1002];

int ans=-9999;

for(i=00;i<1002;i++)

{

for(j=0;j<1002;j++)

rode[i][j]=99999999;

}

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

{

scanf("%d%d%d",&a,&b,&t);

rode[a][b]=t;

}

djs(n,x,m);

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

{

ans1[i]=temp[i];

}

for(i=0;i<1002;i++)

{

for(j=i+1;j<1002;j++)

{

wtf=rode[i][j];

rode[i][j]=rode[j][i];

rode[j][i]=wtf;

}

}

djs(n,x,m);

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

{

if(ans1[i]+temp[i]>ans)

ans=ans1[i]+temp[i];

}

printf("%d\n",ans);

}

return 0;

}

5、最小生成树模板 prime 和kruskal

prime算法：

#include<stdio.h>

#include<string.h>

#define INF 0x3fffffff

int lowcost[110];//此数组用来记录第j个节点到其余节点最少花费

int map[110][110];//用来记录第i个节点到其余n-1个节点的距离

int visit[110];//用来记录最小生成树中的节点

int n;

void prime()

{

int min,i,j,next,mincost=0;

memset(visit,0,sizeof(visit));//给最小生成树数组清零

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

{

lowcost[i]=map[1][i];//初始化lowcost数组为第1个节点到剩下所有节点的距离

}

visit[1]=1;//选择第一个点为最小生成树的起点

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

{

min=INF;

for(j=1;j<=n;j++)

{

if(!visit[j]&&min>lowcost[j])//如果第j个点不是最小生成树中的点并且其花费小于min

{

min=lowcost[j];

next=j;//记录下此时最小的位置节点

}

}

if(min==INF)

{

printf("数据不足以确定\n"); //当数据不够构成最小生成树的时候

return ;

}

mincost+=min;//将最小生成树中所有权值相加

visit[next]=1;//next点加入最小生成树

for(j=1;j<=n;j++)

{

if(!visit[j]&&lowcost[j]>map[next][j])//如果第j点不是最小生成树中的点并且此点处权值大于第next点到j点的权值

{

lowcost[j]=map[next][j]; //更新lowcost数组

}

}

}

printf("%d\n",mincost);

}

int main()

{

int road;

int j,i,x,y,c;

while(scanf("%d%d",&road,&n)&&road!=0)

{

memset(map,INF,sizeof(map));//初始化数组map为无穷大

while(road--)

{

scanf("%d%d%d",&x,&y,&c);

map[x][y]=map[y][x]=c;//城市x到y的花费==城市y到想的花费

}

prime();

}

return 0;

}

kruskal算法：

#include<stdio.h>

#include<algorithm>

using namespace std;

int set[110];

struct record

{

int beg;

int end;

int money;

}s[11000];

int find(int fa)

{

int ch=fa;

int t;

while(fa!=set[fa])

fa=set[fa];

while(ch!=fa)

{

t=set[ch];

set[ch]=fa;

ch=t;

}

return fa;

}

void mix(int x,int y)

{

int fx,fy;

fx=find(x);

fy=find(y);

if(fx!=fy)

set[fx]=fy;

}

bool cmp(record a,record b)

{

return a.money<b.money;

}

int main()

{

int city,road,n,m,j,i,sum;

while(scanf("%d",&road)&&road!=0)

{

scanf("%d",&city);

for(i=0;i<road;i++)

{

scanf("%d%d%d",&s[i].beg,&s[i].end,&s[i].money);

}

for(i=1;i<=city;i++)

set[i]=i;

sort(s,s+road,cmp);

sum=0;

for(i=0;i<road;i++)

{

if(find(s[i].beg)!=find(s[i].end))

{

mix(s[i].beg,s[i].end);

sum+=s[i].money;

}

}

j=0;

for(i=1;i<=city;i++)

{

if(set[i]==i)

j++;

if(j>1)

break;

}

if(j>1)

printf("数据不足以确定\n"); //当数据不够构成最小生成树的时候

else

printf("%d\n",sum);

}

return 0;

}

6、求树的直径

// 求树的直径 即最长的路径

#include<iostream>

#include<queue>

#define INF 999999

#define M 2002

using namespace std;

int n;

int maxx;

int map[M][M],sum[M];

bool flag[M];

int bfs(int begin){

queue<int>f;

int i,m,k,key;

maxx=0;

memset(flag,false,sizeof(flag));

f.push(begin);

while(!f.empty()){

k=f.front();

for(i=1;i<=n;i++){

if(map[k][i]!=INF&&!flag[i]){

flag[i]=true;

f.push(i);

sum[i]=sum[k]+map[k][i];

if(sum[i]>maxx) { maxx=sum[i];key=i; }

}

}

f.pop();

}

return key;

}

int main()

{

int i,j,k,dis,key,cas;

scanf("%d",&cas);

while(cas--)

{

scanf("%d",&n);

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

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

map[i][j]=map[j][i]=INF;

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

{

scanf("%d%d%d",&j,&k,&dis); //a到b的距离为dis

map[j][k]=map[k][j]=dis;

}

sum[1]=0;

key=bfs(1);

sum[key]=0;

key=bfs(key);

printf("%d\n",maxx); //最远的距离

}

7、DINIC 最大流 模板

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

#define N 5005

#define M 10005

#define INF 999999999

using namespace std;

int n,m,s,t,num,adj[N],dis[N],q[N];

struct edge

{

int v,w,pre;

}e[M];

void insert(int u,int v,int w)

{

e[num].v=v;

e[num].w=w;

e[num].pre=adj[u];

adj[u]=num++;

e[num].v=u;

e[num].w=0;

e[num].pre=adj[v];

adj[v]=num++;

}

int bfs()

{

int i,x,v,tail=0,head=0;

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

dis[s]=1;

q[tail++]=s;

while(head<tail)

{

x=q[head++];

for(i=adj[x];i!=-1;i=e[i].pre)

if(e[i].w&&dis[v=e[i].v]==0)

{

dis[v]=dis[x]+1;

if(v==t)

return 1;

q[tail++]=v;

}

}

return 0;

}

int dfs(int s,int limit)

{

if(s==t)

return limit;

int i,v,tmp,cost=0;

for(i=adj[s];i!=-1;i=e[i].pre)

if(e[i].w&&dis[s]==dis[v=e[i].v]-1)

{

tmp=dfs(v,min(limit-cost,e[i].w));

if(tmp>0)

{

e[i].w-=tmp;

e[i^1].w+=tmp;

cost+=tmp;

if(limit==cost)

break;

}

else dis[v]=-1;

}

return cost;

}

int Dinic()

{

int ans=0;

while(bfs())

ans+=dfs(s,INF);

return ans;

}

int main ()

{

while(~scanf("%d%d",&m,&n))

{

int u,v,w;

memset(adj,-1,sizeof(adj));

num=0;

s=1;// 开始的点

t=n;//结束的点

while(m--)

{

scanf("%d%d%d",&u,&v,&w);

insert(u,v,w);

}

printf("%d\n",Dinic());

}

}

8、ISAP最大流模板

#include<cstdio>

#include<cstring>

#include<queue>

#include<algorithm>

#define N 505

#define inf 999999999

using namespace std;

int n,m,s,t,dis[N],pre[N],gap[N],flow[N][N];

struct edge

{

int v,w;

edge \*next,\*rev;

edge(){next=0;}

edge(int vv,int ww,edge \*e){v=vv;w=ww;next=e;}

}\*adj[N],\*path[N],\*e;

void insert(int u,int v,int w)

{

edge \*p=new edge(v,w,adj[u]);

adj[u]=p;

edge \*q=new edge(u,0,adj[v]);

adj[v]=q;

p->rev=q;

q->rev=p;

}

void bfs()

{

memset(dis,0x7f,sizeof(dis));

memset(gap,0,sizeof(gap));

queue<int> q;

dis[t]=0;

gap[0]=1;

q.push(t);

while(q.size())

{

int x=q.front();

q.pop();

for(e=adj[x];e;e=e->next)

{

if(e->rev->w==0||dis[e->v]<t)

continue;

dis[e->v]=dis[x]+1;

++gap[dis[e->v]];

q.push(e->v);

}

}

}

int ISAP()

{

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

memset(gap,0,sizeof(gap));

//bfs();

int ans=0,u=s,d;

while(dis[s]<=t)

{

if(u==t)

{

int minflow=-1u>>1;

for(e=path[u];u!=s;e=path[u=pre[u]])

minflow=min(minflow,e->w);

for(e=path[u=t];u!=s;e=path[u=pre[u]])

{

e->w-=minflow;

e->rev->w+=minflow;

flow[pre[u]][u]+=minflow;

flow[u][pre[u]]-=minflow;

}

ans+=minflow;

}

for(e=adj[u];e;e=e->next)

if(e->w>0&&dis[u]==dis[e->v]+1)

break;

if(e)

{

pre[e->v]=u;

path[e->v]=e;

u=e->v;

}

else

{

if(--gap[dis[u]]==0)

break;

for(d=t,e=adj[u];e;e=e->next)

if(e->w>0)

d=min(d,dis[e->v]);

dis[u]=d+1;

++gap[dis[u]];

if(u!=s)

u=pre[u];

}

}

return ans;

}

int main()

{

int u,v,w;

while(~scanf("%d%d",&m,&n))

{

memset(adj,0,sizeof(adj));

while(m--)

{

scanf("%d%d%d",&u,&v,&w);

insert(u,v,w);

//insert(v,u,w);//无向边

}

s=1;

t=n;

printf("%d\n",ISAP());

}

}

9、LCA 模板

//并查集+深搜

#include<stdio.h>

#include<vector>

#include<string.h>

using namespace std;

#define Size 11111 //节点个数

vector<int> node[Size],que[Size];

int n,pare[Size],anse[Size],in[Size],rank[Size];

int vis[Size];

void init()

{

int i;

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

{

node[i].clear();

que[i].clear();

rank[i]=1;

pare[i]=i;///

}

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

memset(in,0,sizeof(in));

memset(anse,0,sizeof(anse));

}

int find(int nd)//并查集操作 不解释

{

return pare[nd]==nd?nd:pare[nd]=find(pare[nd]);

}

int Union(int nd1,int nd2)//并查集操作 不解释

{

int a=find(nd1);

int b=find(nd2);

if(a==b) return 0;

else if(rank[a]<=rank[b])

{

pare[a]=b;

rank[b]+=rank[a];

}

else

{

pare[b]=a;

rank[a]+=rank[b];

}

return 1;

}

void LCA(int root)

{

int i,sz;

anse[root]=root;//首先自成一个集合

sz=node[root].size();

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

{

LCA(node[root][i]);//递归子树

Union(root,node[root][i]);//将子树和root并到一块

anse[find(node[root][i])]=root;//修改子树的祖先也指向root

}

vis[root]=1;

sz=que[root].size();

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

{

if(vis[que[root][i]])

{

printf("LCA is %d\n",anse[find(que[root][i])]);///root和que[root][i]所表示的值的最近公共祖先

return ;

}

}

return ;

}

int main()

{

int cas,i;

scanf("%d",&cas);

while(cas--)

{

int s,e;

scanf("%d",&n);

init();

for(i=0;i<n-1;i++)

{

scanf("%d %d",&s,&e);

if(s!=e)

{

node[s].push\_back(e);

// node[e].push\_back(s);

in[e]++;

}

}

scanf("%d %d",&s,&e);

que[s].push\_back(e);

que[e].push\_back(s);

for(i=1;i<=n;i++) if(in[i]==0) break;//寻找根节点

//printf("root=%d\n",i);

LCA(i);

}

return 0;

}

//Tarjan算法 带权值

//只能离线处理 降低复杂度

#include <iostream>

#include <cstdio>

#include <cstring>

using namespace std;

#define N 10010

#define M 10010

#define Q 1000010

int head[N];

struct edge{

int u,v,w,next;

}e[2\*M];

int \_\_head[N];

struct ask{

int u,v,lca,next;

}ea[2\*Q];

int dir[N],fa[N],vis[N];

inline void add\_edge(int u ,int v ,int w ,int &k)

{

e[k].u = u; e[k].v = v; e[k].w = w;

e[k].next = head[u]; head[u] = k++;

}

inline void add\_ask(int u ,int v ,int &k)

{

ea[k].u = u; ea[k].v = v; ea[k].lca = -1;

ea[k].next = \_\_head[u]; \_\_head[u] = k++;

}

int find(int x){

return x == fa[x] ? x : fa[x] = find(fa[x]);

}

void Tarjan(int u ,int c)

{

vis[u] = c; fa[u] = u;

for(int k=head[u]; k!=-1; k=e[k].next)

if(!vis[e[k].v])

{

int v = e[k].v , w = e[k].w;

dir[v] = dir[u] + w;

Tarjan(v,c);

fa[v] = u;

}

for(int k=\_\_head[u]; k!=-1; k=ea[k].next)

if(vis[ea[k].v] == c)

ea[k].lca = ea[k^1].lca = find(ea[k].v);

}

int main()

{

int n,m,q,k;

while(scanf("%d%d%d",&n,&m,&q)!=EOF)

{

memset(head,-1,sizeof(head));

memset(\_\_head,-1,sizeof(\_\_head));

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

k = 0;

for(int i=0; i<m; i++)

{

int u,v,w;

scanf("%d%d%d",&u,&v,&w);

add\_edge(u,v,w,k);

add\_edge(v,u,w,k);

}

k = 0;

for(int i=0; i<q; i++)

{

int u,v;

scanf("%d%d",&u,&v);

add\_ask(u,v,k);

add\_ask(v,u,k);

}

k = 0;

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

if(!vis[i])

{

dir[i] = 0;

Tarjan(i,++k);

}

for(int i=0; i<q; i++)

{

int s = i\*2 , u = ea[s].u , v = ea[s].v , lca = ea[s].lca;

if(lca == -1) printf("Not connected\n");

else printf("%d\n",dir[u] + dir[v] - 2\*dir[lca]);

}

}

return 0;

}

10、匈牙利算法

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

const int maxn = 301\*101;

int first[maxn],next[maxn],v[maxn];//头结点，下一点，和边的终点

int vis[maxn]; //记录是否访问过

int link [maxn];//记录与之匹配的点

int p ,n ;

void innt()

{

memset(first,-1,sizeof(first));

memset(link,0,sizeof(link));

return ;

}

int find ( int k)

{

int i;

for(i=first[k];i!=-1;i=next[i])

{//寻找与k连接的点

if(!vis[v[i]])

{//如果没访问过，则访问并标记00

vis[v[i]]=1;

if(link[v[i]]==0||find(link[v[i]]))

{

link[v[i]]=k;

return 1;//与之匹配

}

}

}

return 0;

}

int main ()

{

int t;

scanf("%d",&t);

while(t--)

{

int i,j;

int e=0;

innt();

scanf("%d%d",&p,&n);//p为边的量 n为点的量

for(i=0;i<p;i++) //建立类邻接表的数组

{

int a;

scanf("%d",&a);//a点和之后的都有边的关系

for(j=0;j<a;j++,e++)

{

scanf("%d",&v[e]);

next[e]=first[i+1];

first[i+1]=e;

}

}

int ans=0;

for(i=0;i<p;i++)

{

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

if(!find(i+1))

{

ans=1;

break;

}

}

//判断能否找出二分图的最大匹配

if(ans)

printf("NO\n");

else

printf("YES\n");

}

return 0;

}

11、KM算法模板——带权二分图的最大匹配

#include<string.h>

#include<stdio.h>

#include<math.h>

const int maxn = 101;

const int INF = 0x3fffffff;

int w[maxn][maxn];//权值

int lx[maxn],ly[maxn]; //顶标

int linky[maxn];

int visx[maxn],visy[maxn];//访问

int slack[maxn];

int nx,ny;//x和y的个数，

int find(int x)

{

visx[x] = 1;

for(int y = 0; y < ny; y++)

{

if(visy[y])

continue;

int t = lx[x] + ly[y] - w[x][y];

if(t==0)

{

visy[y] = 1;

if(linky[y]==-1 || find(linky[y]))

{

linky[y] = x;

return 1; //找到增广轨

}

}

else if(slack[y] > t)

slack[y] = t;

}

return 0; //没有找到增广轨（说明顶点x没有对应的匹配，与完备匹配(相等子图的完备匹配)不符）

}

int KM() //返回最优匹配的值

{

int i,j;

memset(linky,-1,sizeof(linky));

memset(ly,0,sizeof(ly));

for(i = 0; i < nx; i++)

for(j = 0,lx[i] = -INF; j < ny; j++)

if(w[i][j] > lx[i])

lx[i] = w[i][j];

for(int x = 0; x < nx; x++)

{

for(i = 0; i < ny; i++)

slack[i] = INF;

while(1)

{

memset(visx,0,sizeof(visx));

memset(visy,0,sizeof(visy));

if(find(x)) //找到增广轨，退出

break;

int d = INF;

for(i = 0; i < ny; i++) //没找到，对l做调整(这会增加相等子图的边)，重新找

{

if(!visy[i] && d > slack[i])

d = slack[i];

}

for(i = 0; i < nx; i++)

{

if(visx[i])

lx[i] -= d;

}

for(i = 0; i < ny; i++)

{

if(visy[i])

ly[i] += d;

else

slack[i] -= d;

}

}

}

int result = 0;

for(i = 0; i < ny; i++)

if(linky[i]>-1)

result += w[linky[i]][i];

return result;

}

int main()

{

while(1)

{

scanf("%d%d",&nx,&ny);

int a,b,c;

while(scanf("%d%d%d",&a,&b,&c),a+b+c)//a到b 的权值为c

{

w[a][b]=c;

}

printf("%d\n",KM());

break;

}

return 0;

}

12、带花树开花算法

//黑箱模板……

#include <cstdio>

#include <cstring>

#include <iostream>

#include <queue>

using namespace std;

const int N = 250;

// 并查集维护

int belong[N];

int findb(int x) {

return belong[x] == x ? x : belong[x] = findb(belong[x]);

}

void unit(int a, int b) {

a = findb(a);

b = findb(b);

if (a != b) belong[a] = b;

}

int n, match[N];

vector<int> e[N];

int Q[N], rear;

int next[N], mark[N], vis[N];

// 朴素算法求某阶段中搜索树上两点x, y的最近公共祖先r

int LCA(int x, int y) {

static int t = 0; t++;

while (true) {

if (x != -1) {

x = findb(x); // 点要对应到对应的花上去

if (vis[x] == t) return x;

vis[x] = t;

if (match[x] != -1) x = next[match[x]];

else x = -1;

}

swap(x, y);

}

}

void group(int a, int p) {

while (a != p) {

int b = match[a], c = next[b];

// next数组是用来标记花朵中的路径的，综合match数组来用，实际上形成了

// 双向链表，如(x, y)是匹配的，next[x]和next[y]就可以指两个方向了。

if (findb(c) != p) next[c] = b;

// 奇环中的点都有机会向环外找到匹配，所以都要标记成S型点加到队列中去，

// 因环内的匹配数已饱和，因此这些点最多只允许匹配成功一个点，在aug中

// 每次匹配到一个点就break终止了当前阶段的搜索，并且下阶段的标记是重

// 新来过的，这样做就是为了保证这一点。

if (mark[b] == 2) mark[Q[rear++] = b] = 1;

if (mark[c] == 2) mark[Q[rear++] = c] = 1;

unit(a, b); unit(b, c);

a = c;

}

}

// 增广

void aug(int s) {

for (int i = 0; i < n; i++) // 每个阶段都要重新标记

next[i] = -1, belong[i] = i, mark[i] = 0, vis[i] = -1;

mark[s] = 1;

Q[0] = s; rear = 1;

for (int front = 0; match[s] == -1 && front < rear; front++) {

int x = Q[front]; // 队列Q中的点都是S型的

for (int i = 0; i < (int)e[x].size(); i++) {

int y = e[x][i];

if (match[x] == y) continue; // x与y已匹配，忽略

if (findb(x) == findb(y)) continue; // x与y同在一朵花，忽略

if (mark[y] == 2) continue; // y是T型点，忽略

if (mark[y] == 1) { // y是S型点，奇环缩点

int r = LCA(x, y); // r为从i和j到s的路径上的第一个公共节点

if (findb(x) != r) next[x] = y; // r和x不在同一个花朵，next标记花朵内路径

if (findb(y) != r) next[y] = x; // r和y不在同一个花朵，next标记花朵内路径

// 将整个r -- x - y --- r的奇环缩成点，r作为这个环的标记节点，相当于论文中的超级节点

group(x, r); // 缩路径r --- x为点

group(y, r); // 缩路径r --- y为点

}

else if (match[y] == -1) { // y自由，可以增广，R12规则处理

next[y] = x;

for (int u = y; u != -1; ) { // 交叉链取反

int v = next[u];

int mv = match[v];

match[v] = u, match[u] = v;

u = mv;

}

break; // 搜索成功，退出循环将进入下一阶段

}

else { // 当前搜索的交叉链+y+match[y]形成新的交叉链，将match[y]加入队列作为待搜节点

next[y] = x;

mark[Q[rear++] = match[y]] = 1; // match[y]也是S型的

mark[y] = 2; // y标记成T型

}

}

}

}

bool g[N][N];

int main()

{

scanf("%d", &n);

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

for (int j = 0; j < n; j++) g[i][j] = false;

// 建图，双向边

int x, y;

while (scanf("%d%d", &x, &y) != EOF)

{

x--, y--;

if (x != y && !g[x][y])

e[x].push\_back(y), e[y].push\_back(x);

g[x][y] = g[y][x] = true;

}

// 增广匹配

for (int i = 0; i < n; i++) match[i] = -1;

for (int i = 0; i < n; i++) if (match[i] == -1) aug(i);

// 输出答案

int tot = 0;

for (int i = 0; i < n; i++) if (match[i] != -1) tot++;

printf("%d\n", tot);

for (int i = 0; i < n; i++) if (match[i] > i)

printf("%d %d\n", i + 1, match[i] + 1);

return 0;

}

13、最小比例生成树

//求最优比例生成树

#include<stdlib.h>

#include<stdio.h>

#include<math.h>

#include<string.h>

#define Maxvalue 99999999

int n,x[1000],y[1000],h[1000];

double cost[1000][1000],dist[1000][1000],dis[1000];

double prim(double p);

bool flag[1000];

int main()

{

int i,j;

double a,b;

while(scanf("%d",&n)==1&&n)

{

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

scanf("%d%d%d",&x[i],&y[i],&h[i]); //俩点间的路径是坐标间的距离 x,y 两点间的权值是h相减

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

{

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

{

b=(x[i]-x[j])\*(x[i]-x[j])+(y[i]-y[j])\*(y[i]-y[j]);

dist[i][j]=sqrt(b);

dist[j][i]=dist[i][j];

cost[i][j]=h[i]-h[j];

if (cost[i][j]<0)

cost[i][j]=-cost[i][j];

cost[j][i]=cost[i][j];

}

}

a=0;

while (1)

{

b=prim(a);

if (fabs(b-a)<0.0001)

break;

a=b;

}

printf("%.3lf\n",b);

}

return 0;

}

double prim(double p)

{

int i,j,k,pre[1000]={0};

double mincost,v,totcost=0,totdist=0;

dis[0]=0;

memset(flag,false,sizeof(flag));

for (j=1;j<n;j++)

dis[j]=cost[j][0]-p\*dist[j][0];

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

{

mincost=Maxvalue;

for (j=1;j<n;j++)

{

if (flag[j]==false && mincost>dis[j])

{

mincost=dis[j];

k=j;

}

}

flag[k]=true;

totcost+=cost[pre[k]][k];

totdist+=dist[pre[k]][k];

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

{

if (flag[j]==false)

{

if ((v=cost[k][j]-p\*dist[k][j])<dis[j])

{

dis[j]=v;

pre[j]=k;

}

}

}

}

return totcost/totdist;

}

14、左偏树模板

#include <cstdio>

#include <algorithm>

using namespace std;

#define typec int

const int na = -1;

const int N = 1000;

struct node

{

typec key;

int l, r, f, dist;

}tr[N];

// find i's root

int iroot(int i)

{

if(i == na)

{

return i;

}

while(tr[i].f != na)

{

i = tr[i].f;

}

return i;

}

// two root: rx, ry

int merge(int rx, int ry)

{

if(rx == na)

{

return ry;

}

if(ry == na)

{

return rx;

}

if(tr[rx].key > tr[ry].key)

{

swap(rx, ry);

}

int r = merge(tr[rx].r, ry);

tr[rx].r = r;

tr[r].f = rx;

if(tr[r].dist > tr[tr[rx].l].dist)

{

swap(tr[rx].l, tr[rx].r);

}

if(tr[rx].r == na)

{

tr[rx].dist = 0;

}

else

{

tr[rx].dist = tr[tr[rx].r].dist + 1;

}

return rx;

}

// add a new node (i, key)

int ins(int i, typec key, int root)

{

tr[i].key = key;

tr[i].l = tr[i].r = tr[i].f = na;

tr[i].dist = 0;

return root = merge(root, i);

}

// delete node i

int del(int i)

{

if(i == na)

{

return i;

}

int x, y, l, r;

l = tr[i].l;

r = tr[i].r;

y = tr[i].f;

tr[i].l = tr[i].r = tr[i].f = na;

tr[x=merge(l,r)].f = y;

if(y != na && tr[y].l == i)

{

tr[y].l = x;

}

if(y != na && tr[y].r == i)

{

tr[y].r = x;

}

for(; y != na; x = y, y = tr[y].f)

{

if(tr[tr[y].l].dist < tr[tr[y].r].dist)

{

swap(tr[y].l, tr[y].r);

}

if(tr[tr[y].r].dist + 1 == tr[y].dist)

{

break;

}

tr[y].dist = tr[tr[y].r].dist + 1;

}

if(x != na)

{

return iroot(x);

}

else

{

return iroot(y);

}

}

node top(int root)

{

return tr[root];

}

node pop(int &root)

{

node out = tr[root];

int l = tr[root].l, r = tr[root].r;

tr[root].l = tr[root].r = tr[root].f = na;

tr[l].f = tr[r].f = na;

root = merge(l, r);

return out;

}

int add(int i, typec val)

{

if(i == na)

{

return i;

}

if(tr[i].l == na && tr[i].r == na && tr[i].f == na)

{

tr[i].key += val;

return i;

}

typec key = tr[i].key + val;

int rt = del(i);

return ins(i, key, rt);

}

void init(int n)

{

for(int i = 1; i < N; i++)

{

scanf("%d", &tr[i].key);

tr[i].l = tr[i].r = tr[i].f = na;

tr[i].dist = 0;

}

}

// print the info of node i

void print(int i)

{

printf("node %d : l-> %d, r-> %d, f-> %d, dist-> %d\n", i, tr[i].l, tr[i].r, tr[i].f, tr[i].dist);

}

int main()

{

int root = na;

for(int i = 1; i < 16; i++)

{

root = ins(i, i, root);

}

for(int i = 1; i < 16; i++)

{

print(i);

}

del(1);

for(int i = 1; i < 16; i++)

{

print(i);

}

return 0;

}

15、伸展树模板

#include<stdio.h>

#include<string.h>

#define mem(a,b) memset(a,b,sizeof(a))

#define min(a,b) a<b?a:b

#define INF 1000000007

#define maxn 100010

using namespace std;

typedef long long ll;

typedef unsigned long long ull;

struct splaytree //伸展树

{

int pre[maxn];//前驱

int key[maxn];//键值

int ch[maxn][2];//左右孩子,0为左孩子,1为右孩子

int root,tot;//根节点，节点数量

splaytree()

{

tot=root=0;

mem(ch,0);

mem(pre,0);

mem(key,0);

}

void newnode(int &r,int father,int k)

{//在r位置，父亲为father，新建一个值点。

r=++tot;

pre[r]=father;

key[r]=k;

ch[r][0]=ch[r][1]=0;

}

void rotate(int x,int kind)//kind=1表示右旋，kind=0表示左旋

{

int y=pre[x];

ch[y][!kind]=ch[x][kind];

pre[ch[x][kind]]=y;

ch[x][kind]=y;

if(pre[y]) ch[pre[y]][ch[pre[y]][1]==y]=x;

pre[x]=pre[y];

pre[y]=x;

}

void splay(int x,int goal) //把x伸展到目标位置

{

while(pre[x]!=goal)

{

if(pre[pre[x]]==goal) rotate(x,ch[pre[x]][0]==x);

else

{

int y=pre[x];

int kind=ch[pre[y]][0]==y;

if(ch[y][kind]==x) rotate(x,!kind),rotate(x,kind);

else rotate(y,kind),rotate(x,kind);

}

}

if(!goal) root=x;//把root更新成x

}

int insert(int x)//插入值x

{

int r=root;

while(ch[r][key[r]<x])

{

if(key[r]==x) {splay(r,0);return 0;}

r=ch[r][key[r]<x];

}

newnode(ch[r][x>key[r]],r,x);

splay(ch[r][x>key[r]],0);

return 1;

}

int find(int x)//查找键值为x的结点编号

{

int r=root;

if(!r) return -INF;//树未建，未找到

if(key[r]==x) return r;

while(ch[r][x>key[r]])

{

r=ch[r][x>key[r]];

if(key[r]==x) return r;

}

return -INF;//未找到

}

int get\_pre(int x)//寻找节点x的前驱的值，未找到则返回INF

{

int r=ch[x][0];

if(!r) return -INF;

while(ch[r][1]) r=ch[r][1];

return key[r];

}

int get\_next(int x)

{

int r=ch[x][1];

if(!r) return INF;

while(ch[r][0]) r=ch[r][0];

return key[r];

}

};

int main()

{

int sum=0,a,n;

splaytree tree;

scanf("%d",&n);

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

{

scanf("%d",&a);

if(!i)

{

sum+=a;

tree.newnode(tree.root,0,a);

continue;

}

if(!tree.insert(a)) continue;

int x=tree.get\_pre(tree.root);

int y=tree.get\_next(tree.root);

sum+=min(a-x,y-a);

}

printf("%d",sum);

return 0;

}

16、求图的割点

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

const int N=105;

struct node

{

int to;

int next;

}p[N\*N];

int dfn[N],low[N];//用dfn[u]记录节点u在DFS过程中被遍历到的次序号，

//low[u]记录节点u或u的子树通过非父子边追溯到最早的祖先节点（即DFS次序号最小）

int head[N],vis[N];

int cut [N];

int m,e,root;

int min (int a,int b)

{

return a<b?a:b;

}

void add(int u,int v)

{

p[e].to=v;

p[e].next=head[u];

head[u]=e++;

}

void dfs (int u,int father,int a)

{

int son=0;

vis[u]=1;

dfn[u]=low[u]=a;

for(int i=head[u];i!=-1;i=p[i].next)

{

int v=p[i].to;

if(vis[v]==1&&v!=father)

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

if(vis[v]==0)

{

dfs(v,u,a+1);

son++;

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

if((u==root&&son>1)||(u!=root&&low[v]>=dfn[u]))

cut[u]=1;

}

}

vis[u]=2;

}

int main ( )

{

while(scanf("%d",&m)!=EOF&&m!=0)

{

memset(dfn,0,sizeof(dfn));

memset(low,0,sizeof(low));

memset(head,-1,sizeof(head));

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

memset(cut,0,sizeof(cut));

int x;

e=0;

while(scanf("%d",&x)!=EOF&&x!=0)

{

while(getchar()!='\n')

{

int y; //表示x和y都有相连 且是无向图

scanf("%d",&y);

add(x,y);

add(y,x);

}

}

root =1;

dfs(1,-1,1);

int sum=0;

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

{

if(cut[i]==1)

sum++;

}

printf("%d\n",sum);

}

return 0;

}

五、数论

1、欧几里得和扩展欧几里得

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

int gcd (int m,int n)

{

int flag;

while(n>0)

{

flag=m%n;

m=n;

n=flag;

}

return m;

}

/\*int gcd(int a,int b) //之前的算法

{

int i;

int ans =1;

if(a>b)

{

a^=b;

b^=a;

a^=b;

}

for(i=2;i<=a;)

{

if(a%i==0&&b%i==0)

{

a/=i;

b/=i;

ans\*=i;

continue;

}

i++;

}

return ans;

}

\*/

int x,y;

int gcdass(int a,int b)//求出AX+BY =GCD （A,B）的xy

{

if(b==0)

{

x=1;

y=0;

return a;

}

int e=gcdass(b,a%b);

int t;

t=x;

x=y;

y=t=a/b\*y;

return e;

}

int main ()

{

//printf("%d\n",gcd(100,150));

return 0;

}

2、欧拉函数

对正整数n，欧拉函数是少于或等于n的数中与n互质的数的数目。例如euler(8)=4，因为1,3,5,7均和8互质。

Euler函数表达通式：euler(x)=x(1-1/p1)(1-1/p2)(1-1/p3)(1-1/p4)…(1-1/pn),

其中p1,p2……pn为x的所有素因数，x是不为0的整数。euler(1)=1（唯一和1互质的数就是1本身）。

#include<stdio.h> //欧拉之实现

long long ef(long long n)

{

long long cnt=n;

long long i;

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

if(n%i==0)

{

cnt -=cnt/i; // m-m/p

while(n%i==0)

n/=i;

}

return cnt;

}

int main()

{

long long m,n;

long long count;

while(scanf("%lld%lld",&m,&n)!=EOF)

{

int i;

long long int ans=0;

for(i=m;i<=n;i++)

ans+=ef(i);

printf("%lld\n",ans);

}

return 0;

}

/\*线性筛O(n)时间复杂度内筛出a-b内欧拉函数值\*/

#include<stdio.h>

#include<math.h>

#define maxn 3000000

int phi[3000000+100];

void get\_PHI()

{

int i,j;

for (i = 1; i <= maxn; i++) phi[i] = i;

for (i = 2; i <= maxn; i += 2) phi[i] /= 2;

for (i = 3; i <= maxn; i += 2) if(phi[i] == i)

{

for (j = i; j <= maxn; j += i)

phi[j] = phi[j] / i \* (i - 1);

}

}

int main()

{

int a,b,i;

get\_PHI();

while(scanf("%d%d",&a,&b)!=EOF)

{

\_\_int64 sum=0;

for(i=a;i<=b;i++)

sum+=phi[i];

printf("%I64d\n",sum);

}

return 0;

}

3、欧拉判别式a^((p-1)2)=(ap)mod p

// 公式 ( a^((p-1)/2) ）mod p=(a/p)mod p

#include<iostream>

#include<cmath>

#include<algorithm>

int MIN(int a,int b) { if( a<b ) return a; else return b; }

int MAX(int a,int b) { if( a>b ) return a; else return b; }

#define CLR(NAME,VALUE) memset(NAME,VALUE,sizeof(NAME))

using namespace std;

typedef \_\_int64 LL;

const int MAXPRI=1000000+10;

bool isNotPri[MAXPRI];

int priList[MAXPRI],priNum;

void CalPri() { //线性筛法

int i,j;

memset(isNotPri,0,sizeof(isNotPri));

isNotPri[0]=isNotPri[1]=1;

for(i=4;i<MAXPRI;i+=2) {

isNotPri[i]=1;

}

priNum=0;

for(i=3;i<MAXPRI;i+=2) {

if( !isNotPri[i] ) {

priList[priNum++]=i;

}

for(j=0;j<priNum&&i\*priList[j]<MAXPRI;++j) {

isNotPri[i\*priList[j]]=1;

if( i%priList[j]==0 ) {

break;

}

}

}

}

LL MyPow(LL a,LL p,LL m) {

if( p==0 ) return 1;

if( p==1 ) return a%m;

LL tmp=MyPow(a,p/2,m);

if( p&1 ) return ((tmp\*tmp)%m)\*a%m;

else return (tmp\*tmp)%m;

}

int Cal(int a,int n) {

if( a%n==0%n ) return 0;

LL tmp=MyPow(a,(n-1)/2,n);

if( tmp==1%n ) return 1;

else return -1;

}

int main() {

int a,n,cnt,ans,i,tmp;

CalPri();

while( scanf("%d%d",&a,&n)!=EOF ) {

ans=1;

if( isNotPri[n] ) {

i=0;

while( n!=1 ) {

cnt=0;

while( n%priList[i]==0 && n!=1 ) {

n/=priList[i];

++cnt;

}

++i;

if( cnt!=0 ) {

tmp=Cal(a,priList[i-1]);

ans\*=pow(tmp,cnt);

}

}

}

else {

ans=Cal(a,n);

}

printf("%d\n",ans);

}

return 0;

}

4、求解模方程a^x=b(mod n)

//hdu 2815 Mod Tree

#include <cstdio>

#include <cstring>

#include <cmath>

#include <map>

#include <iostream>

#include <algorithm>

using namespace std;

#define LL \_\_int64

LL gcd(LL a,LL b)

{

return b==0?a:gcd(b,a%b);

}

//拓展欧几里得定理，求ax+by=gcd(a,b)的一组解(x,y),d=gcd(a,b)

void gcd\_mod(LL a,LL b,LL &d,LL &x,LL &y)

{

if(!b){d=a;x=1;y=0;}

else{gcd\_mod(b,a%b,d,y,x);y-=x\*(a/b);}

}

//求解模方程d\*a^(x-c)=b(mod n)。d,a和n互质，无解返回-1

LL log\_mod (LL a,LL b,LL n,LL c,LL d)

{

LL m,v,e=1,i,x,y,dd;

m=ceil(sqrt(n+0.5)); //x=i\*m+j

map<LL,LL>f;

f[1]=m;

for(i=1;i<m;i++) //建哈希表，保存a^0,a^1,...,a^m-1

{

e=(e\*a)%n;

if(!f[e])f[e]=i;

}

e=(e\*a)%n;//e=a^m

for(i=0;i<m;i++)//每次增加m次方，遍历所有1<=f<=n

{

gcd\_mod(d,n,dd,x,y);//d\*x+n\*y=1-->(d\*x)%n=1-->d\*(x\*b)%n==b

x=(x\*b%n+n)%n;

if(f[x])

{

LL num=f[x];

f.clear();//需要清空，不然会爆内存

return c+i\*m+(num==m?0:num);

}

d=(d\*e)%n;

}

return -1;

}

int main()

{

LL a,b,n;

while(scanf("%I64d%I64d%I64d",&a,&n,&b)!=EOF)

{

if(b>=n)

{

printf("Orz,I can’t find D!\n");

continue;

}

if(b==0)

{

printf("0\n");

continue;

}

LL ans=0,c=0,d=1,t;

while((t=gcd(a,n))!=1)

{

if(b%t){ans=-1;break;}

c++;

n=n/t;

b=b/t;

d=d\*a/t%n;

if(d==b){ans=c;break;}//特判下是否成立。

}

if(ans!=0)

{

if(ans==-1){printf("Orz,I can’t find D!\n");}

else printf("%I64d\n",ans);

}

else

{

ans=log\_mod(a,b,n,c,d);

if(ans==-1)printf("Orz,I can’t find D!\n");

else printf("%I64d\n",ans);

}

}

return 0;

}

/\*

求解模方程a^x=b(mod n)，n不为素数。拓展Baby Step Giant Step

模板题。

方法：

初始d=1,c=0,i=0;

1.令g=gcd(a,n),若g==1则执行下一步。否则由于a^x=k\*n+b;(k为某一整数),则(a/g)\*a^k=k\*(n/g)+b/g,(b/g为整除，若不成立则无解)

令n=n/g，d=d\*a/g，b=b/g,c++则d\*a^(x-c)=b(mod n),接着重复1步骤。

2.通过1步骤后，保证了a和d都与n互质，方程转换为d\*a^(x-c)=b(mod n)。由于a和n互质，所以由欧拉定理a^phi(n)==1(mod n),(a,n互质)

可知，phi(n)<=n,a^0==1(mod n),所以构成循环，且循环节不大于n。从而推出如果存在解，则必定1<=x<n。(在此基础上我们就可以用

Baby Step Giant Step方法了)

3.令m=ceil(sqrt(n)),则m\*m>=n。用哈希表存储a^0,a^1,..,a^(m-1)，接着判断1~m\*m-1中是否存在解。

4.为了减少时间，所以用哈希表缩减复杂度。分成m次遍历，每次遍历a^m长度。由于a和d都与n互质，所以gcd(d,n)=1，

所以用拓展的欧几里德定理求得d\*x+n\*y=gcd(d,n)=1,的一组整数解(x,y)。则d\*x+n\*y=1-->d\*x%n=(d\*x+n\*y)%n=1-->d\*(x\*b)%n=((d\*x)%n\*b%n)%n=b。

所以若x\*b在哈希表中存在，值为k，则a^k\*d=b(mod n),答案就是ans=k+c+i\*m。如果不存在，则令d=d\*a^m,i++后遍历下一个a^m，直到遍历a^0到a^(m-1)还未找到，则说明不解并退出。

\*/

5、中国剩余定理

#include<cstdio>

#include<cstring>

#include<iostream>

using namespace std;

//注意是\_\_int 64不是\_int 64

//注意是\_\_int 64不是\_int 64

//注意是\_\_int 64不是\_int 64

/\*

根据模的性质，我们容易得知，x+a0\*k均为该方程的解。(k为正整数)

如果多了一个方程：x mod a1 = r1。

那么，我们为了使之间求得的解x0=a0+r0能够同时满足这两个方程，只好令x0=x0+a0\*k，

显然这样做x0仍然满足第一个方程。

这时候我 们相当于要求解这样一个模方程：(x0+a0\*k) mod a1 = r1。

这个方程我们可以用拓展欧几里得算法求得k的值。

这样，只要令x0变成x0+a0\*k，就能同时满足这两个方程了。

推而广之，对于方程x mod ai = ri，假如我们之前求得的解为X，

那么我们要令X变成X+k\*LCM(a0,a1,a2...ai-1)，使得它满足这个方程。

k我们可以用拓展欧几里得 算法求解，LCM可以在每一次更新，

这样就能在接近O(klogk)的时间复杂度内解决这个问题了。

\*/

\_\_int64 x;

\_\_int64 y;

\_\_int64 t;

\_\_int64 gcdass(\_\_int64 a,\_\_int64 b)//求出AX+BY =GCD （A,B）的xy

{

if(b==0)

{

x=1;

y=0;

return a;

}

\_\_int64 e=gcdass(b,a%b);

t=x;

x=y;

y=t-a/b\*y;

return e;

}

int main ()

{

// freopen ("D:\\text.txt","r",stdin);

\_\_int64 m1,m2,r1,r2,t,d,c;

int flag=0;

int n;

while(~scanf("%d",&n))

{

flag=0;

scanf("%I64d%I64d",&m1,&r1);

n--;

while(n--)

{

scanf("%I64d%I64d",&m2,&r2);

if(flag)

continue;

d=gcdass(m1,m2);

c=r2-r1;

if(c%d)

{

flag=1;

continue;

}

t=m2/d;

x=(x\*c/d%t+t)%t;//x就是之前的欧几里得中的k

r1+=m1\*x;

m1=m1\*m2/d;//最小公倍数

}

if(flag)

printf("-1\n");

else

printf("%I64d\n",r1);

}

return 0;

}

6、判断素数

#include<iostream>

#include<cstdio>

#include<time.h>

#include<stdlib.h>

using namespace std;

typedef long long ll;

int T,kase = 1,i,j,k,n,m;

ll mult(ll x,ll y,ll mod) // 防止x\*y爆long long;

{

ll ans = 0;x %= mod;

while(y){

if(y&1) ans += x, y--;

if(ans >= mod) ans -= mod;

y >>= 1;

x <<= 1;

if(x >= mod) x -= mod;

}

return ans;

}

ll pow(ll a,ll n,ll mod)

{

a %= mod;

ll ans = 1;

while(n){

if(n&1) ans = mult(ans,a,mod);

a = mult(a,a,mod);

n >>= 1;

}

return ans;

}

bool Miller\_Rabin(ll n)

{

if(n <= 2) return n == 2;

if(n%2 == 0) return false;

ll t = n - 1;

while(t%2 == 0) t >>= 1;

srand(time(NULL));

for(int i = 0;i < 15;i++){

ll p = rand()%(n-2)+2;

if(n%p == 0) return false;

ll tmp = t;

ll x = pow(p,t,n); // p[i]^t % n;

while(tmp < n){

ll y = mult(x,x,n);

if(y == 1 && x != 1 && x != n-1) return false;

x = y;

tmp <<= 1;

}

if(x != 1) return false; // Fermat检测

}

return true;

}

int main()

{

ll x;

scanf("%d",&T);

while(T--){

scanf("%lld",&x);

puts(Miller\_Rabin(x)?"Yes":"No");

}

return 0;

}

7、Pollard\_rho大整数分解 模板

#include <iostream>

#include <stdlib.h>

#include <string.h>

#include <algorithm>

#include <stdio.h>

const int Times = 10;

const int N = 5500;

using namespace std;

typedef long long LL;

LL ct, cnt;

LL fac[N], num[N];

LL gcd(LL a, LL b)

{

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

}

LL multi(LL a, LL b, LL m)

{

LL ans = 0;

a %= m;

while(b)

{

if(b & 1)

{

ans = (ans + a) % m;

b--;

}

b >>= 1;

a = (a + a) % m;

}

return ans;

}

LL quick\_mod(LL a, LL b, LL m)

{

LL ans = 1;

a %= m;

while(b)

{

if(b & 1)

{

ans = multi(ans, a, m);

b--;

}

b >>= 1;

a = multi(a, a, m);

}

return ans;

}

bool Miller\_Rabin(LL n)

{

if(n == 2) return true;

if(n < 2 || !(n & 1)) return false;

LL m = n - 1;

int k = 0;

while((m & 1) == 0)

{

k++;

m >>= 1;

}

for(int i=0; i<Times; i++)

{

LL a = rand() % (n - 1) + 1;

LL x = quick\_mod(a, m, n);

LL y = 0;

for(int j=0; j<k; j++)

{

y = multi(x, x, n);

if(y == 1 && x != 1 && x != n - 1) return false;

x = y;

}

if(y != 1) return false;

}

return true;

}

LL pollard\_rho(LL n, LL c)

{

LL i = 1, k = 2;

LL x = rand() % (n - 1) + 1;

LL y = x;

while(true)

{

i++;

x = (multi(x, x, n) + c) % n;

LL d = gcd((y - x + n) % n, n);

if(1 < d && d < n) return d;

if(y == x) return n;

if(i == k)

{

y = x;

k <<= 1;

}

}

}

void find(LL n, int c)

{

if(n == 1) return;

if(Miller\_Rabin(n))

{

fac[ct++] = n;

return ;

}

LL p = n;

LL k = c;

while(p >= n) p = pollard\_rho(p, c--);

find(p, k);

find(n / p, k);

}

int main()

{

LL n;

while(cin>>n)

{

ct = 0;

find(n, 120);

sort(fac, fac + ct);

num[0] = 1;

int k = 1;

for(int i=1; i<ct; i++)

{

if(fac[i] == fac[i-1])

++num[k-1];

else

{

num[k] = 1;

fac[k++] = fac[i];

}

}

cnt = k;

for(int i=0; i<cnt; i++)

cout<<fac[i]<<"^"<<num[i]<<" ";

cout<<endl;

}

return 0;

}

8、 组合公式

1)、C(m,n)

C(m,n)=C(m-1,n)+C(m-1,n-1)

derangement D(n) = n!(1 - 1/1! + 1/2! - 1/3! + ... + (-1)^n/n!)

= (n-1)(D(n-2) - D(n-1))

Q(n) = D(n) + D(n-1)

卢卡斯定理

表达式

C(n,m)%p=C(n/p,m/p)\*C(n%p,m%p)

适用领域范围

大组合数求模

求解组合数C(n,m)，即从n个相同物品中取出m个的方案数，由于结果可能非常大，对结果模10007即可。

方案1: 暴力求解，C(n,m)=n\*(n-1)\*...\*(n-m+1)/m!，n<=15

方案2: 打表，C(n,m)=C(n-1,m-1)+C(n-1,m)，n<=10,000

方案3: 质因数分解，C(n,m)=n!/(m!\*(n-m)!)，C(n,m)=p1a1-b1-c1p2a2-b2-c2…pkak-bk-ck,n<=10,000,000

方案4: Lucas定理，将m,n化为p进制,有:C(n,m)=C(n0,m0)\*C(n1,m1)...(mod p)，算一个不是很大的C(n,m)%p,p为素数，化为线性同余方程,用扩展的欧几里德定理求解，n在int范围内，修改一下可以满足long long范围内。

方案1:

int Combination(int n, int m)

{

const int M = 10007;

int ans = 1;

for(int i=n; i>=(n-m+1); --i)

ans \*= i;

while(m)

ans /= m--;

return ans % M;

}

方案2:

const int M = 10007;

const int MAXN = 1000;

int C[MAXN+1][MAXN+1];

void Initial()

{

int i,j;

for(i=0; i<=MAXN; ++i)

{

C[0][i] = 0;

C[i][0] = 1;

}

for(i=1; i<=MAXN; ++i)

{

for(j=1; j<=MAXN; ++j)

C[i][j] = (C[i-1][j] + C[i-1][j-1]) % M;

}

}

int Combination(int n, int m)

{

return C[n][m];

}

方案3：

//用筛法生成素数

const int MAXN = 1000000;

bool arr[MAXN+1] = {false};

vector<int> produce\_prim\_number()

{

vector<int> prim;

prim.push\_back(2);

int i,j;

for(i=3; i\*i<=MAXN; i+=2)

{

if(!arr[i])

{

prim.push\_back(i);

for(j=i\*i; j<=MAXN; j+=i)

arr[j] = true;

}

}

while(i<=MAXN)

{

if(!arr[i])

prim.push\_back(i);

i+=2;

}

return prim;

}

//计算n!中素因子p的指数

int Cal(int x, int p)

{

int ans = 0;

long long rec = p;

while(x>=rec)

{

ans += x/rec;

rec \*= p;

}

return ans;

}

//计算n的k次方对M取模，二分法

int Pow(long long n, int k, int M)

{

long long ans = 1;

while(k)

{

if(k&1)

{

ans = (ans \* n) % M;

}

n = (n \* n) % M;

k >>= 1;

}

return ans;

}

//计算C(n,m)

int Combination(int n, int m)

{

const int M = 10007;

vector<int> prim = produce\_prim\_number();

long long ans = 1;

int num;

for(int i=0; i<prim.size() && prim[i]<=n; ++i)

{

num = Cal(n, prim[i]) - Cal(m, prim[i]) - Cal(n-m, prim[i]);

ans = (ans \* Pow(prim[i], num, M)) % M;

}

return ans;

}

方案4：

#include <stdio.h>

const int M = 10007;

int ff[M+5]; //打表，记录n!，避免重复计算

//求最大公因数

int gcd(int a,int b)

{

if(b==0)

return a;

else

return gcd(b,a%b);

}

//解线性同余方程，扩展欧几里德定理

int x,y;

void Extended\_gcd(int a,int b)

{

if(b==0)

{

x=1;

y=0;

}

else

{

Extended\_gcd(b,a%b);

long t=x;

x=y;

y=t-(a/b)\*y;

}

}

//计算不大的C(n,m)

int C(int a,int b)

{

if(b>a)

return 0;

b=(ff[a-b]\*ff[b])%M;

a=ff[a];

int c=gcd(a,b);

a/=c;

b/=c;

Extended\_gcd(b,M);

x=(x+M)%M;

x=(x\*a)%M;

return x;

}

//Lucas定理

int Combination(int n, int m)

{

int ans=1;

int a,b;

while(m||n)

{

a=n%M;

b=m%M;

n/=M;

m/=M;

ans=(ans\*C(a,b))%M;

}

return ans;

}

int main(void)

{

int i,m,n;

ff[0]=1;

for(i=1;i<=M;i++) //预计算n!

ff[i]=(ff[i-1]\*i)%M;

scanf("%d%d",&n, &m);

printf("%d\n",func(n,m));

return 0;

}

2）、求和公式,k = 1..n

\*1. sum( k ) = n(n+1)/2

\*2. sum( 2k-1 ) = n^2

\*3. sum( k^2 ) = n(n+1)(2n+1)/6

\*4. sum( (2k-1)^2 ) = n(4n^2-1)/3

\*5. sum( k^3 ) = (n(n+1)/2)^2

\*6. sum( (2k-1)^3 ) = n^2(2n^2-1)

\*7. sum( k^4 ) = n(n+1)(2n+1)(3n^2+3n-1)/30

\*8. sum( k^5 ) = n^2(n+1)^2(2n^2+2n-1)/12

\*9. sum( k(k+1) ) = n(n+1)(n+2)/3

\*10. sum( k(k+1)(k+2) ) = n(n+1)(n+2)(n+3)/4

\*12. sum( k(k+1)(k+2)(k+3) ) = n(n+1)(n+2)(n+3)(n+4)/5

六、计算几何

## 0、注意

1. 注意舍入方式(0.5的舍入方向);防止输出-0.

2. 几何题注意多测试不对称数据.

3. 整数几何注意xmult和dmult是否会出界;

符点几何注意eps的使用.

4. 避免使用斜率;注意除数是否会为0.

5. 公式一定要化简后再代入.

6. 判断同一个2\*PI域内两角度差应该是

abs(a1-a2)<beta||abs(a1-a2)>pi+pi-beta;

相等应该是

abs(a1-a2)<eps||abs(a1-a2)>pi+pi-eps;

7. 需要的话尽量使用atan2,注意:atan2(0,0)=0,

atan2(1,0)=pi/2,atan2(-1,0)=-pi/2,atan2(0,1)=0,atan2(0,-1)=pi.

8. cross product = |u|\*|v|\*sin(a)

dot product = |u|\*|v|\*cos(a)

9. (P1-P0)x(P2-P0)结果的意义:

正: <P0,P1>在<P0,P2>顺时针(0,pi)内

负: <P0,P1>在<P0,P2>逆时针(0,pi)内

0 : <P0,P1>,<P0,P2>共线,夹角为0或pi

10. 误差限缺省使用1e-8!

1、凸包

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<math.h>

#define Maxn 100005

#define PF(a) ((a)\*(a))

int stack [Maxn];

int top;

typedef struct POINT

{

double x;

double y;

} POINT;

POINT point [Maxn];

void swap (POINT point[],int x,int y)

{

POINT tmp;

tmp=point[x];

point[x]=point[y];

point[y]=tmp;

return ;

}

double multi(POINT p1,POINT p2,POINT p0) //叉乘

{

return ((p1.x-p0.x)\*(p2.y-p0.y)-(p1.y-p0.y)\*(p2.x-p0.x));

}

double distence(POINT p1,POINT p2) //p1,p2的距离

{

return sqrt((double)(p1.x-p2.x)\*(p1.x-p2.x)+(double)(p1.y-p2.y)\*(p1.y-p2.y));

}

int cmp(const void \*a,const void \*b)

{

POINT \*c=(POINT \*)a;

POINT \*d=(POINT \*)b;

double k=multi(\*c,\*d,point[0]);

if(k<0) return 1;

else if(k==0&&distence(\*c,point[0])>=distence(\*d,point[0])) return 1; //极角相同，比距离

else return -1;

}

void grahamscan(int n)

{

int i,u;

u=0;

for(i = 1;i<= n-1;i++) //找到最左下的点p0

if((point[i].y < point[u].y)||(point[i].y==point[u].y&&point[i].x < point[u].x))

u=i;

swap(point ,0,u);

qsort(point +1,n-1,sizeof(point[0]),cmp);

for(i=0;i<=2;i++)

stack[i]=i;

top=2;

for(i=3;i<=n-1;i++)

{

while(multi(point[i],point[stack[top]],point[stack[top-1]])>=0) //弹出非左转的点

{

if(top==0)break;

top--;

}

top++;

stack[top] = i;

}

}

//求凸包的面积

double polygonArea(int n,POINT p[])

{

double area;

int i;

area = 0;

for(i = 1;i <= n;i++){

area += (p[stack[i - 1]].x \* p[stack[i % n]].y - p[stack[i % n]].x \* p[stack[i - 1]].y);

}

return fabs(area) / 2;

}

//求凸包的周长

double polygonLen(int n,POINT p[])

{

double len=0;

for(int i=0;i<top;i++)

len+= sqrt((double) PF(p[stack[i +1]].x-p[stack[i ]].x)+(double)PF(p[stack[i +1]].y-p[stack[i ]].y) );

len+= sqrt((double) PF(p[stack[top]].x-p[stack[0 ]].x)+(double)PF(p[stack[top]].y-p[stack[0 ]].y) );

return len;

}

int main()

{

int i;

int n;

while(scanf("%d",&n)!=EOF)

{

if(!n)

break;

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

scanf("%lf%lf",&(point[i].x),&(point[i].y));

grahamscan(n);

printf("凸包的点 （点大于三个时）\n");

for(i=0;i<=top;i++)

printf("%lf %lf \n",point[stack[i]].x,point[stack[i]].y);

//求凸包的周长

if(n==1)

{

printf("len=0.00\n");

}

else if(n==2)

{

double dis=sqrt( (point[0].x-point[1].x)\*(point[0].x-point[1].x)+(point[0].y-point[1].y)\*(point[0].y-point[1].y));

printf("len=%.2lf\n",dis);

}

else

printf("len=%.2lf\n",polygonLen(n,point));

//求凸包的面积

printf("area=%.2lf\n",polygonArea(n,point));

}

return 0;

}

2、求凸包的直径

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<math.h> 4

#define Maxn 100005

#define PF(a) ((a)\*(a))

#define max(a,b) a>b?(a) :(b)

#define INF 0x3fffffff

int stack [Maxn];

int top;

typedef struct POINT

{

int x;

int y;

} POINT;

POINT point [Maxn];

POINT wtf[Maxn];

void swap (POINT point[],int x,int y)

{

POINT tmp;

tmp=point[x];

point[x]=point[y];

point[y]=tmp;

return ;

}

double multi(POINT p1,POINT p2,POINT p0) //叉乘

{

return ((p1.x-p0.x)\*(p2.y-p0.y)-(p1.y-p0.y)\*(p2.x-p0.x));

}

double multi2 (POINT p1,POINT p2)

{

return (p1.x\*p2.y-p1.y\*p2.x);

}

double distence(POINT p1,POINT p2) //p1,p2的距离

{

return sqrt((double)(p1.x-p2.x)\*(p1.x-p2.x)+(double)(p1.y-p2.y)\*(p1.y-p2.y));

}

double distence2(POINT p1,POINT p2) //p1,p2的距离的平方

{

return (double) ( (p1.x-p2.x)\*(p1.x-p2.x)+(p1.y-p2.y)\*(p1.y-p2.y));

}

int cmp(const void \*a,const void \*b)

{

POINT \*c=(POINT \*)a;

POINT \*d=(POINT \*)b;

double k=multi(\*c,\*d,point[0]);

if(k<0) return 1;

else if(k==0&&distence(\*c,point[0])>=distence(\*d,point[0])) return 1; //极角相同，比距离

else return -1;

}

void grahamscan(int n)

{

int i,u;

u=0;

for(i = 1;i<= n-1;i++) //找到最左下的点p0

if((point[i].y < point[u].y)||(point[i].y==point[u].y&&point[i].x < point[u].x))

u=i;

swap(point ,0,u);

qsort(point +1,n-1,sizeof(point[0]),cmp);

for(i=0;i<=2;i++)

stack[i]=i;

top=2;

for(i=3;i<=n-1;i++)

{

while(multi(point[i],point[stack[top]],point[stack[top-1]])>=0) //弹出非左转的点

{

if(top==0)break;

top--;

}

top++;

stack[top] = i;

}

}

double rotating\_calipers(POINT ch[], int m) /\*\*旋转卡壳模板\*/

{

int q=1;

double ans=0;

ch[m+1]=ch[0];

for(int i=0;i<=m;i++)

{

POINT flag1,flag2,flag3;

flag1.x=ch[i+1].x-ch[i].x;

flag1.y=ch[i+1].y-ch[i].y;

flag2.x=ch[q+1].x-ch[i].x;

flag2.y=ch[q+1].y-ch[i].y;

flag3.x=ch[q].x-ch[i].x;

flag3.y=ch[q].y-ch[i].y;

while(multi2(flag1,flag2)>multi2(flag1,flag3))

{

q=(q+1)%(m+1);

flag2.x=ch[q+1].x-ch[i].x;

flag2.y=ch[q+1].y-ch[i].y;

flag3.x=ch[q].x-ch[i].x;

flag3.y=ch[q].y-ch[i].y;

}

double fff;

fff=max(distence(ch[i],ch[q]),distence(ch[i+1],ch[q+1]));

ans=max(ans,fff);

//printf("%d\n",ans);

}

return ans;

}

int main()

{

int i;

int n;

while(scanf("%d",&n)!=EOF&&n!=0)

{

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

scanf("%d%d",&(point[i].x),&(point[i].y));

grahamscan(n);

for(i=0;i<=top;i++)

{

//printf("%d %d \n",point[stack[i]].x,point[stack[i]].y);

wtf[i].x=point[stack[i]].x;

wtf[i].y=point[stack[i]].y;

}

printf("%lf\n", rotating\_calipers(wtf, top) );// 输出凸包的直径

}

return 0;

}

3、三角形

1). 半周长 P=(a+b+c)/2

2). 面积 S=aHa/2=absin(C)/2=sqrt(P(P-a)(P-b)(P-c))

3). 中线 Ma=sqrt(2(b^2+c^2)-a^2)/2=sqrt(b^2+c^2+2bccos(A))/2

4). 角平分线 Ta=sqrt(bc((b+c)^2-a^2))/(b+c)=2bccos(A/2)/(b+c)

5). 高线 Ha=bsin(C)=csin(B)=sqrt(b^2-((a^2+b^2-c^2)/(2a))^2)

6). 内切圆半径 r=S/P=asin(B/2)sin(C/2)/sin((B+C)/2)

=4Rsin(A/2)sin(B/2)sin(C/2)=sqrt((P-a)(P-b)(P-c)/P)

=Ptan(A/2)tan(B/2)tan(C/2)

7). 外接圆半径 R=abc/(4S)=a/(2sin(A))=b/(2sin(B))=c/(2sin(C))

4、四边形

D1,D2为对角线,M对角线中点连线,A为对角线夹角

1. a^2+b^2+c^2+d^2=D1^2+D2^2+4M^2
2. S=D1D2sin(A)/2
3. (以下对圆的内接四边形)
4. ac+bd=D1D2
5. S=sqrt((P-a)(P-b)(P-c)(P-d)),P为半周长

5、正n边形:

R为外接圆半径,r为内切圆半径

1. 中心角 A=2PI/n
2. 内角 C=(n-2)PI/n
3. 边长 a=2sqrt(R^2-r^2)=2Rsin(A/2)=2rtan(A/2)
4. 面积 S=nar/2=nr^2tan(A/2)=nR^2sin(A)/2=na^2/(4tan(A/2))

6、圆:

1. 弧长 l=rA
2. 弦长 a=2sqrt(2hr-h^2)=2rsin(A/2)
3. 弓形高 h=r-sqrt(r^2-a^2/4)=r(1-cos(A/2))=atan(A/4)/2
4. 扇形面积 S1=rl/2=r^2A/2
5. 弓形面积 S2=(rl-a(r-h))/2=r^2(A-sin(A))/2

7、棱柱

1. 体积 V=Ah,A为底面积,h为高
2. 侧面积 S=lp,l为棱长,p为直截面周长
3. 全面积 T=S+2A

8、棱锥

1. 体积 V=Ah/3,A为底面积,h为高
2. (以下对正棱锥)
3. 侧面积 S=lp/2,l为斜高,p为底面周长
4. 全面积 T=S+A

9、棱台

1. 体积 V=(A1+A2+sqrt(A1A2))h/3,A1.A2为上下底面积,h为高
2. (以下为正棱台)

3). 侧面积 S=(p1+p2)l/2,p1.p2为上下底面周长,l为斜高

4). 全面积 T=S+A1+A2

10、圆柱:

1. 侧面积 S=2PIrh
2. 全面积 T=2PIr(h+r)
3. 体积 V=PIr^2h

11、圆锥:

1. 母线 l=sqrt(h^2+r^2)
2. 侧面积 S=PIrl
3. 全面积 T=PIr(l+r)
4. 体积 V=PIr^2h/3

12、圆台:

1. 母线 l=sqrt(h^2+(r1-r2)^2)
2. 侧面积 S=PI(r1+r2)l
3. 全面积 T=PIr1(l+r1)+PIr2(l+r2)
4. 体积 V=PI(r1^2+r2^2+r1r2)h/3

13、球:

1. 全面积 T=4PIr^2
2. 体积 V=4PIr^3/3

14、球台:

1. 侧面积 S=2PIrh
2. 全面积 T=PI(2rh+r1^2+r2^2)
3. 体积 V=PIh(3(r1^2+r2^2)+h^2)/6

15、球扇形

1. 全面积 T=PIr(2h+r0),h为球冠高,r0为球冠底面半径
2. 体积 V=2PIr^2h/3

16、多边形

#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);

}

1)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];

}

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];

}

3)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];

}

4)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];

}

5)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;

}

6)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);

}

7)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;

}

17、浮点函数

1. 浮点几何函数库

#include <math.h>

#define eps 1e-8

#define zero(x) (((x)>0?(x):-(x))<eps)

struct point{double x,y;};

struct line{point a,b;};

1. 计算cross product (P1-P0)x(P2-P0)

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);

}

double xmult(double x1,double y1,double x2,double y2,double x0,double y0){

return (x1-x0)\*(y2-y0)-(x2-x0)\*(y1-y0);

}

1. 计算dot product (P1-P0).(P2-P0)

double dmult(point p1,point p2,point p0){

return (p1.x-p0.x)\*(p2.x-p0.x)+(p1.y-p0.y)\*(p2.y-p0.y);

}

double dmult(double x1,double y1,double x2,double y2,double x0,double y0){

return (x1-x0)\*(x2-x0)+(y1-y0)\*(y2-y0);

}

1. 两点距离

double distance(point p1,point p2){

return sqrt((p1.x-p2.x)\*(p1.x-p2.x)+(p1.y-p2.y)\*(p1.y-p2.y));

}

double distance(double x1,double y1,double x2,double y2){

return sqrt((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2));

}

1. 判三点共线

int dots\_inline(point p1,point p2,point p3){

return zero(xmult(p1,p2,p3));

}

int dots\_inline(double x1,double y1,double x2,double y2,double x3,double y3){

return zero(xmult(x1,y1,x2,y2,x3,y3));

}

1. 判点是否在线段上,包括端点

int dot\_online\_in(point p,line l){

return zero(xmult(p,l.a,l.b))&&(l.a.x-p.x)\*(l.b.x-p.x)<eps&&(l.a.y-p.y)\*(l.b.y-p.y)<eps;

}

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;

}

int dot\_online\_in(double x,double y,double x1,double y1,double x2,double y2){

return zero(xmult(x,y,x1,y1,x2,y2))&&(x1-x)\*(x2-x)<eps&&(y1-y)\*(y2-y)<eps;

}

1. 判点是否在线段上,不包括端点

int dot\_online\_ex(point p,line l){

return dot\_online\_in(p,l)&&(!zero(p.x-l.a.x)||!zero(p.y-l.a.y))&&(!zero(p.x-l.b.x)||!zero(p.y-l.b.y));

}

int dot\_online\_ex(point p,point l1,point l2){

return dot\_online\_in(p,l1,l2)&&(!zero(p.x-l1.x)||!zero(p.y-l1.y))&&(!zero(p.x-l2.x)||!zero(p.y-l2.y));

}

int dot\_online\_ex(double x,double y,double x1,double y1,double x2,double y2){

return dot\_online\_in(x,y,x1,y1,x2,y2)&&(!zero(x-x1)||!zero(y-y1))&&(!zero(x-x2)||!zero(y-y2));

}

1. 判两点在线段同侧,点在线段上返回0

int same\_side(point p1,point p2,line l){

return xmult(l.a,p1,l.b)\*xmult(l.a,p2,l.b)>eps;

}

int same\_side(point p1,point p2,point l1,point l2){

return xmult(l1,p1,l2)\*xmult(l1,p2,l2)>eps;

}

1. 判两点在线段异侧,点在线段上返回0

int opposite\_side(point p1,point p2,line l){

return xmult(l.a,p1,l.b)\*xmult(l.a,p2,l.b)<-eps;

}

int opposite\_side(point p1,point p2,point l1,point l2){

return xmult(l1,p1,l2)\*xmult(l1,p2,l2)<-eps;

}

1. 判两直线平行

int parallel(line u,line v){

return zero((u.a.x-u.b.x)\*(v.a.y-v.b.y)-(v.a.x-v.b.x)\*(u.a.y-u.b.y));

}

int parallel(point u1,point u2,point v1,point v2){

return zero((u1.x-u2.x)\*(v1.y-v2.y)-(v1.x-v2.x)\*(u1.y-u2.y));

}

1. 判两直线垂直

int perpendicular(line u,line v){

return zero((u.a.x-u.b.x)\*(v.a.x-v.b.x)+(u.a.y-u.b.y)\*(v.a.y-v.b.y));

}

int perpendicular(point u1,point u2,point v1,point v2){

return zero((u1.x-u2.x)\*(v1.x-v2.x)+(u1.y-u2.y)\*(v1.y-v2.y));

}

1. 判两线段相交,包括端点和部分重合

int intersect\_in(line u,line v){

if (!dots\_inline(u.a,u.b,v.a)||!dots\_inline(u.a,u.b,v.b))

return !same\_side(u.a,u.b,v)&&!same\_side(v.a,v.b,u);

return dot\_online\_in(u.a,v)||dot\_online\_in(u.b,v)||dot\_online\_in(v.a,u)||dot\_online\_in(v.b,u);

}

int intersect\_in(point u1,point u2,point v1,point v2){

if (!dots\_inline(u1,u2,v1)||!dots\_inline(u1,u2,v2))

return !same\_side(u1,u2,v1,v2)&&!same\_side(v1,v2,u1,u2);

return dot\_online\_in(u1,v1,v2)||dot\_online\_in(u2,v1,v2)||dot\_online\_in(v1,u1,u2)||dot\_online\_in(v2,u1,u2);

}

1. 判两线段相交,不包括端点和部分重合

int intersect\_ex(line u,line v){

return opposite\_side(u.a,u.b,v)&&opposite\_side(v.a,v.b,u);

}

int intersect\_ex(point u1,point u2,point v1,point v2){

return opposite\_side(u1,u2,v1,v2)&&opposite\_side(v1,v2,u1,u2);

}

1. 计算两直线交点,注意事先判断直线是否平行!

//线段交点请另外判线段相交(同时还是要判断是否平行!)

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 intersection(point u1,point u2,point v1,point v2){

point ret=u1;

double t=((u1.x-v1.x)\*(v1.y-v2.y)-(u1.y-v1.y)\*(v1.x-v2.x))

/((u1.x-u2.x)\*(v1.y-v2.y)-(u1.y-u2.y)\*(v1.x-v2.x));

ret.x+=(u2.x-u1.x)\*t;

ret.y+=(u2.y-u1.y)\*t;

return ret;

}

1. 点到直线上的最近点

point ptoline(point p,line l){

point t=p;

t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;

return intersection(p,t,l.a,l.b);

}

point ptoline(point p,point l1,point l2){

point t=p;

t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;

return intersection(p,t,l1,l2);

}

1. 点到直线距离

double disptoline(point p,line l){

return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);

}

double disptoline(point p,point l1,point l2){

return fabs(xmult(p,l1,l2))/distance(l1,l2);

}

double disptoline(double x,double y,double x1,double y1,double x2,double y2){

return fabs(xmult(x,y,x1,y1,x2,y2))/distance(x1,y1,x2,y2);

}

1. 点到线段上的最近点

point ptoseg(point p,line l){

point t=p;

t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;

if (xmult(l.a,t,p)\*xmult(l.b,t,p)>eps)

return distance(p,l.a)<distance(p,l.b)?l.a:l.b;

return intersection(p,t,l.a,l.b);

}

point ptoseg(point p,point l1,point l2){

point t=p;

t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;

if (xmult(l1,t,p)\*xmult(l2,t,p)>eps)

return distance(p,l1)<distance(p,l2)?l1:l2;

return intersection(p,t,l1,l2);

}

1. 点到线段距离

double disptoseg(point p,line l){

point t=p;

t.x+=l.a.y-l.b.y,t.y+=l.b.x-l.a.x;

if (xmult(l.a,t,p)\*xmult(l.b,t,p)>eps)

return distance(p,l.a)<distance(p,l.b)?distance(p,l.a):distance(p,l.b);

return fabs(xmult(p,l.a,l.b))/distance(l.a,l.b);

}

double disptoseg(point p,point l1,point l2){

point t=p;

t.x+=l1.y-l2.y,t.y+=l2.x-l1.x;

if (xmult(l1,t,p)\*xmult(l2,t,p)>eps)

return distance(p,l1)<distance(p,l2)?distance(p,l1):distance(p,l2);

return fabs(xmult(p,l1,l2))/distance(l1,l2);

}

1. 矢量V以P为顶点逆时针旋转angle并放大scale倍

point rotate(point v,point p,double angle,double scale){

point ret=p;

v.x-=p.x,v.y-=p.y;

p.x=scale\*cos(angle);

p.y=scale\*sin(angle);

ret.x+=v.x\*p.x-v.y\*p.y;

ret.y+=v.x\*p.y+v.y\*p.x;

return ret;

}

18、面积

#include <math.h>

struct point{double x,y;};

1. 计算cross product (P1-P0)x(P2-P0)

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);

}

double xmult(double x1,double y1,double x2,double y2,double x0,double y0){

return (x1-x0)\*(y2-y0)-(x2-x0)\*(y1-y0);

}

1. 三角形面积,输入三顶点

double area\_triangle(point p1,point p2,point p3){

return fabs(xmult(p1,p2,p3))/2;

}

double area\_triangle(double x1,double y1,double x2,double y2,double x3,double y3){

return fabs(xmult(x1,y1,x2,y2,x3,y3))/2;

}

1. 计算三角形面积,输入三边长

double area\_triangle(double a,double b,double c){

double s=(a+b+c)/2;

return sqrt(s\*(s-a)\*(s-b)\*(s-c));

}

1. 计算多边形面积,顶点按顺时针或逆时针给出

double area\_polygon(int n,point\* p){

double s1=0,s2=0;

int i;

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

s1+=p[(i+1)%n].y\*p[i].x,s2+=p[(i+1)%n].y\*p[(i+2)%n].x;

return fabs(s1-s2)/2;

}

19、三角形详细

#include <math.h>

struct point{double x,y;};

struct line{point a,b;};

double distance(point p1,point p2){

return sqrt((p1.x-p2.x)\*(p1.x-p2.x)+(p1.y-p2.y)\*(p1.y-p2.y));

}

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;

}

1. 外心

point circumcenter(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.x=u.a.x-a.y+b.y;

u.b.y=u.a.y+a.x-b.x;

v.a.x=(a.x+c.x)/2;

v.a.y=(a.y+c.y)/2;

v.b.x=v.a.x-a.y+c.y;

v.b.y=v.a.y+a.x-c.x;

return intersection(u,v);

}

1. 内心

point incenter(point a,point b,point c){

line u,v;

double m,n;

u.a=a;

m=atan2(b.y-a.y,b.x-a.x);

n=atan2(c.y-a.y,c.x-a.x);

u.b.x=u.a.x+cos((m+n)/2);

u.b.y=u.a.y+sin((m+n)/2);

v.a=b;

m=atan2(a.y-b.y,a.x-b.x);

n=atan2(c.y-b.y,c.x-b.x);

v.b.x=v.a.x+cos((m+n)/2);

v.b.y=v.a.y+sin((m+n)/2);

return intersection(u,v);

}

1. 垂心

point perpencenter(point a,point b,point c){

line u,v;

u.a=c;

u.b.x=u.a.x-a.y+b.y;

u.b.y=u.a.y+a.x-b.x;

v.a=b;

v.b.x=v.a.x-a.y+c.y;

v.b.y=v.a.y+a.x-c.x;

return intersection(u,v);

}

1. 重心

//到三角形三顶点距离的平方和最小的点

//三角形内到三边距离之积最大的点

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);

}

1. 到三角形三顶点距离之和最小的点

//费马点

point fermentpoint(point a,point b,point c){

point u,v;

double step=fabs(a.x)+fabs(a.y)+fabs(b.x)+fabs(b.y)+fabs(c.x)+fabs(c.y);

int i,j,k;

u.x=(a.x+b.x+c.x)/3;

u.y=(a.y+b.y+c.y)/3;

while (step>1e-10)

for (k=0;k<10;step/=2,k++)

for (i=-1;i<=1;i++)

for (j=-1;j<=1;j++){

v.x=u.x+step\*i;

v.y=u.y+step\*j;

if (distance(u,a)+distance(u,b)+distance(u,c)>distance(v,a)+distance(v,b)+distance(v,c))

u=v;

}

return u;

}

20、圆详细

#include <math.h>

#define eps 1e-8

struct point{double x,y;};

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);

}

double distance(point p1,point p2){

return sqrt((p1.x-p2.x)\*(p1.x-p2.x)+(p1.y-p2.y)\*(p1.y-p2.y));

}

double disptoline(point p,point l1,point l2){

return fabs(xmult(p,l1,l2))/distance(l1,l2);

}

point intersection(point u1,point u2,point v1,point v2){

point ret=u1;

double t=((u1.x-v1.x)\*(v1.y-v2.y)-(u1.y-v1.y)\*(v1.x-v2.x))

/((u1.x-u2.x)\*(v1.y-v2.y)-(u1.y-u2.y)\*(v1.x-v2.x));

ret.x+=(u2.x-u1.x)\*t;

ret.y+=(u2.y-u1.y)\*t;

return ret;

}

1. 判直线和圆相交,包括相切

int intersect\_line\_circle(point c,double r,point l1,point l2){

return disptoline(c,l1,l2)<r+eps;

}

1. 判线段和圆相交,包括端点和相切

int intersect\_seg\_circle(point c,double r,point l1,point l2){

double t1=distance(c,l1)-r,t2=distance(c,l2)-r;

point t=c;

if (t1<eps||t2<eps)

return t1>-eps||t2>-eps;

t.x+=l1.y-l2.y;

t.y+=l2.x-l1.x;

return xmult(l1,c,t)\*xmult(l2,c,t)<eps&&disptoline(c,l1,l2)-r<eps;

}

1. 判圆和圆相交,包括相切

int intersect\_circle\_circle(point c1,double r1,point c2,double r2){

return distance(c1,c2)<r1+r2+eps&&distance(c1,c2)>fabs(r1-r2)-eps;

}

1. 计算圆上到点p最近点,如p与圆心重合,返回p本身

point dot\_to\_circle(point c,double r,point p){

point u,v;

if (distance(p,c)<eps)

return p;

u.x=c.x+r\*fabs(c.x-p.x)/distance(c,p);

u.y=c.y+r\*fabs(c.y-p.y)/distance(c,p)\*((c.x-p.x)\*(c.y-p.y)<0?-1:1);

v.x=c.x-r\*fabs(c.x-p.x)/distance(c,p);

v.y=c.y-r\*fabs(c.y-p.y)/distance(c,p)\*((c.x-p.x)\*(c.y-p.y)<0?-1:1);

return distance(u,p)<distance(v,p)?u:v;

}

1. 计算直线与圆的交点,保证直线与圆有交点

//计算线段与圆的交点可用这个函数后判点是否在线段上

void intersection\_line\_circle(point c,double r,point l1,point l2,point& p1,point& p2){

point p=c;

double t;

p.x+=l1.y-l2.y;

p.y+=l2.x-l1.x;

p=intersection(p,c,l1,l2);

t=sqrt(r\*r-distance(p,c)\*distance(p,c))/distance(l1,l2);

p1.x=p.x+(l2.x-l1.x)\*t;

p1.y=p.y+(l2.y-l1.y)\*t;

p2.x=p.x-(l2.x-l1.x)\*t;

p2.y=p.y-(l2.y-l1.y)\*t;

}

1. 计算圆与圆的交点,保证圆与圆有交点,圆心不重合

void intersection\_circle\_circle(point c1,double r1,point c2,double r2,point& p1,point& p2){

point u,v;

double t;

t=(1+(r1\*r1-r2\*r2)/distance(c1,c2)/distance(c1,c2))/2;

u.x=c1.x+(c2.x-c1.x)\*t;

u.y=c1.y+(c2.y-c1.y)\*t;

v.x=u.x+c1.y-c2.y;

v.y=u.y-c1.x+c2.x;

intersection\_line\_circle(c1,r1,u,v,p1,p2);

}