IO

```
while(fgets(s,sizeof(s),stdin)!=0)...
freopen(“inputfile”,”r”,stdin);
freopen(“output”,”w”,stdout);
std::ios::sync_with_stdio(false);
mod: (a*b)%c=((a%c)*b)%c
```

Merge Sort (求反序)

```
void Merge(int L,int R){
     if(R-L==1)return;
     int mid=(L+R)/2;
     Merge(L,mid);
     Merge(mid,R);
     for(int p1=L, p2=mid, p=0; p1<=mid && p2<=R; p++){
          if((p1!=mid && ar[p1]<=ar[p2])||p2==R)
               C[p]=ar[p1++];
          else {
               C[p]=ar[p2++];
               if(p1!=mid) ans+=(mid-p1);
          }
     }
     for(int i=L; i<R; i++) ar[i]=C[i-L];
}
```

a^b % c (use “(a*b)%c=((a%c)*b)%c”)

```
int a_b_Mod_c(int a,int b,int c){       // 前提 abc 都是正數
    int digit[32], i=0, resualt=1;
    while(b){ digit[i++]=b%2; b>>=1; }  // 把 b 化成 2 進制
    for(int k=i-1; k>=0; k--){          // 計算(a^b) mod c
        resualt=(resualt*resualt)%c;
        if(digit[k]==1)  resualt=(resualt*a)%c;
    }
    return resualt;
}
```

Binary Search

```
void find_min(){
    int beg,end,mid,best=INT_MAX;
    do{
        mid=(beg+end)>>1;
        if(canbe(mid)){
            best=min(best,mid);
            end=mid;
        }
        else  beg=mid+1;
    }while(beg<end);
}

void fimd_max(){
    int beg,end,mid,best=INT_MAX;
    do{
        mid=(beg+end+1)>>1;
        if(check(mid)){
            best=max(best,mid);
            beg=mid;
        }
        else   end=mid-1;
    }while(beg<end);
    //ans=0 no solution
    //otherwise maximum
    //ans in end
}
///double  find max
do{
    mid=(beg+end)/2.0;
    if(canbe(mid))     beg=mid,ans=max(ans,mid);
    else               end=mid-eps;
}while( fabs(beg-end)>eps );
printf("%lf\n",ans);
```

Disjoint Set

```
memset(Rank, 0, sizeof(Rank));
for(int i=1; i<=n; i++) stu[i]=i;      //Make set
Link(FindSet(i),FindSet(j));
void Link(int x, int y){
     if(x==y)return;
     if(Rank[x]>Rank[y]) stu[y]=x;
     else{
          stu[x]=y;
          if(Rank[x]==Rank[y]) Rank[y]++;
     }
}
int FindSet(int n){ /*return n==stu[n]?stu[n]:FindSet(stu[n]);*/
     if(n!=stu[n]) stu[n]=FindSet(stu[n]);
     return stu[n];
}
```

BFS, DFS, prime sieve

Backtrack

```
permutation:
    dfs(0, n-1);
    void dfs(int a, int b){
            if(a==b){
                for(int i=0; i<n; i++)cout << ans[i] << ' ';
                    cout << endl;
                    return ;
        }
            for (int i=a; i<=b; i++){
                myswap(&ans[a], &ans[i]);
                dfs(a+1,b);
                myswap(&ans[a], &ans[i]);
        }
    }
lotto:
    dfs(0,0);
    void dfs(int dep, int index){
            if(dep==6){
                for(int i=0; i<6; i++) cout << ans[i] << ' '; cout
<< endl;
                return;
            }
            for(int i=index; i<n; i++){
                if(used[i]==0){
                       used[i]=1;
                       ans[dep]=ar[i];
                       dfs(dep+1, i);
                       used[i]=0;
                }
            }
    }
all:
    dfs(0);
    void dfs(int dep){
        if(dep==n){
            for(int i=0; i<n; i++)
                if(s[i])cout << ans[i] << ' ';
            cout << endl;
            return ;
        }
        s[dep] = 1;
        dfs(dep+1);
        s[dep] = 0;
        dfs(dep+1);
    }
```

**

```
    stirng a="";
    a+=basic[i][j];
    N_basic[i][n]=atoi(a.c_str());
```

MST Prim (MST 不唯一) **

```
#define MAX_V //點的最大個數  #define INF 2147483647
int graph[MAX_V][MAX_V];
int dis[MAX_V];    //dis[i] = 從找好的 mst 到 i 的最短距離
bool used[MAX_V];  //紀錄點 i 有沒有找過了
int Vnum;          //點的個數
int prim(){
    fill(dis, dis+Vnum, INF);    //#include<algorithm>
    fill(used, used+Vnum, false);
    int res = 0;
    dis[0] = 0;
    while(true) {
        // find min dis
        int v = -1;
        for(int u = 0; u < V; ++u)
          if(!used[u] && (v == -1)|| dis[u] < dis[v]) v = u;
         if(v == -1)  break;
         used[v] = true;
         res += dis[v];
         //update dis
         for(int u = 0; u < Vnum; ++u) dis[u] = min(dis[u],
graph[v][u]);
    }
}
```

MST Kruskal (MST 不唯一)

```
/*存邊*/
struct ar{int a,b,dis;}ar[M];  //N points, M edges
int Find(int n){ return n==Set[n]?Set[n]:Find(Set[n]); }
int Kruskal() {
    for(int i=0; i<=N; i++) Set[i]=i; //make set
    sort(ar,ar+M);
    Que_n=0;
    for(int i=0; i<M; i++){
        int ita,itb;
        ita=Find(ar[i].a); itb=Find(ar[i].b);
        if(ita!=itb){     //different set
            ans=max(ans,ar[i].dis);  //max distance
            Que[Que_n][0]=ar[i].a; Que[Que_n++][1]=ar[i].b;
            Set[ita]=itb;  //put Set the same
        }
    }
    printf("%d\n%d\n",ans,Que_n); //max_dis, road_num, each_road
    for(int i=0; i<Que_n; i++) printf("%d
%d\n",Que[i][0],Que[i][1]);
}
/* dij 二維陣列 */
bool Jud[105]; //是否可以拜訪
int Que[105];  //記錄放進去的點
void Kruskal(){
    Que[Que_n++]=0; Jud[0]=false;
    while(Que_n<n) ans+=FindMin();  //total distance
}
double FindMin(){
    double Mmin=1e9; int Back=-1;
    for(int i=0; i<Que_n; i++){
        int I=Que[i];
        for(int j=0; j<n; j++){
            if(j==I)continue;
            if(Jud[j] && Dis[I][j]<Mmin){
                Que[Que_n]=j; Back=j;
                Mmin=Dis[I][j];
            }
        }
    }
    Jud[Back]=false; Que_n++;
    return Mmin;
}
```

Trie

```
struct Trie{
    int Child[26], isWord;
}Trie[1000000];
```

LCS

LIS

```
/*找個數*/
int LIS(int N){  //回傳最長個數
    int n=0,a,tmp[N];
    for(int i=0; i<N; i++){
        scanf("%d",&a);
        if(!n) tmp[n++]=a;  //如果陣列裡沒東西
        else if(a>tmp[n-1]) tmp[n++]=a;
        else *lower_bound(tmp,tmp+n,a)=a;
    }
    return n;
}
/*先全部存，找最長數列各個數*/
void LIS(list){ //list 存放全部數
    up[0] = list[0];
    pos[0]=0;
    m = 1;
    for(i=1;i<top;i++){
        t = lower_bound(up,up+m,list[i])-up;
        up[t] = list[i];
        pos[i]=t;
        m = t+1>m?t+1:m;
    }  //此時 m 為最長個數 (答案)
    m--;
    int k=0;
    for(i=top-1;i>=0;i--){
        if(pos[i]==m){
            ans[k++]=list[i];
            m--;
        }
    }
    for(i=k-1;i>=0;i--) printf("%d\n",ans[i]); //ans 為答案(倒著存)
}
```

Cut Vertex

```
class CutVertex{
    #define MAX_VERTEX 110
    bool graph[MAX_VERTEX][MAX_VERTEX];
    int dfn[MAX_VERTEX], low[MAX_VERTEX], answer[MAX_VERTEX];
    int deapth, ansc, Vnum, Enum;
    void dfnlow(int u, int v){
        /*--u is the visiting point, v is the u's parent and w
            is u's child.--*/
        int w;
        bool yes=0;
        int child=0;
        dfn[u]=low[u]=deapth++;
        for(w=0;w<MAX_VERTEX;w++)
```

```
            if(graph[u][w]){
                if(dfn[w]<0) {    //w isn't visited.
                    dfnlow(w,u);
                    child++;
                    if(dfn[u]<=low[w])yes=1;
                    low[u]=(low[u]<low[w])?low[u]:low[w];
                }
                else if(w!=v) //Back edge
                    low[u]=(low[u]<dfn[w])?low[u]:dfn[w];
            }
        if((child>1||v>=0)&&yes)answer[ansc++]=u;
    }
    void initialization(){
        int i;
        memset(graph,0,MAX_VERTEX*MAX_VERTEX*sizeof(bool));
        ansc = deapth = 0;
        for(i=0;i<MAX_VERTEX;i++) dfn[i]=low[i]=-1;
    }
    void FindCutVertex(){
        for(int i=0;i<MAX_VERTEX;i++)
            if(dfn[i]==-1) dfnlow(i,-1);
    }
};
```

Bipartite Matching

```
int mx[505],my[505];   //memset -1
bool vx[505];
int Matching(){
  for(int i=0; i<Y; i++)  //Y 左邊  X 右邊  (兩排做 mathcing)
      if(my[i]==-1){
          memset(vx,false,sizeof(vx));
          if(Aug(i)) match++;
      }
  return match;
}
bool Aug(int y){ //Augmenting Path 擴充路徑
    for(int j=0; j<X; j++)
        if(ar[y][j] && !vx[j]){
            vx[j]=true;
            if(mx[j]==-1 || Aug(mx[j])){
                mx[j]=y;
                my[y]=j;
                return true;
            }
        }
    return false;
}
```

Multiple Bipartite Matching

```
bool Matching(int Max){
    for(int i=0; i<N; i++){
            memset(vx,false,sizeof(vx));
            if(!Augument(i,Max)) return false;
    }
    return true;
}

bool Augument(int y,int Max){
    for(int x=first[y]; x!=-1; x=e[x].next){
        int i=e[x].v;
        if(!vx[i]){
            vx[i]=true;
            if(peo[i]<Max){
                Mx[i][peo[i]++]=y;
                return true;
            }
            for(int k=0; k<peo[i]; k++)
                if(Augument(Mx[i][k],Max)){
                    Mx[i][k]=y;
                    return true;
                }
        }
    }
    return false;
}
```

Binary Indexed Tree (get sum)

```
void ini(){ memset(tree_sum, 0, sizeof(tree_sum)); }
int low(int in){ return in&(-in); }
void change(int x, int d){
    for(; x<MAXN; x+=low(x)) tree_sum[x] += d;
}
int getsum(int x){
    int s = 0;
    for(; x>=1; x-=low(x)) s += tree_sum[x];
    return s;
}
```

KMP

```
/* string A compare B (LenA<=LenB), pi for A */
LenA=strlen(A); LenB=strlen(B); ans=0; pi[0]=-1;
for(int i=1,k=-1; i<LenA; ++i){
    if(k>=0 && A[i]!=A[k+1]) k=-1;
    if(A[i]==A[k+1]) ++k;
    pi[i]=k;
}
for(int i=0,k=-1; i<LenB; ++i){
    while(k>=0 && B[i]!=A[k+1]) k=pi[k];
    if(B[i]==A[k+1]) ++k;
    if(k+1==LenA){ ++ans; k=pi[k];}
}
printf("%d\n",ans);
```

Segment tree

```
/* Note the index of the first is from 1. */
int NODE [3*MAXN];
void create(int id, int beg, int end){
     if(beg==end){ /* For the leaf */
          NODE[id] = 1;     //user defined cost for leaf
          return;
     }
     int mid = (beg+end) >> 1;
     create(2*id, beg, mid);
     create(2*id+1, mid+1, end);
     NODE[id] = max(NODE[2*id], NODE[2*id+1]);
}
int find_interval(int id, int beg, int end, int ita, int itb){
     if( ita>end || itb<beg ) return 0;  /* invalid condition */
     if( ita<=beg && end<=itb ) return NODE[id]; /* for all cover */
     if(beg==end) return 1;     // for the leaf, cost is user-defined
          int lf, rf, mid;
     mid = (beg+end) >>1;
     lf = find_interval(2*id, beg, mid, ita, itb);
     rf = find_interval(2*id+1, mid+1, end, ita, itb);
     return max(lf, rf);
}
```

Floyd-warshall

```
void floyd_warshall(){
    for(int k = 0; k < Vnum; ++k)
        for(int i = 0; i < Vnum; ++i)
            for(int j = 0; j < Vnum; ++j)
                dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
            /* dis[i][j]|=((dis[i][k] & dis[k][j]) || dis[i][j]); */
}
```

Bellman Ford

```
int Dis[550];
for(int i=1; i<=N; i++) Dis[i]=1e9; Dis[1]=0; //從 1~N //找到 1 的距離
for(int k=(N==1?0:1) ; k<N; k++){  //找每個邊 N-1 次
     for(int i=0; i<Edge_n; i++){
          int v1=Edge[i].v1, v2=Edge[i].v2, dis=Edge[i].dis;
          if(Dis[v2]>Dis[v1]+dis) Dis[v2]=Dis[v1]+dis;
     }
}
for(int i=0; i<Edge_n; i++){  //找是否有負環  (沒需求的話以下省略)
     int v1=Edge[i].v1, v2=Edge[i].v2, dis=Edge[i].dis;
     if(Dis[v2]>Dis[v1]+dis){
          Dis[v2]=Dis[v1]+dis;
          Ans=true; break;    //找到負環 (Ans 判斷是否為負環)
     }
}
```

SPFA (edge, dij)

```
bool Visit[20010];  //memset false
int Dis[20010];     //all 1e9 except the beginning=0
int first[20010];   //memset -1
struct edge{ int a,b,w,next; }e[Max];
//read the data
e[en].a=a; e[en].b=b; e[en].w=w; e[en].next=first[a]; first[a]=en++;
e[en].a=b; e[en].b=a; e[en].w=w; e[en].next=first[b]; first[b]=en++;
queue <int> Q;  Q.push(S); Dis[S]=0;  //S: start point
while(!Q.empty()){
     int a=Q.front(); Q.pop();
     Visit[a]=false;
     /*矩陣 for(int i=0; i<n; i++) find[S][i] distance*/
     for(int i=first[a]; i!=-1; i=e[i].next){
          int b=e[i].b;
          if(Dis[b]>e[i].w+Dis[a]){
               Dis[b]=Dis[a]+e[i].w;
               if(!Visit[b]){
                    Visit[b]=true;
                    Q.push(b);
               }
          }
     }
}
```

Dijkstra

```
#define INF 2147483647
int dis[MAX_V];
int graph[MAX_V][MAX_V];
typedef struct p{
     int v;
     int l;
     p(int v_, int l_) { v = v_; l = l_; }
}
class cmp{
     public:
     bool operator() (const P& lhs, const P& rhs) {
         return lhs.l > rhs.l;
     }
}
void dijkstra(int s) {
     fill(dis, dis+Vnum, INF);    //#include<algorithm>
     dis[s] = 0;
     priority_queue<P, vector<P>, cmp > pq;
//#include<vector><queue>
     pq.push(P(s, 0));
     used[s] = true;
     while(!pq.empty()) {
         P x = pq.top(); pq.pop();
         int v = x.v;
         if(dis[v] < x.l)  continue;
         for(int i = 0; i < Vnum; ++i) {
             if(dis[i] > dis[v] + graph[v][i]) {
                 dis[i] = dis[v] + graph[v][i];
                 pq.push(P(i, dis[i]));
             }
         }
     }
}
```

SPFA

```
#define MAX_V        //點最大的個數
#define INF 2147483547

int graph[MAX_V][MAX_V];
bool inque[MAX_V];    //點 i 是否在 queue 裡
int dis[MAX_V];       //dis[i] = 起點到點 i 最短距離
int ct[MAX_V];        //計算點 i 進入 queue 幾次了
int Vnum;

//false -> negative cycle
bool SPFA(int s) {
     fill(dis, dis+Vnum, INF);
     fill(ct, ct+Vnum, 0);
     dis[s] = 0;
     queue<int> q;
     q.push(s);
     inque[s] = true;
     ct[s]++;
     while(!q.empty()) {
         int u = q.front(); q.pop();
         inque[u] = false;
         for(int i = 0; i < Vnum; ++i) {
             if(dis[i] > dis[u] + graph[u][i]) {
                 dis[i] = dis[u] + graph[u][i];
                 if(!inque[i]) {
                     ct[i]++;
                     if(ct[i] >= Vnum)
                         return false;
                     inque[i] = true;
                     q.push(i);
                 }
             }
         }
     }
     return true;
}
```

MaxFlow BFS

```
#define MAX_V      //點的個數最大值
int graph[MAX_V][MAX_V];
bool used[MAX_V];
int prev[MAX_V];
int Vnum;
void add_edge(int u, int v, int c){
     graph[u][v] = c;
}
int dfs(int v, int sink, int flow, int Vnum){
    if(v == sink) return flow;
    used[v] = true;
    for(int i = 0; i < Vnum; ++i){
        if(!used[i] && graph[v][i] > 0){
            int d = dfs(i, sink, min(graph[v][i], flow), Vnum);
            if(d > 0){
                graph[v][i] -= d;
                graph[i][v] += d;
                return d;
            }
        }
    }
}
//s:source, t:sink, return maxflow
int max_flow(int s, int t){
     int flow = 0;
     for(;;){
          fill(used, used+Vnum, false);
          int f = dfs(s, t, INF, Vnum);
          if( f == 0)    return flow;
          else flow += f;
     }
}
```

MaxFlow dinic

```
typedef struct edge{
    int v;
    int c;
    int rev;
    edge(int v_, int c_, int rev_){
        v = v_;  c = c_;  rev = rev_;
    }
}E;
vector<E> graph[MAX_V];
int level[MAX_V];
void add_edge(int u, int v, int c){
    graph[u].push_back(edge(v, c, graph[v].size()));
    graph[v].push_back(edge(u, 0, graph[u].size()-1));
}
void bfs(int s){
    fill(level, level+Vnum, -1);
    queue<int> que;
    que.push(s);
    while(!que.empty()){
        int v = que.front(); que.pop();
        for(int i = 0; i < graph[v].size(); i++){
            edge &e = graph[v][i];
            if(e.c > 0 && level[v] < 0){
                level[e.v] = level[v] + 1;
                que.push(e.v);
            }
        }
    }
}
int dfs(int u, int t, int f){
    if(u == t)  return f;
    for(int i = 0; i < graph[v].size(); ++i){
        edge &e = graph[v][i];
        if(e.c > 0 && level[v] < level[e.v]){
            int d = dfs(e.v, t, min(f, e.c));
            if(d > 0){
                e.c -= d;
                graph[e.v][e.rev].c += d;
                return d;
            }
        }
    }
    return 0;
}
int max_flow(int s, int t){
    int flow = 0;
    for(;;){
        bfs(s);
        if(level[t] < 0) return flow;
        int f;
        while((f = dfs(s, t, INF)) > 0)
            flow += f;
    }
}
```

Min cost Max Flow

```
#define INF 2147483647
typedef pair<int, int> P;
typedef struct edge{
     int to;
     int cap;
     int cost;
     int rev;
}E;

int Vnum;
vector<E> graph[MAX_V];
```

```
int h[MAX_V];
int dis[MAX_V];
int prevv[MAX_V];
int preve[MAX_V];

void add_edge(int from, int to, int cap, int cost) {
     graph[from].push_back((E){ to, cap, cost, graph[to].size()});
     graph[to].push_back((E){ from, 0, -cost, graph[from].size()-
1});
}

int min_cost_flow(int s, int t, int f) {
     int result = 0;
    fill(h, h+V, 0);
    while(f > 0) {
        priority_queue<P, vector<P>, greater<P> > que;
        fill(dis, dis+Vnum, INF);
        dis[s] = 0;
        que.push(P(0, s));
        while(!que.empty()) {
           P p = que.top(); que.pop();
           int v = p.second;
           if(dis[v] < p.first) continue;
           for(int i = 0; i < graph[v].size(); ++i) {
             edge &e = graph[v][i];
             if(e.cap > 0 && dis[e.to] > dis[v] + e.cost + h[v] -
h[e.to]){
                prevv[e.to] = v;
                preve[e.to] = i;
                que.push(P(dis[e.to], e.to));
             }
           }
        }
        if(dis[t] == INF) {
            return -1;
        }
        for(int v = 0; v < Vnum; ++v) h[v] += dis[v];
        int d = f;
        for(int v = t; v != s; v = prevv[v]) {
             d = min(d, graph[prevv[v]][preve[v]].cap);
        }
        f -= d;
        result += d * h[t];
        for(int v = t; v!= s; v= prevv[v]) {
            edge &e = graph[prevv[v]][preve[v]];
            e.cap -= d;
            graph[v][e.rev].cap += d;
        }
    }
    return result;
}
```

Segment Tree

```
#define INF 2147483647
#define MAX_N 1<<17     //區段最長可以的個數
int n;                  //區段個數
int dat[2 * MAX_N -1]; //線段樹

void init(int n_){
    n = 1;
    while(n < n_) n *= 2;
    for(int i = 0; i < 2 * n - 1; ++i)  dat[i] = INF;
}

//將第 k 個值變更成 a   k 值的算法是 0~n-1
void update(int k, int a) {
     k += n-1;
     while(k > 0) {
          k = (k-1) / 2;
          dat[k] = min(dat[k*2+1], dat[k*2+2]);
          // #include<algorithm>
     }
}

//求[a,b]的最小值  k 是節點編號 l,r 表示 k 對應[l,r]
//從外面呼叫要用 query(a, b, 0, 0, n)
int query(int a, int b, int k, int l, int r) {
     //如果[a,b] and [l,r] 沒有交錯 回傳 INF
    if(r <= a || b <= l)  return INF;
    //如果[a,b]完全涵蓋[l,r]的話 回傳此節點的值
    if(a <= l && r <= b)  return dat[k];
    else {
     //否則 回傳兩個子節點的最小值
     int vl = query(a, b, k*2+1, l, (l+r)/2);
     int vr = query(a, b, k*2+2, (l+r)/2, r);
     return min(vl, vr);
    }
}
```

Binary Index Tree

```
//區段[1, n]
#define MAX_N     //區段最長可以的個數
int bit[MAX_N+1];
int n;            //區段個數
//計算[1, i]區段和
int sum(int i){
    int s = 0;
    while(i > 0) {
        s += bit[i];
        i -= i & -i;
    }
    return s;
}


//把區段第 i 個值加上 x
void add(int i, int x) {
    while(i <= n) {
        bit[i] += x;
        i += i & -i;
    }
}
```

Bignumber

```
struct BigNumber
{
    int array[1000];    // 一個欄位存一個數字，可以存 1000 位數
    bool sign;          // 正負號
    int length;         // 位數
};

void print(int a[100])
{
    int i = 100 - 1;            // 要印的數字位置
    while (a[i] == 0) i--;      // 數字開頭的零都不印
    while (i >= 0) cout << a[i--];
}

// a > b
bool largerthan(int a[100], int b[100])
{
    for (int i=100-1; i>=0; i--)    // 從高位數開始比，對應的位數相比
較。
        if (a[i] != b[i])       // 發現 a b 不一樣大，馬上回傳結果。
            return a[i] > b[i];
    return false;       // 完全相等
}

void mul(int a[100], int b[100], int c[100])
{
    for (int i=0; i<100; i++)
        c[i] = 0;

    for (int i=0; i<100; i++)
        for (int j=0; j<100; j++)
            if (i+j < 100)
                c[i+j] += a[i] * b[j];

    for (int i=0; i<100-1; i++) // 一口氣進位
    {
        c[i+1] += c[i] / 10;
        c[i] %= 10;
    }
}

void mul(int a[100], int b, int c[100])
{
    for (int i=0; i<100; i++)
        c[i] = a[i] * b;

    for (int i=0; i<100-1; i++) // 一口氣進位
    {
        c[i+1] += c[i] / 10;
        c[i] %= 10;
    }
}


void div(int a[100], int b[100], int c[100])
{
    int t[100];

    for (int i=100-1; i>=0; i--)
        for (int k=9; k>0; k--) // 嘗試商數
        {
            mul(b+i, k, t);
            if (largerthan(a+i, t))
            {
                sub(a+i, t, c+i);
                break;
            }
        }
}

void div(int a[100], int b, int c[100])
{
    int r = 0;
    for (int i=100-1; i>=0; i--)
    {
        r = r * 10 + a[i];
        c[i] = r / b;
        r %= b;
    }
}
```

計算幾何

```
//線與線段交點
//p1,p2 為線段  p3,p4 為線
bool intersect(Point p1,Point p2,Point p3,Point p4)
{
    double d1,d2;
    d1=(p1.x-p3.x)*(p4.y-p3.y)-(p4.x-p3.x)*(p1.y-p3.y);
    d2=(p2.x-p3.x)*(p4.y-p3.y)-(p4.x-p3.x)*(p2.y-p3.y);
    if(d1*d2<0.0)
        return true;
    else
    {
        if(!d1 || !d2)
            return true;
        return false;
    }
}

//兩線求交點
bool On_Segment(Point pi,Point pj,Point pk)
{
if(min(pi.x,pj.x)<=pk.x&&pk.x<=max(pi.x,pj.x)&&(min(pi.y,pj.y)<=pk.y
&&pk.y<=max(pi.y,pj.y)))
        return true;
    else
        return false;
}
bool intersect(Point p1,Point p2,Point p3,Point p4)
{
    double d1,d2,d3,d4;
    d1=(p1.x-p3.x)*(p4.y-p3.y)-(p4.x-p3.x)*(p1.y-p3.y);
    d2=(p2.x-p3.x)*(p4.y-p3.y)-(p4.x-p3.x)*(p2.y-p3.y);
    d3=(p3.x-p1.x)*(p2.y-p1.y)-(p2.x-p1.x)*(p3.y-p1.y);
    d4=(p4.x-p1.x)*(p2.y-p1.y)-(p2.x-p1.x)*(p4.y-p1.y);
    if(d1*d2<0&&d3*d4<0)
        return true;
    else
    {
        if(!d1&&On_Segment(p3,p4,p1))
            return true;
        if(!d2&&On_Segment(p3,p4,p2))
            return true;
        if(!d3&&On_Segment(p1,p2,p3))
            return true;
        if(!d4&&On_Segment(p1,p2,p4))
            return true;
        return false;
    }
}
```

Convex Hull

```
#define MAXN 100005
// P 為平面上散佈的點。設定為點。
// CH 為凸包上的頂點。設定為逆時針方向排列。可以視作一個 stack。
struct Point {int x, y;} P[MAXN], CH[MAXN*2];
// 向量 OA 外積向量 OB。大於零表示從 OA 到 OB 為逆時針旋轉。
int cross(const Point& o, const Point& a, const Point& b)
{
    return (a.x - o.x) * (b.y - o.y) - (a.y - o.y) * (b.x - o.x);
}
// 小於。依座標大小排序，先排 x 再排 y。
bool compare(const Point& a, const Point& b)
{
    return (a.x < b.x) || (a.x == b.x && a.y < b.y);
}
int main()
{
    int n,i,nn,t,x,y,k;
    char c[2];
    scanf("%d",&t);
    while(t--)
    {
        scanf("%d",&nn);
        n=0;
        for(i=0;i<nn;++i)
        {
            scanf("%d%d%s",&x,&y,c);
            if(c[0]=='Y')
            {
                P[n].x=x;
                P[n].y=y;
                n++;
            }
        }
        // 將所有點依照座標大小排序
        sort(P, P+n, compare);
        int m = 0;  // m 為凸包頂點數目
        // 包下半部
        for (i=0; i<n; ++i)
        {
            while (m >= 2 && cross(CH[m-2], CH[m-1], P[i]) < 0) m--;
            CH[m++] = P[i];
        }
        // 包上半部，不用再包入方才包過的終點，但會再包一次起點
        for (i=n-2,k=m+1; i>=0; --i)
        {
            while (m >= k && cross(CH[m-2], CH[m-1], P[i]) < 0) m--;
            CH[m++] = P[i];
        }
        m--;    // 最後一個點是重複出現兩次的起點，故要減一。
        printf("%d\n",m);
        for(i=0;i<m;++i)
        {
            printf("%d %d\n",CH[i].x,CH[i].y);
        }
    }
    return 0;
}
```

半平面交求核

```
//此題為順時針
#define MAXN 3000    //點為最大時
#define eps 1e-5
struct Point
{
    double x,y;
};
struct Center
{
    int n;          //點的數量
    Point p[MAXN];
}tem;
struct Line
{
    double a,b,c;
};
double cross(Point p0,Point p1,Point p2)
{
    return (p1.x-p0.x)*(p2.y-p0.y)-(p1.y-p0.y)*(p2.x-p0.x);
}
void getl(Point p1,Point p2,Line &l){    //得到線段的係數
    l.a=p1.y-p2.y;
    l.b=p2.x-p1.x;
    l.c=(p1.x-p2.x)*p1.y-(p1.y-p2.y)*p1.x;
}
void getp(Line l1,Line l2,Point &pot){   //得到交點
    double bse;
    bse=l1.a*l2.b-l1.b*l2.a;
    pot.x=(l2.c*l1.b-l2.b*l1.c)/bse;  //x
    pot.y=(l1.c*l2.a-l1.a*l2.c)/bse;   //y
}
int equal(Point p1,Point p2){      //判斷相等時精準度
    return fabs(p1.x-p2.x)<eps&&fabs(p1.y-p2.y)<eps;
}
void cut(Point p1,Point p2) {
    int i,c=0,j;
    double css1,css2;
    Center newret;
    Point pot,pre;
    Line l1,l2;
    for (i=0;i<tem.n;i++){
        css1=cross(p1,p2,tem.p[i]);
        css2=cross(p1,p2,tem.p[i+1]);
        if (css1>eps||css2>eps){ //or >=eps //當有點在逆時針方向
```

```
         if (css1*css2<=0.0){//or eps//一點在順一點在逆時
                //考慮多點在一直線上時<=0.0
           getl(p1,p2,l1);    //得到 l1 的係數
           getl(tem.p[i],tem.p[i+1],l2);//得到 l2 的係數
                getp(l1,l2,pot);   //得到交點
                if (css1<eps){ //or0.0  當 p[i]在順時針方向時  newret.p
要照順序考慮多點在一直線上時<=0.0
                    newret.p[c++]=tem.p[i];
                    newret.p[c++]=pot;
                }
                else {
                    newret.p[c++]=pot;
                    newret.p[c++]=tem.p[i+1];
                }
           }
        }
        else
        {
            newret.p[c++]=tem.p[i];
            newret.p[c++]=tem.p[i+1];
        }
    }
    //if(c==0) return false;
    j=1;
    tem.p[0]=pre=newret.p[0];
    for (i=1;i<c;i++){      //把相等的點排除
        if(!equal(newret.p[i-1],newret.p[i]))
            tem.p[j++]=newret.p[i];
    }
    if(equal(tem.p[j-1],tem.p[0]))   //考慮頭尾
        j--;
    tem.p[j]=tem.p[0];
    tem.n=j;
    //return true;
}
double area(Center tem){   //計算面積
    double S=0;
    for (int i=0;i<tem.n;i++)
        S+=tem.p[i].x*tem.p[i+1].y-tem.p[i].y*tem.p[i+1].x;
    return fabs(S/2);
}
int main()
{
    int T,N,i;
    Center ret;    //輸入的測資
    scanf("%d",&T);
    while (T--){
        scanf("%d",&N);
        for (i=0;i<N;i++)
            scanf("%lf%lf",&ret.p[i].x,&ret.p[i].y);
        ret.n=N;
        ret.p[N]=ret.p[0];
        tem=ret;
        for (i=0;i<N;i++)
            cut(ret.p[i],ret.p[i+1]);
        printf("%.2lf\n",area(tem));
    }
    return 0;
}
```

求矩形重合面積

```
#define MAXN 1005
int p[MAXN*4];
struct Line{
    int x1,y1,x2;
    bool ud;
    bool operator<(const Line& t) const{
        return y1 > t.y1;
    }
};
Line line[MAXN*2];
int main(){
    int x1,x2,y1,y2,i,j,k,w,lu,ld,cnt,area,ii;
    while(scanf("%d%d%d%d",&x1,&y1,&x2,&y2),x1!=-1){
        bool P[50005]={false};
        j=0;k=0;
        line[j].x1=x1;  line[j].y1=y1;  line[j].x2=x2;
line[j].ud=false;  ++j;
        line[j].x1=x1;  line[j].y1=y2;  line[j].x2=x2;
line[j].ud=true;   ++j;
        if(!P[x1]) {p[k]=x1;  P[x1]=true;  ++k;}
        if(!P[x2]) {p[k]=x2;  P[x2]=true;  ++k;}
        while(scanf("%d%d%d%d",&x1,&y1,&x2,&y2),x1!=-1){
            line[j].x1=x1;  line[j].y1=y1;  line[j].x2=x2;
line[j].ud=false;  ++j;
            line[j].x1=x1;  line[j].y1=y2;  line[j].x2=x2;
line[j].ud=true;   ++j;
```

```
        if(!P[x1]) {p[k]=x1;  P[x1]=true;  ++k;}
        if(!P[x2]) {p[k]=x2;  P[x2]=true;  ++k;}
    }
    sort(p,p+k);
    sort(line,line+j);
    cnt=0;area=0;
    for(i=1;i<k;++i){
        w=p[i]-p[i-1];
        for(ii=0;ii<j;++ii){
         if(line[ii].x1<=p[i-1]&&line[ii].x2>=p[i]){
                if(cnt==0&&line[ii].ud==true) lu=line[ii].y1;
                if(line[ii].ud==true)
                    ++cnt;
                else --cnt;
                if(cnt==0&&line[ii].ud==false) {
                    ld=line[ii].y1;
                    area+=w*(lu-ld);
                }
            }
        }
    }
    printf("%d\n",area);
  }
  return 0;
}
```

Cut Vertex

```
#define MAXN 102
int N,cnt,dfn[MAXN],low[MAXN],ans;    //ans 是 cut vertex 數量
bool edge[MAXN][MAXN];
void DFN_LOW(int now, int parent){
     int child=0,i;    //child 數量大於為 cut vertex
     bool ye=false;    //cut vertex 條件標記
     dfn[now] = low[now] = ++cnt;    //初始化
     for(i = 1; i <= N; ++i)
          if(edge[now][i]){    //有這個邊
               if(dfn[i] == -1){    //沒被拜訪過
                    DFN_LOW(i,now);    //DFS
                    ++child;    //child 數量+1
    //如果兒子的 low 值>現在的拜訪順序,代表不能回到之前拜訪過的點
                    if(dfn[now] <= low[i])
                         //(cut edge 的條件是 dfn[low] < low[i] ,沒有等號)
                         ye = true;
               low[now] = min(low[now],low[i]);//跟 child 取小的 low 值
               }
               else if(i != parent)//有被拜訪且兒子不是父母
                    low[now] = min(low[now],dfn[i]);//跟 child 的拜訪順
序取小者
          }
     if((parent||child>1) && ye)//如果有(parent or 兩個以上的 child)
&& 兒子不能回到之前拜訪過的點
          ++ans;//cut vertex 數量+1
}
int main() {
     char str[3*MAXN],*p;
     int root,root2,i,n;
     while(scanf("%d", &N) == 1){
          getchar();//換行
          if(!N)
               break;
          memset(edge,false,sizeof(edge));
          memset(dfn,-1,sizeof(dfn));
          memset(low,-1,sizeof(low));
          cnt = 0;  ans = 0;
          while(1){
               gets(str);//一次讀一行
               if(str[0] == '0')//第一個數字是
                    break;
               for(n=0, p=strtok(str," ");p;p = strtok(NULL," ")){//未
知數量 input
                    root2 = atoi(p);//char->int
                    if(n)//邊的另一點,undirected
                         edge[root][root2] = edge[root2][root] = true;
                    else//第一個數字是邊的其中一點
                         root = root2;
                    ++n;//n 的數量+1
               }
          }
          for(i = 1; i <= N; ++i)//對每個沒被拜訪過的點 dfs
               if(dfn[i] == -1)
                    DFN_LOW(i,0);//沒 parent 傳
          printf("%d\n", ans);
     }
     return 0;
}
```

KMP

```
#include<stdio.h>
```

```
char s[1000010];
long next[1000010];
int main(){
    long i,j,len,sum=0,t,n;
    while(scanf("%d",&n)&&n){
        scanf("%s",s); sum++;
        printf("Test case #%d\n",sum);
        i=0;j=-1;next[0]=-1;
        while(i<=n){
            if(j==-1 || s[i]==s[j]){
                ++i;++j;
             // if(s[i]!=s[j])
              next[i]=j;
             // else next[i]=next[j];
             }
             else j=next[j];
             }
             for(i=2;i<=n;i++){
                 t=i-next[i];
                 if(i%t==0&&i/t>1) printf("%d %d\n",i,i/t);
             }
             printf("\n");
         }
       return 0;
}
```

Topological sort

```
bool adj[9][9]; // adjacency matrix
int ref[9];     // 紀錄圖上每一個點目前仍被多少條邊連到
void topological_sort(){
    for (int i=0; i<9; ++i) ref[i] = 0; // 初始化為 0
    // 累計圖上每一個點被幾條邊連到
    for (int i=0; i<9; ++i)
        for (int j=0; j<9; ++j)
            if (adj[i][j])
                ref[j]++;
    // 開始找出一個合理的排列順序
    for (int i=0; i<9; ++i){
        // 尋找沒有被任何邊連向的點
        int s = 0;
        while (s < 9 && ref[s] != 0) ++s;
        if (s == 9) break;  // 找不到。表示目前殘存的圖是個環。
        ref[s] = -1;       // 設為已找過（刪去 s 點）

        cout << s;         // 印出合理的排列順序的第 i 點

        // 更新 ref 的值（刪去由 s 點連出去的邊）
        for (int t=0; t<9; ++t)
            if (adj[s][t])
                ref[t]--;
    }
}
```

Strongly connected component

```
int adj[9][9];          // adjacency matrix
bool visit[9];          // DFS visit record
vector<int> finish[9];  // DFS 的離開順序
int scc[9];             // 每個點的強連通分量編號

void DFS1(int i){
    visit[i] = true;
    for (int j=0; j<9; ++j)
        if (adj[i][j] && !visit[j])
            DFS1(j);

    finish.push_back(i);
}

void DFS2(int i, int c){
    cout << "第" << c << "個強連通分量";
    cout << "包含第" << i << "點";
    scc[i] = c; // 設定第 i 點屬於第 c 個強連通分量

    visit[i] = true;
    for (int j=0; j<9; ++j)
        if (adj[j][i] && !visit[j]) // 顛倒所有邊的方向
            DFS2(j, c);
}

void kosaraju(){
    finish.clear();
    memset(visit, false, sizeof(visit));
    for (int i=0; i<9; ++i)
        if (!visit[i])
            DFS1(i);
    memset(visit, false, sizeof(visit));
    for (int i=9-1; i>=0; --i)
        if (!visit[finish[i]])       // 原圖的拓樸順序
            DFS2(finish[i], c++);    // 找到一個強連通分量
}
```

Largest square

```
#define MAXSIZE 6010
using namespace std ;
struct P{ int y , v ;};
P pt ;
int main( ){
     int m , n ;
     int i , j , k , ans , t , tx , ty ;
     int t1 , t2 ;
     scanf( "%d" , &t ) ;
     while( t-- ){
          scanf( "%d %d" , &m , &n ) ;
          vector <P> p[MAXSIZE] ;
          for( i = 0 ; i < n ; i++ ){
               scanf( "%d %d" , &tx , &ty ) ;
               pt.y = ty+1 ;
               pt.v = 1 ;
               p[tx+1].push_back( pt ) ;
          }
          ans = 1 ;
          for( i = 0 ; i <= m ; i++ ){
               for( j = 0 ; j < p[i].size( ) ; j++ ){
                    t1 = p[i][j].y ;
                    t2 = i ;
                    bool find = false ;
                    bool find2 = false ;
                    for( k = 0 ; k < p[i-1].size( ) ; k++ )
                         if(p[i-1][k].y==t1-1 && p[i-1][k+1].y == t1){
                              find = true ;
                              break ;
                         }
                    if( j > 0 && p[i][j-1].y == t1-1 )
                         find2 = true ;
                    if( find && find2 ){
                         if( p[i-1][k+1].v == p[i][j-1].v && p[i-
1][k+1].v == p[i-1][k].v )
                              p[i][j].v = p[i-1][k].v + 1 ;
                         else
                              p[i][j].v = min( p[i-1][k].v , min( p[i-
1][k+1].v , p[i][j-1].v ) )+1;
                              ans = max( p[i][j].v , ans ) ;
                    }
               }
          }
          printf( "%d\n" , ans ) ;
          for( i = 0 ; i <= m ; i++ )
               for( j = 0 ; j < p[i].size( ) ; j++ )
                    if( p[i][j].v == ans )
                         printf( "%d %d\n" , i - ans , p[i][j].y -
ans ) ;
     }
     return 0 ;
}
```

Coin Change have limit NM

```
int coin[ 11 ] ;
int dp[ 120010 ], num[ 11 ] ;
int total, n, m, x;
int main(){
     int i, j, k, left;
     while( ~scanf( "%d", &n ) ){
          for( i = 1; i <= n; i++ )
               scanf( "%d", &coin[ i ] );
          total = 0;
          for( i = 1; i <= n; i++ ){
               scanf( "%d", &num[ i ] );
               total += num[ i ] * coin[ i ];
          }
          memset( dp, 0, sizeof( dp ) );
          dp[ 0 ] = 1;
          for( i = 1; i <= n; i++ )
            for( j = total - coin[ i ] ; j >= 0 ; --j )
                for( int k = 1; k <= num[ i ] && j + k * coin[ i ] <=
total ; ++k )
                    dp[ j + k * coin[ i ] ] += dp[ j ];
          for( i = 0; i <= total; i++ )
               printf( "%d: %d\n", i, dp[ i ] );
          printf( "\n" );
     }
     return 0;
}
```

Euler’s Totient Function

```
#define MAX 3000010
using namespace std ;
long long euler[ MAX ] ;
bool      prime[ MAX ] ;
void Great_Euler( ){
   //求 euler[ i ] way1
/*  for( int i = 2 ; i <= MAX ; i++ )
        euler[ i ] = 0 ;
    euler[ 1 ] = 1 ;
    for( int i = 2 ; i <= MAX ; i++ )
        if( !euler[ i ] )
            for( int j = i ; j <= n ; j += i ){
                if( !euler[ j ] )
                    euler[ j ] = j ;
                euler[ j ] = euler[ j ] / i *( i - 1 ) ;
            }
*/  //求 euler[ i ] way2
     memset( prime , true , sizeof( prime ) ) ;
     for( int i = 0 ; i < MAX ; i++ )
         euler[ i ] = i ;                       //初始話數字 i, 因數 i 個
     prime[ 2 ] = true ;
     for( int i = 2 ; i < MAX ; i++ ){
         if( prime[ i ] == true ){
             euler[ i ] = i - 1 ;               //和質數互質的個數 i-1
             for( int j = i+i ; j < MAX ; j += i ){
                 prime[ j ] = false ;              //篩法
                 euler[ j ] -= euler[ j ] / i ; //扣除公因數個數
             }
         }
     }
}
//配合 Sieve_2, prime[ i ] = 0 表示 i 為質數, k(質數)表示 i 的最小質因數
//求 euler[ i ] way3
void Seive2( ){
    int t1 = (int)sqrt( MAX ) ;
    euler[ 1 ] = 1 ;
    for( int i = 2 ; i <= t1 ; i++ )
        if( !euler[ i ] )
            for( int k = (MAX-1)/i , j = i*k ; k >= i ; k-- , j-= i )
                if( !euler[ k ] ) euler[ j ] = i ;
}
void Great_Euler2( ){
    for( int i = 2 ; i < MAX ; i++ ){
            if( euler[ i ] == 0 ) euler[ i ] = i-1 ;
            else{
                int j = i / euler[ i ] ;
                if( j%euler[i]==0 ) euler[i]=euler[i]*euler[j];
                else euler[ i ] = euler[ j ] * ( euler[ i ]-1 ) ;
            }
    }
}
int main( ){
     int n , m ;
     bool flag = false ;
     Great_Euler( ) ;
     //Seive2( ) ;
     //Great_Euler2( ) ;
     //加總動作  不做就是 Phi( k )
     for( int i = 1 ; i < MAX ; i++ ) euler[ i ] += euler[ i-1 ] ;
     scanf( "%d" , &n ) ;
     while( n-- ){
          scanf( "%d" , &m ) ;
          printf( "%I64d\n" , euler[ m ] ) ;
     }
     return 0 ;
}
/*加總尤拉函數
euler[n] = Phi(1) + Phi(2) + ... + Phi(n)
Phi(k) = 對 1 <= j <= k 有 ( j , k ) = 1 的個數
基本題型對應:
利用 phi(k), 求累積的最大公因數和
求累積總合 gcd( i , n ) ;( 1 <= i <= n-1 ) ; 不包括 n
int i , j , bit ;
for( i = 2 ; i < n ; i++ )
    a[ i ] = phi[ i ] ;     //最大公因數為 1 的數量
double k = sqrt( 0.5 + n ) ;
for( i = 2 ; i <= k ; i++ ){
    for( j = i , bit = 1 ; j < n ; j += i , bit++ ){
        // bit = j / i , i < bit 時, 不要重複算
        //j 有 bit 個 i 的倍數, 其中和 j 的 gcd 恰為 i 者有 phi[ bit ] 個
        //j 有 i 個 bit 的倍數, 其中和 j 的 gcd 恰為 bit 者有 phi[ i ] 個
        if( i < bit )  a[ j ] += phi[ bit ] * i + phi[ i ] * bit ;
        else if( i == bit ) a[ j ] += phi[ i ] * bit ;
    }
}
```

Extended Euclid's Algorithm

```
int exGCD( int a , int b , int &x , int &y ){
    if( b == 0 ){
        x = 1 ;
        y = 0 ;
        return a ;
    }
    int r = exGCD( b , a%b , x , y ) ;
    int t = x ;
    x = y ;
    y = t - a/b*y ;
    return r ;
}
int main( ){
    int a , b , x , y , GCD ;
    while( scanf( "%d%d" , &a ,&b ) != EOF )
    {
        GCD = exGCD( a , b , x , y ) ;
        printf( "%d %d %d\n" , x , y , GCD ) ;
    }
    return 0 ;
}
```

Gaussian Elimination Determinant

```
#define MAX 32
using namespace std ;
int n ;
double matrix[ MAX ][ MAX ] ;       //原係數矩陣
double     x[ MAX ] ;               //聯立線性方程式的解
void gaussian_elimination( ){
   for( int i = 0 ; i < n ; ++i ){
      // 如果上方 row 的首項係數為零，則考慮與下方 row 交換。
      if( matrix[ i ][ i ] == 0 )
         for( int j = i+1 ; j < n ; ++j )
            if( matrix[ j ][ i ] != 0 ){
               // 交換上方 row 與下方 row。
               for( int k = i ; k < n ; ++k )
                  swap( matrix[ i ][ k ] , matrix[ j ][ k ] ) ;
               break ;
            }
      if( matrix[ i ][ i ] == 0 )
         continue ;
      // 上方 row 的首項係數調整成一。目的是為了讓對角線皆為一。
      double t = matrix[ i ][ i ] ;    // 首項係數
      for( int k = i ; k < n ; ++k )
         matrix[ i ][ k ] /= t ;
      // 窮舉並消去下方 row。
      for( int j = i+1 ; j < n ; ++j )
         if( matrix[ j ][ i ] != 0 ){
            double t = matrix[ j ][ i ] ;
            for( int k = i ; k < n ; ++k )
               matrix[ j ][ k ] -= matrix[ i ][ k ] * t ;
         }
   }
}
// 使用高斯消去法求得聯立解, 注意要多 1 行
void back_substitution( ){
   for( int i = n-1 ; i >= 0 ; --i ){
      double t = 0 ;
      for( int k = i+1 ; k < n ; ++k )
         t += matrix[ i ][ k ] * x[ k ] ;
      x[ i ] = ( matrix[ i ][ n ] - t ) / matrix[ i ][ i ] ;
   }
}
/*
//求得矩陣 Determinant
//注意 matrix 用 int
void rowgcd( int *a , int *b , int i ){
   if( abs( a[ i ] ) < abs( b[ i ] ) )
      swap( a , b ) ;
   while( b[ i ] != 0 ){
      int t = a[ i ] / b[ i ] ;
      for( int k = i ; k < n ; ++k )
         a[ k ] -= b[ k ] * t ;
      swap( a , b ) ;
   }
}
int determinant( ){
   int det = 1 ;
   for( int i = 0 ; i < n ; ++i ){
      for( int j = i+1 ; j < n ; ++j ){
         // 輾轉相除法進行消去。
         rowgcd( matrix[ i ], matrix[ j ] , i ) ;
         // 把非零的 row，挪到當前的 row。
         if( matrix[ i ][ i ] == 0 && matrix[ j ][ i ] != 0 ){
            for( int k = i ; k < n ; ++k )
               swap( matrix[ i ][ k ] , matrix[ j ][ k ] ) ;
            // 兩行交換，determinant 會相差一負號。
```

```
                det *= -1 ;
            }
        }
        // determinant 等於上三角方陣的對角線乘積。
        det *= matrix[ i ][ i ] ;
        if( det == 0 )
            break ;
    }
    return det ;
}*/
int main( ){
    while( ~scanf( "%d" , &n ) ){
        if( n == 0 )
            break ;
        for( int i = 0 ; i < n ; i++ )
            for( int j = 0 ; j < n ; j++ )
                scanf( "%lf" , &matrix[ i ][ j ] ) ;
        //gaussian_elimination( ) ;
        //printf( "%d\n" , determinant( ) ) ;
        for( int i = 0 ; i < n ; i++ ){
            for( int j = 0 ; j < n ; j++ )
                printf( "%.0lf " , matrix[ i ][ j ] ) ;
            printf( "\n" ) ;
        }
    }
    printf( "*\n" ) ;
    return 0 ;
}
```

中國剩餘定理(同於方程組)

```
#define MAX 52
using namespace std ;
int a[ MAX ] ; //餘數要求
int p[ MAX ] ; //兩兩互質的質數
int n ;
int exGCD( int a , int b , int &x , int &y ){
     if( b == 0 ){
         x = 1 ;
         y = 0 ;
         return a ;
     }
     //ax + by => bx' + (a%b)y' , x' = y , y' = x - a/b*y
     int r = exGCD( b , a%b , x , y ) ;
     int temp = x ;
     x = y ;
     y = temp - a/b*y ;
     return r ;
}
int lmes( ){
     int M1 = 1 , M2 , x , y , sum = 0 ;
     for( int i = 0 ; i < n ; i++ )
         M1 *= p[ i ] ;
     for( int i = 0 ; i < n ; i++ ){
         M2  = M1 / p[ i ] ;
         exGCD( M2 , p[ i ] , x , y ) ;
         sum += ( a[ i ] * M2 * ( x % p[ i ] ) ) % M1 ;//notice
     }
     return ( sum + M1 ) % M1 ;
}
int main( ){
     int ans , t = 1 ;
     while( ~scanf( "%d" , &n ) ){
         for( int i = 0 ; i < n ; i++ )
             scanf( "%d" , &a[i] ) ;
         for( int i = 0 ; i < n ; i++ )
             scanf( "%d" , &p[i] ) ;
         ans = lmes( ) ;
         printf( "Case %d: X = %d\n" , t++ , ans ) ;
     }
     return 0 ;
}
/*中國剩餘定理(同餘方程組)
求 x 滿足  x+d mod 23 = p , x+d mod 28 = e , x+d mod 33 = i
使用擴展歐基理德 求得 (28*33)x' + 23y' = 1 的 x'一解  進行調整倍數
再加總 3 項要求 在減去 d
sol:
28 * 33 % 23 = 4 ,4 * 6 % 23 = 1 ,28 * 33 * 6   = 5544   ( p 倍化 )
23 * 33 % 28 = 3 ,3 * 19 % 28 = 1 ,23 * 33 * 19 = 14421   ( e 倍化 )
23 * 28 % 33 = 17 ,17 * 2 % 33 = 1 ,23 * 28 * 2  = 1288   ( i 倍化 )
```

計算方程式

```
#define Eps 1e-7
int p,q,r,s,t,u;
double f(double x){
     return p*exp(-x)+q*sin(x)+r*cos(x)+s*tan(x)+t*x*x+u;
}
double FindSol(){
    double L=0,R=1,mid;
    while(L+Eps<R){
        mid=(L+R)/(double)2.0;
        if(f(L)*f(mid)<=0)
            R=mid;
        else
            L=mid;
    }
    return mid;
}
int main(){
    while(scanf("%d%d%d%d%d%d",&p,&q,&r,&s,&t,&u)!=EOF){
        if(f(0)*f(1)>0)
            printf("No solution\n");
    //  else if(f(0)*f(1)==0)printf("0.0000\n");
        else
            printf("%.4lf\n",FindSol());
    }
    return 0;
}
```

Map

```
#include <map>
map <string,int> mymap;
map <string,int>::iterator it;
string a;
it=mymap.find(a);
if(it==mymap.end()) mymap.insert(pair<string,int>(a,100)); //not
find
else int n=(*it).second;
else string s=(*it).first;
```

Max flow

```
int FordFulkerson(int s, int t){
    int ret = 0;
    while (1) {
        memset(visited, 0, sizeof(visited));
        if (!DFS(s, t))
        break;
        ret += findFlow(s, t);
    }
    return ret;
}
```

```
bool DFS(int k, int t) {
    visited[k] = 1;
    if (now == t) return 1;
    for (int i = 0; i < n; ++i ) {
        if (visited[i]) continue;
        if (cap[k][i]-flow[k][i]>0 || flow[i][k]>0) {
            path[i] = k;
            if (DFS(i, t))
            return 1;
        }
    }
    return 0;
}
int findFlow(int s, int t) {
    int f = INF;
    for (int i = t, pre; i != s; i = pre) {
        pre = path[i];
        if (cap[pre][i] - flow[pre][i] > 0)
            f = min(f, cap[pre][i] - flow[pre][i]);
        else
            f = min(f, flow[i][pre]);
    }
    for (int i = t, pre; i != s; i = pre) {
        pre = path[i];
        if (cap[pre][i] - flow[pre][i] > 0)
        flow[pre][i] += f;
        else flow[i][pre] -= f;
    }
    return f;
}
```

Max flow

```
#include <cstdio>
#include <cstring>
#include <stack>
using namespace std;
#define MAXN 105
#define INF 0x3f3f3f3f
#define min(x,y) (x<y?x:y)

int cap[MAXN][MAXN];
int Ford_Fulkerson(int, int);

int main(){
    int n, np, nc, m, a, b, c, ans;
    while(scanf("%d %d %d %d", &n, &np, &nc, &m)!=EOF){
        memset(cap, 0, sizeof(cap));
        for(int i=0;i<m;++i){
            scanf(" (%d,%d)%d ", &a, &b, &c);
            cap[a][b] = c;
        }
        for(int i=0;i<np;++i){
            scanf(" (%d)%d ", &a, &c);
            cap[n][a] = c;
        }
        for(int i=0;i<nc;++i){
            scanf(" (%d)%d ", &a, &c);
            cap[a][n+1] = c;
        }
        ans = Ford_Fulkerson(n, n+1);
        printf("%d\n", ans);
    }
}

int Ford_Fulkerson(int start, int terminal){
    stack<int> sta;
    int path[MAXN], flow[MAXN][MAXN]={0}, tmp, limit, ans=0;
    bool visited[MAXN];
    while(1){
        memset(visited, false, sizeof(visited));
        sta.push(start), visited[start] = true;
        while(!sta.empty()){
            tmp = sta.top(), sta.pop();
            for(int i=0;i<=terminal;++i){
                if(!visited[i] && ((cap[tmp][i]-flow[tmp][i])>0 ||
flow[i][tmp]>0)){
                    sta.push(i), visited[i] = true;
                    if((cap[tmp][i]-flow[tmp][i])>0)
                        path[i] = tmp;
                    else
                        path[i] = -tmp;
                }
            }
        }
        if(!visited[terminal])    return ans;
        limit = INF;
        for(int i=terminal;i!=start;){
            if(path[i]>=0){
                limit = min(limit, cap[path[i]][i]-
flow[path[i]][i]);
                i = path[i];
            }
            else{
                limit = min(limit, flow[i][-path[i]]);
                i = -path[i];
            }
        }
        ans += limit;
        for(int i=terminal;i!=start;){
            if(path[i]>=0){
                flow[path[i]][i] += limit;
                i = path[i];
            }
            else{
                flow[i][-path[i]] -= limit;
                i = -path[i];
            }
        }
    }
}
```