ACM ICPC
Regional 2012

1. Generales

1.1. LIS en O(nlgn).

```
vector<int> LIS(vector<int> X){
   int n = X.size(),L = 0,M[n+1],P[n];
   int lo,hi,mi;

   L = 0;
   M[0] = 0;

   for(int i=0,j;i<n;i++){
      lo = 0; hi = L;

      while(lo!=hi){
         mi = (lo+hi+1)/2;

         if(X[M[mi]]<X[i]) lo = mi;
         else hi = mi-1;
      }

      j = lo;
      P[i] = M[j];

      if(j==L || X[i]<X[M[j+1]]){
         M[j+1] = i;
         L = max(L,j+1);
      }
   }

   int a[L];

   for(int i=L-1,j=M[L];i>=0;i--){
      a[i] = X[j];
      j = P[j];
   }

   return vector<int>(a,a+L);
}
```

1.2. Problema de Josephus.

```
int survivor(int n, int m){
  for (int s=0,i=1;i<=n;++i) s = (s+m)%i;
  return (s+1);
}
```

1.3. Contar inversiones.

```
#define MAX_SIZE 100000

int A[MAX_SIZE],C[MAX_SIZE],pos1,pos2,sz;

long long countInversions(int a, int b){
   if(a==b) return 0;

   int c = ((a+b)>>1);
```

```
   long long aux = countInversions(a,c)+countInversions(c+1,b);
   pos1 = a; pos2 = c+1; sz = 0;

   while(pos1<=c && pos2<=b){
      if(A[pos1]<A[pos2]) C[sz] = A[pos1++];
      else{
         C[sz] = A[pos2++];
         aux += c-pos1+1;
      }
      ++sz;
   }

   if(pos1>c) memcpy(C+sz,A+pos2,(b-pos2+1)*sizeof(int));
   else memcpy(C+sz,A+pos1,(c-pos1+1)*sizeof(int));

   sz = b-a+1;
   memcpy(A+a,C,sz*sizeof(int));

   return aux;
}
```

1.4. Números dada la suma de pares.

```
bool solve(int N, int sums[], int ans[]){
   int M = N*(N-1)/2;
   multiset<int> S;
   multiset<int> :: iterator it;

   sort(sums,sums+M);

   for(int i = 2;i<M;++i){
      if((sums[0]+sums[1]-sums[i])%2!=0) continue;

      ans[0] = (sums[0]+sums[1]-sums[i])/2;
      S = multiset<int>(sums,sums+M);

      bool valid = true;

      for(int j = 1;j<N && valid;++j){
         ans[j] = (*S.begin())-ans[0];

         for(int k = 0;k<j && valid;++k){
            it = S.find(ans[k]+ans[j]);

            if(it==S.end()) valid = false;
            else S.erase(it);
         }
      }

      if(valid) return true;
   }

   return false;
}
```

2. Grafos

2.1. Ciclo de Euler.

```
// Las listas de adyacencia se deben ordenar de forma ascendente para
// obtener el ciclo lexicografico minimo deacuerdo a la numeracion
// de las aristas

#define MAX_V 44
#define MAX_E 1995

int N,deg[MAX_V],eu[MAX_E],ev[MAX_E];
list<int> G[MAX_V],L;

bool visited[MAX_V];
stack<int> S;
queue<int> Q;

bool connected(){
   int cont = 0;
   Q.push(0);
   memset(visited,false,sizeof(visited));
   visited[0] = true;
```

```
   while(!Q.empty()){
      int v = Q.front(); Q.pop();
      ++cont;

      for(list<int>::iterator it = G[v].begin();it!=G[v].end();++it){
            int e = *it;
            int w = eu[e]==v? ev[e] : eu[e];

         if(!visited[w]){
            visited[w] = true;
            Q.push(w);
         }
      }
   }

   return cont==N;
}

bool eulerian(){
   if(!connected()) return false;

   for(int v = 0;v<N;++v)
      if(deg[v]&1)
         return false;

   return true;
}

void take_edge(int v, int w){
   --deg[v]; --deg[w];
   int e = G[v].front();
   G[v].pop_front();

   for(list<int>::iterator it = G[w].begin();it!=G[w].end();++it){
      if(*it==e){
         G[w].erase(it);
         break;
      }
   }
}

void euler(int v){
   while(true){
      if(G[v].empty()) break;
      int e = G[v].front();
      int w = eu[e]==v? ev[e] : eu[e];
      S.push(e);
      take_edge(v,w);
      v = w;
   }
}

bool find_cycle(int s){
   if(!eulerian()) return false;

   int v = s,e;
   L.clear();

   do{
      euler(v);
      e = S.top(); S.pop();
      L.push_back(e);

      v = eu[e]==v? ev[e] : eu[e];
   }while(!S.empty());

   return true;
}

void print_cycle(int s){
   if(!find_cycle(s)) printf("-1\n");
   else{
      bool first = true;
      reverse(L.begin(),L.end());
      for(list<int>::iterator e = L.begin();e!=L.end();++e){
            if(!first) printf("␣");
            first = false;
         printf("%d",1+(*e));
      }
      printf("\n");
   }
}
```

2.2. Euler (Directed graph).

```
int V,E,to[32000],nxt[32000],last[1000],now[1000];
int ans[32000];

void init(){
   memset(last,-1,sizeof(last));
   E = 0;
}

void make_edge(int u, int v){
   to[E] = v; nxt[E] = last[u]; last[u] = E++;
}

// A : vertice inicial
void euler(int A){
   for(int i = 0;i < V;++i)
      now[i] = last[i];

   stack<int> S;
   S.push(A);

   int cur,sz = 0;

   while(!S.empty()){
      cur = S.top();

      if(now[cur] != -1){
         S.push(to[now[cur]]);
         now[cur] = nxt[now[cur]];
      }else{
         ans[sz++] = cur;
         S.pop();
      }
   }

   for(int i = sz - 1;i > 0;--i)
      printf("%d %d\n",ans[i] + 1,ans[i - 1] + 1)
}
```

2.3. Punto de articulación.

```
#define SZ 100
bool M[SZ][SZ];
int N,colour[SZ],dfsNum[SZ],num,pos[SZ],leastAncestor[SZ],parent[SZ];

int dfs(int u){
   int ans = 0,cont = 0,v;

   stack<int> S;
   S.push(u);

   while(!S.empty()){
      v = S.top();
      if(colour[v]==0){
         colour[v] = 1;
         dfsNum[v] = num++;
         leastAncestor[v] = num;
      }

      for(;pos[v]<N;++pos[v]){
         if(M[v][pos[v]] && pos[v]!=parent[v]){
            if(colour[pos[v]]==0){
               parent[pos[v]]=v;
               S.push(pos[v]);
               if(v==u) ++cont;
               break;
            }else leastAncestor[v]<?=dfsNum[pos[v]];
         }
      }

      if(pos[v]==N){
         colour[v] = 2;
         S.pop();

         if(v!=u) leastAncestor[parent[v]]<?=leastAncestor[v];
      }
   }

   if(cont>1){
      ++ans;
      printf("%d\n",u);
```

```
    }

    for(int i = 0;i<N;++i){
        if(i==u) continue;
        for(int j = 0;j<N;j++)
            if(M[i][j] && parent[j]==i && leastAncestor[j]>=dfsNum[i]){
                printf("%d\n",i);
                ++ans;
                break;
            }
    }

    return ans;
}
```

```
void Articulation_points(){
    memset(colour,0,sizeof(colour));
    memset(pos,0,sizeof(pos));
    memset(parent,-1,sizeof(parent));
    num = 0;

    int total = 0;
    for(int i = 0;i<N;++i) if(colour[i]==0) total += dfs(i);

    printf("#␣Articulation␣Points␣:␣%d\n",total);
}
```

2.4. Detección de puentes.

```
#define SZ 100
bool M[SZ][SZ];
int N,colour[SZ],dfsNum[SZ],num,pos[SZ],leastAncestor[SZ],parent[SZ];

void dfs(int u){
    int v;
    stack<int> S;
    S.push(u);


    while(!S.empty()){
        v = S.top();
        if(colour[v]==0){
            colour[v] = 1;
            dfsNum[v] = num++;
            leastAncestor[v] = num;
        }

        for(;pos[v]<N;++pos[v]){
            if(M[v][pos[v]] && pos[v]!=parent[v]){
                if(colour[pos[v]]==0){
                    parent[pos[v]] = v;
                    S.push(pos[v]);
                    break;
                }else leastAncestor[v] <?= dfsNum[pos[v]];
            }
        }

        if(pos[v]==N){
            colour[v] = 2;
            S.pop();

            if(v!=u) leastAncestor[parent[v]] <?= leastAncestor[v];
        }
    }
}

void Bridge_detection(){
    memset(colour,0,sizeof(colour));
    memset(pos,0,sizeof(pos));
    memset(parent,-1,sizeof(parent));
    num = 0;

    int ans = 0;

    for(int i = 0;i<N;i++) if(colour[i]==0) dfs(i);

    for(int i = 0;i<N;i++)
        for(int j = 0;j<N;j++)
            if(parent[j]==i && leastAncestor[j]>dfsNum[i]){
                printf("%d␣-␣%d\n",i,j);
                ++ans;
            }


    printf("%d␣bridges\n",ans);
}
```

2.5. Componentes biconexas (Tarjan).

```
#define MAXN 100000

int V;
vector<int> adj[MAXN];
int dfn[MAXN],low[MAXN];
vector< vector<int> > C;
stack< pair<int, int> > stk;

void cache_bc(int x, int y){
   vector<int> com;
   int tx,ty;

   do{
      tx = stk.top().first, ty = stk.top().second;
      stk.pop();
      com.push_back(tx), com.push_back(ty);
   }while(tx!=x || ty!=y);

   C.push_back(com);
}

void DFS(int cur, int prev, int number){
   dfn[cur] = low[cur] = number;

   for(int i = adj[cur].size()-1;i>=0;--i){
      int next = adj[cur][i];
      if(next==prev) continue;
      if(dfn[next]==-1){
         stk.push(make_pair(cur,next));
         DFS(next,cur,number+1);
         low[cur] = min(low[cur], low[next]);
         if(low[next]>=dfn[cur]) cache_bc(cur,next);
      }else low[cur] = min(low[cur],dfn[next]);
   }
}

void biconn_comp(){
   memset(dfn,-1,sizeof(dfn));
   C.clear();
   DFS(0,0,0);

   int comp = C.size();

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

   for(int i = 0;i < comp;++i){
      sort(C[i].begin(),C[i].end());
      C[i].erase(unique(C[i].begin(),C[i].end()),C[i].end());
      int m = C[i].size();
      for(int j = 0;j < m;++j) printf("%d ",1 + C[i][j]);
      printf("\n");
   }
}
```

2.6. DFS para calcular low iterativo.

```
#define MAXN 100001
#define MAXE 500000

int last[MAXN],nxt[2 * MAXE],to[2 * MAXE],ne = 0;

void add_edge(int &u, int &v){
   to[ne] = v; nxt[ne] = last[u]; last[u] = ne++;
   to[ne] = u; nxt[ne] = last[v]; last[v] = ne++;
}

int low[MAXN],parent[MAXN],level[MAXN],comp[MAXN];

void dfs(int r){
   int u,v;

   stack<int> S;

   S.push(r);
   comp[r] = r;
   low[r] = level[r] = 0;
   parent[r] = -1;

   while(!S.empty()){
      u = S.top();
```

```
      for(int &e = last[u];e != -1;e = nxt[e]){
         v = to[e];

         if(comp[v] != -1 && v != parent[u] && level[u] > level[v]){
            low[u] = min(low[u],level[v]);
         }else if(comp[v] == -1){
            S.push(v);
            comp[v] = r;
            low[v] = level[v] = level[u] + 1;
            parent[v] = u;
                  break;
               }
            }

            if(last[u] == -1){
               S.pop();
               if(u != r)
                  low[ parent[u] ] = min(low[ parent[u] ],low[u]);
            }
         }
      }
```

2.7. Componentes fuertemente conexas (Tarjan).

```
#define MAX_V 100000

vector<int> L[MAX_V],C[MAX_V];
int V,dfsPos,dfsNum[MAX_V],lowlink[MAX_V],num_scc,comp[MAX_V];
bool in_stack[MAX_V];
stack<int> S;

void tarjan(int v){
   dfsNum[v] = lowlink[v] = dfsPos++;
   S.push(v); in_stack[v] = true;

   for(int i = L[v].size()-1;i>=0;--i){
      int w = L[v][i];
      if(dfsNum[w]==-1){
         tarjan(w);
         lowlink[v] = min(lowlink[v],lowlink[w]);
      }else if(in_stack[w]) lowlink[v] = min(lowlink[v], lowlink[w]);
   }

   if(dfsNum[v]==lowlink[v]){
      vector<int> &com = C[num_scc];
      com.clear();
      int aux;

      do{
         aux = S.top(); S.pop();
         comp[aux] = num_scc;
         com.push_back(aux);
         in_stack[aux] = false;
      }while(aux!=v);

      ++num_scc;
   }
}

void build_scc(int _V){
   V = _V;
   memset(dfsNum,-1,sizeof(dfsNum));
   memset(in_stack,false,sizeof(in_stack));
   dfsPos = num_scc = 0;

   for(int i = 0;i<V;++i)
      if(dfsNum[i]==-1)
         tarjan(i);
}
```

2.8. Ciclo de peso promedio mínimo (Karp).

```
#define MAX_V 676

vector< pair<int, int> > L[MAX_V+1];
int dist[MAX_V+1][MAX_V+2];
```

```
void karp(int n){
   for(int i = 0;i<n;++i)
      if(!L[i].empty())
            L[n].push_back(make_pair(i,0));
   ++n;

   for(int i = 0;i<n;++i)
      fill(dist[i],dist[i]+(n+1),INT_MAX);

   dist[n-1][0] = 0;

   for (int k = 1;k<=n;++k) for (int u = 0;u<n;++u){
      if(dist[u][k-1]==INT_MAX) continue;

      for(int i = L[u].size()-1;i>=0;--i)
         dist[L[u][i].first][k] = min(dist[L[u][i].first][k],
                                 dist[u][k-1]+L[u][i].second);
   }

   bool flag = true;

   for(int i = 0;i<n && flag;++i)
      if(dist[i][n]!=INT_MAX)
         flag = false;

   if(flag){
      //El grafo es aciclico
      return;
   }

   double ans = 1e15;

   for(int u = 0;u+1<n;++u){
      if(dist[u][n]==INT_MAX) continue;
      double W = -1e15;

      for(int k = 0;k<n;++k)
         if(dist[u][k]!=INT_MAX)
            W = max(W,(double)(dist[u][n]-dist[u][k])/(n-k));

      ans = min(ans,W);
   }
}
```

2.9. Minimum cost arborescence.

```
#define MAX_V 1000
typedef int edge_cost;
edge_cost INF = INT_MAX;

int V,root,prev[MAX_V];
bool adj[MAX_V][MAX_V];
edge_cost G[MAX_V][MAX_V],MCA;
bool visited[MAX_V],cycle[MAX_V];

void add_edge(int u, int v, edge_cost c){
   if(adj[u][v]) G[u][v] = min(G[u][v],c);
   else G[u][v] = c;
   adj[u][v] = true;
}

void dfs(int v){
   visited[v] = true;

   for(int i = 0;i<V;++i)
      if(!visited[i] && adj[v][i])
         dfs(i);
}

bool check(){
   memset(visited,false,sizeof(visited));
   dfs(root);

   for(int i = 0;i<V;++i)
      if(!visited[i])
         return false;

   return true;
}

int exist_cycle(){
   prev[root] = root;

   for(int i = 0;i<V;++i){
      if(!cycle[i] && i!=root){
         prev[i] = i; G[i][i] = INF;
```

```
            for(int j = 0;j<V;++j)
                if(!cycle[j] && adj[j][i] && G[j][i]<G[prev[i]][i])
                    prev[i] = j;
        }
    }

    for(int i = 0,j;i<V;++i){
        if(cycle[i]) continue;
        memset(visited,false,sizeof(visited));

        j = i;

        while(!visited[j]){
            visited[j] = true;
            j = prev[j];
        }

        if(j==root) continue;
        return j;
    }

    return -1;
}

void update(int v){
    MCA += G[prev[v]][v];

    for(int i = prev[v];i!=v;i = prev[i]){
        MCA += G[prev[i]][i];
        cycle[i] = true;
    }

    for(int i = 0;i<V;++i)
        if(!cycle[i] && adj[i][v])
            G[i][v] -= G[prev[v]][v];

    for(int j = prev[v];j!=v;j = prev[j]){
        for(int i = 0;i<V;++i){
            if(cycle[i]) continue;

            if(adj[i][j]){
                if(adj[i][v]) G[i][v] = min(G[i][v],G[i][j]-G[prev[j]][j]);
                else G[i][v] = G[i][j]-G[prev[j]][j];
                adj[i][v] = true;
            }

            if(adj[j][i]){
                if(adj[v][i]) G[v][i] = min(G[v][i],G[j][i]);
                else G[v][i] = G[j][i];
                adj[v][i] = true;
            }
        }
    }
}

bool min_cost_arborescence(int _root){
    root = _root;
    if(!check()) return false;

    memset(cycle,false,sizeof(cycle));
    MCA = 0;

    int v;

    while((v = exist_cycle())!=-1)
        update(v);

    for(int i = 0;i<V;++i)
        if(i!=root && !cycle[i])
            MCA += G[prev[i]][i];

    return true;
}
```

2.10. Stable marriage.

```
#define MAX_N 500

int N,pref_men[MAX_N][MAX_N],pref_women[MAX_N][MAX_N];
int inv[MAX_N][MAX_N],cont[MAX_N],wife[MAX_N],husband[MAX_N];

void stable_marriage(){
        for(int i = 0;i<N;++i)
                for(int j = 0;j<N;++j)
                        inv[i][pref_women[i][j]] = j;
```

```
    fill(cont,cont+N,0);
    fill(husband,husband+N,-1);

    int m,w,dumped;

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

        while(m>=0){
            while(true){
                w = pref_men[m][cont[m]];
                ++cont[m];
                if(husband[w]<0 || inv[w][m]<inv[w][husband[w]]) break;
            }

            dumped = husband[w];
            husband[w] = m;
            wife[m] = w;
            m = dumped;
        }
    }
}
```

2.11. Bipartite matching (Hopcroft Karp).

```
#define MAX_V1 50000
#define MAX_V2 50000
#define MAX_E 150000

int V1,V2,left[MAX_V2],right[MAX_V1];
int E,to[MAX_E],next[MAX_E],last[MAX_V1];

void hk_init(int v1, int v2){
   V1 = v1; V2 = v2; E = 0;
   memset(last,-1,sizeof last);
}

void hk_add_edge(int u, int v){
   to[E] = v; next[E] = last[u]; last[u] = E++;
}

bool visited[MAX_V1];

bool hk_dfs(int u){
   if(visited[u]) return false;
   visited[u] = true;

   for(int e = last[u],v;e != -1;e = next[e]){
      v = to[e];

      if(left[v] == -1 || hk_dfs(left[v])){
         right[u] = v;
         left[v] = u;
         return true;
      }
   }

   return false;
}

int hk_match(){
   memset(left,-1,sizeof left);
   memset(right,-1,sizeof right);
   bool change = true;

   while(change){
      change = false;
      memset(visited,false,sizeof visited);

      for(int i = 0;i < V1;++i)
         if(right[i] == -1)
            change |= hk_dfs(i);
   }

   int ret = 0;

   for(int i = 0;i < V1;++i)
      if(right[i] != -1) ++ret;

   return ret;
}
```

2.12. Algoritmo húngaro.

```
// Maximiza costo del matching

#define MAX_V 500

int V,cost[MAX_V][MAX_V];
int lx[MAX_V],ly[MAX_V];
int max_match,xy[MAX_V],yx[MAX_V],prev[MAX_V];
bool S[MAX_V],T[MAX_V];
int slack[MAX_V],slackx[MAX_V];
int q[MAX_V],head,tail;

void init_labels(){
   memset(lx,0,sizeof(lx));
   memset(ly,0,sizeof(ly));

   for(int x = 0;x<V;++x)
      for(int y = 0;y<V;++y)
         lx[x] = max(lx[x],cost[x][y]);
}

void update_labels(){
   int x,y,delta = INT_MAX;

   for(y = 0;y<V;++y) if(!T[y]) delta = min(delta,slack[y]);
   for(x = 0;x<V;++x) if(S[x]) lx[x] -= delta;
   for(y = 0;y<V;++y) if(T[y]) ly[y] += delta;
   for(y = 0;y<V;++y) if(!T[y]) slack[y] -= delta;
}

void add_to_tree(int x, int prevx){
   S[x] = true;
   prev[x] = prevx;

   for(int y = 0;y<V;++y){
      if(lx[x]+ly[y]-cost[x][y]<slack[y]){
         slack[y] = lx[x]+ly[y]-cost[x][y];
         slackx[y] = x;
      }
   }
}

void augment(){
   int x,y,root;
```

```
   head = tail = 0;
   memset(S,false,sizeof(S));
   memset(T,false,sizeof(T));
   memset(prev,-1,sizeof(prev));

   for(x = 0;x<V;++x){
      if(xy[x]==-1){
         q[tail++] = root = x;
         prev[root] = -2;
         S[root] = true;
         break;
      }
   }

   for(y = 0;y<V;++y){
      slack[y] = lx[root]+ly[y]-cost[root][y];
      slackx[y] = root;
   }

   while(true){
      while(head<tail){
         x = q[head++];

         for(y = 0;y<V;++y){
            if(cost[x][y]==lx[x]+ly[y] && !T[y]){
               if(yx[y]==-1) break;

               T[y] = true;
               q[tail++] = yx[y];
               add_to_tree(yx[y],x);
            }
         }

         if(y<V) break;
      }

      if(y<V) break;

      update_labels();
      head = tail = 0;

      for(y = 0;y<V;++y){
         if(!T[y] && slack[y]==0){
```

```
            if(yx[y]==-1){
               x = slackx[y];
               break;
            }

            T[y] = true;

            if(!S[yx[y]]){
               q[tail++] = yx[y];
               add_to_tree(yx[y],slackx[y]);
            }
         }
      }

      if(y<V) break;
   }

   ++max_match;
```

```
   for(int cx = x,cy = y,ty;cx!=-2;cx = prev[cx],cy = ty){
      ty = xy[cx];
      yx[cy] = cx;
      xy[cx] = cy;
   }
}

int hungarian(){
   int ret = 0;
   max_match = 0;
   memset(xy,-1,sizeof(xy));
   memset(yx,-1,sizeof(yx));

   init_labels();
   for(int i = 0;i<V;++i) augment();
   for(int x = 0;x<V;++x) ret += cost[x][xy[x]];

   return ret;
}
```

2.13. Non bipartite matching.

```
#define MAXN 222

int n;
bool adj[MAXN][MAXN];
int p[MAXN],m[MAXN],d[MAXN],c1[MAXN], c2[MAXN];
int q[MAXN], *qf, *qb;

int pp[MAXN];
int f(int x) {return x == pp[x] ? x : (pp[x] = f(pp[x]));}
void u(int x, int y) {pp[f(x)] = f(y);}

int v[MAXN];

void path(int r, int x){
   if (r == x) return;

   if (d[x] == 0){
      path(r, p[p[x]]);
      int i = p[x], j = p[p[x]];
      m[i] = j; m[j] = i;
   }
   else if (d[x] == 1){
      path(m[x], c1[x]);
      path(r, c2[x]);
      int i = c1[x], j = c2[x];
      m[i] = j; m[j] = i;
   }
}

int lca(int x, int y, int r){
   int i = f(x), j = f(y);
   while (i != j && v[i] != 2 && v[j] != 1){
      v[i] = 1; v[j] = 2;
      if (i != r) i = f(p[i]);
      if (j != r) j = f(p[j]);
   }

   int b = i, z = j;
   if(v[j] == 1) swap(b, z);

   for (i = b; i != z; i = f(p[i])) v[i] = -1;
   v[z] = -1;
   return b;
}

void shrink_one_side(int x, int y, int b){
```

```
    for(int i = f(x); i != b; i = f(p[i])){
        u(i, b);
        if(d[i] == 1) c1[i] = x, c2[i] = y, *qb++ = i;
    }
}

bool BFS(int r){
    for(int i=0; i<n; ++i)
        pp[i] = i;

    memset(v, -1, sizeof(v));
    memset(d, -1, sizeof(d));

    d[r] = 0;
    qf = qb = q;
    *qb++ = r;

    while(qf < qb){
        for(int x=*qf++, y=0; y<n; ++y){
            if(adj[x][y] && m[y] != y && f(x) != f(y)){
                if(d[y] == -1){
                    if(m[y] == -1){
                                path(r, x);
                                m[x] = y; m[y] = x;
                                return true;
                    }
                    else{
                                p[y] = x; p[m[y]] = y;
                                d[y] = 1; d[m[y]] = 0;
                                *qb++ = m[y];
                    }
                }
                else if(d[f(y)] == 0){
                            int b = lca(x, y, r);
                            shrink_one_side(x, y, b);
                            shrink_one_side(y, x, b);
                }
            }
        }
    }

    return false;
}

int match(){
    memset(m, -1, sizeof(m));
    int c = 0;
    for (int i=0; i<n; ++i)
        if (m[i] == -1)
            if (BFS(i)) c++;
            else m[i] = i;

    return c;
}
```

2.14. Flujo máximo.

```
struct flow_graph{
    int MAX_V,E,s,t;
    int *cap,*to,*next,*last;
    bool *visited;

    flow_graph(){}

    flow_graph(int V, int MAX_E){
        MAX_V = V; E = 0;
        cap = new int[2*MAX_E], to = new int[2*MAX_E], next = new int[2*MAX_E];
        last = new int[MAX_V], visited = new bool[MAX_V];
        fill(last,last+MAX_V,-1);
    }

    void clear(){
        fill(last,last+MAX_V,-1);
        E = 0;
    }

    void add_edge(int u, int v, int uv, int vu = 0){
        to[E] = v, cap[E] = uv, next[E] = last[u]; last[u] = E++;
        to[E] = u, cap[E] = vu, next[E] = last[v]; last[v] = E++;
    }

    int dfs(int v, int f){
        if(v==t || f<=0) return f;
        if(visited[v]) return 0;
        visited[v] = true;

        for(int e = last[v];e!=-1;e = next[e]){
```

```
        int ret = dfs(to[e],min(f,cap[e]));

        if(ret>0){
            cap[e] -= ret;
            cap[e^1] += ret;
            return ret;
        }
    }

    return 0;
}

int max_flow(int source, int sink){
        s = source, t = sink;
        int f = 0,x;

        while(true){
            fill(visited,visited+MAX_V,false);
            x = dfs(s,INT_MAX);
            if(x==0) break;
            f += x;
        }

        return f;
    }
};
```

2.15. Flujo máximo (Dinic).

```
struct flow_graph{
    static const int MAX_V = 500;
    static const int MAX_E = 10000;

    int E,s,t,head,tail;
    int cap[2 * MAX_E],to[2 * MAX_E],next[2 * MAX_E],last[MAX_V],dist[MAX_V],q[MAX_V],now[MAX_V];

    flow_graph(){
        E = 0;
        memset(last,-1,sizeof last);
    }

    void clear(){
        E = 0;
        memset(last,-1,sizeof last);
    }

    void add_edge(int u, int v, int uv){
        to[E] = v, cap[E] = uv, next[E] = last[u]; last[u] = E++;
        to[E] = u, cap[E] = 0, next[E] = last[v]; last[v] = E++;
    }

    bool bfs(){
        memset(dist,-1,sizeof dist);
        head = tail = 0;

        q[tail] = t; ++tail;
        dist[t] = 0;

        while(head < tail){
            int v = q[head]; ++head;

            for(int e = last[v];e != -1;e = next[e]){
                if(cap[e^1] > 0 && dist[to[e]] == -1){
                    q[tail] = to[e]; ++tail;
                    dist[to[e]] = dist[v]+1;
                }
            }
        }

        return dist[s] != -1;
    }

    int dfs(int v, int f){
        if(v == t) return f;

        for(int &e = now[v];e != -1;e = next[e]){
            if(cap[e] > 0 && dist[to[e]] == dist[v]-1){
                int ret = dfs(to[e],min(f,cap[e]));

                if(ret > 0){
                    cap[e] -= ret;
                    cap[e^1] += ret;
                    return ret;
                }
            }
        }
```

```
      return 0;
   }

   int max_flow(int source, int sink){
      s = source; t = sink;
      int f = 0,df;

      while(bfs()){
         for(int i = 0;i <= sink;++i) now[i] = last[i];
         while(true){
            df = dfs(s,INT_MAX);
            if(df == 0) break;
            f += df;
         }
      }

      return f;
   }
};
```

2.16. Flujo máximo - Costo Mínimo (Succesive Shortest Path).

```
#define MAX_V 350
#define MAX_E 2*12500

typedef int cap_type;
typedef long long cost_type;
const cost_type INF = LLONG_MAX;

int V,E,prev[MAX_V],last[MAX_V],to[MAX_E],next[MAX_E];
bool visited[MAX_V];
cap_type flowVal, cap[MAX_E];
cost_type flowCost,cost[MAX_E],dist[MAX_V],pot[MAX_V];

void init(int _V){
   memset(last,-1,sizeof(last));
   V = _V; E = 0;
}

void add_edge(int u, int v, cap_type _cap, cost_type _cost){
   to[E] = v, cap[E] = _cap;
   cost[E] = _cost, next[E] = last[u];
   last[u] = E++;
   to[E] = u, cap[E] = 0;
   cost[E] = -_cost, next[E] = last[v];
   last[v] = E++;
}

// only if there is initial negative cycle
void BellmanFord(int s, int t){
   bool stop = false;
   for(int i = 0;i<V;++i) dist[i] = INF;
   dist[s] = 0;

   for(int i = 1;i<=V && !stop;++i){
      stop = true;

      for(int j = 0;j<E;++j){
         int u = to[j^1], v = to[j];

         if(cap[j]>0 && dist[u]!=INF && dist[u]+cost[j]<dist[v]){
            stop = false;
            dist[v] = dist[u]+cost[j];
         }
      }
   }

   for(int i = 0;i<V;++i) if (dist[i]!=INF) pot[i] = dist[i];
}

void mcmf(int s, int t){
   flowVal = flowCost = 0;
   memset(pot,0,sizeof(pot));

   BellmanFord(s,t);

   while(true){
      memset(prev,-1,sizeof(prev));
      memset(visited,false,sizeof(visited));
      for(int i = 0;i<V;++i) dist[i] = INF;

      priority_queue< pair<cost_type, int> > Q;
      Q.push(make_pair(0,s));
      dist[s] = prev[s] = 0;

      while(!Q.empty()){
```

```
         int aux = Q.top().second;
         Q.pop();

         if(visited[aux]) continue;
         visited[aux] = true;

         for(int e = last[aux];e!=-1;e = next[e]){
            if(cap[e]<=0) continue;
            cost_type new_dist = dist[aux]+cost[e]+pot[aux]-pot[to[e]];
            if(new_dist<dist[to[e]]){
               dist[to[e]] = new_dist;
               prev[to[e]] = e;
               Q.push(make_pair(-new_dist,to[e]));
            }
         }
      }

      if (prev[t]==-1) break;

      cap_type f = cap[prev[t]];
      for(int i = t;i!=s;i = to[prev[i]^1]) f = min(f,cap[prev[i]]);
      for(int i = t;i!=s;i = to[prev[i]^1]){
         cap[prev[i]] -= f;
         cap[prev[i]^1] += f;
      }

      flowVal += f;
      flowCost += f*(dist[t]-pot[s]+pot[t]);

      for(int i = 0;i<V;++i) if (prev[i]!=-1) pot[i] += dist[i];
   }
}
```

2.17. Flujo máximo (Dinic + Lower Bounds).

```
struct flow_graph{
   int V,E,s,t;
   int *flow,*low,*cap,*to,*next,*last,*delta;
   int *dist,*q,*now,head,tail;

   flow_graph(){}

   flow_graph(int V, int E){
      (*this).V = V; (*this).E = 0;
      int TE = 2*(E+V+1);
      flow = new int[TE]; low = new int[TE]; cap = new int[TE];
      to = new int[TE]; next = new int[TE];
      last = new int[V+2]; delta = new int[V];
      dist = new int[V+2]; q = new int[V+2]; now = new int[V+2];
   }

   void clear(int V){
      (*this).V = V; (*this).E = 0;
      fill(last,last+V,-1);
   }

   void add_edge(int a, int b, int l, int u){
      to[E] = b; low[E] = l; cap[E] = u; flow[E] = 0;
      next[E] = last[a]; last[a] = E++;

      to[E] = a; low[E] = 0; cap[E] = 0; flow[E] = 0;
      next[E] = last[b]; last[b] = E++;
   }

   bool bfs(){
      fill(dist,dist+V+2,-1);
      head = tail = 0;

      q[tail] = t; ++tail;
      dist[t] = 0;

      while(head<tail){
         int v = q[head]; ++head;

         for(int e = last[v];e!=-1;e = next[e]){
            if(cap[e^1]>flow[e^1] && dist[to[e]]==-1){
               q[tail] = to[e]; ++tail;
               dist[to[e]] = dist[v]+1;
            }
         }
      }

      return dist[s]!=-1;
   }

   int dfs(int v, int f){
      if(v==t) return f;
```

```
        for(int &e = now[v];e!=-1;e = next[e]){
            if(cap[e]>flow[e] && dist[to[e]]==dist[v]-1){
                int ret = dfs(to[e],min(f,cap[e]-flow[e]));

                if(ret>0){
                    flow[e] += ret;
                    flow[e^1] -= ret;
                    return ret;
                }
            }
        }

        return 0;
    }

    int max_flow(int source, int sink){
        fill(delta,delta+V,0);

        for(int e = 0;e<E;e += 2){
            delta[to[e^1]] -= low[e];
            delta[to[e]] += low[e];
            cap[e] -= low[e];
        }

        last[V] = last[V+1] = -1;
        int sum = 0;

        for(int i = 0;i<V;++i){
            if(delta[i]>0){
                add_edge(V,i,0,delta[i]);
                sum += delta[i];
            }
            if(delta[i]<0) add_edge(i,V+1,0,-delta[i]);
        }

        add_edge(sink,source,0,INT_MAX);

        s = V; t = V+1;
        int f = 0,df;
        fill(flow,flow+E,0);

        while(bfs()){
            for(int i = V+1;i>=0;--i) now[i] = last[i];

            while(true){
                df = dfs(s,INT_MAX);
                if(df==0) break;
                f += df;
            }
        }

        if(f!=sum) return -1;

        for(int e = 0;e<E;e += 2){
            cap[e] += low[e];
            flow[e] += low[e];
            flow[e^1] -= low[e];
            cap[e^1] -= low[e];
        }

        s = source; t = sink;

        last[s] = next[last[s]];
        last[t] = next[last[t]];
        E -= 2;

        while(bfs()){
            for(int i = V-1;i>=0;--i) now[i] = last[i];

            while(true){
                df = dfs(s,INT_MAX);
                if(df==0) break;
                f += df;
            }
        }

        return f;
    }
};
```

2.18. Corte mínimo de un grafo (Stoer - Wagner).

```
#define MAX_V 500
int M[MAX_V][MAX_V],w[MAX_V];
bool A[MAX_V],merged[MAX_V];

int minCut(int n){
   int best = INT_MAX;
   for(int i=1;i<n;++i) merged[i] = false;
   merged[0] = true;

   for(int phase=1;phase<n;++phase){
      A[0] = true;

      for(int i=1;i<n;++i){
         if(merged[i]) continue;
         A[i] = false;
         w[i] = M[0][i];
      }

      int prev = 0,next;

      for(int i=n-1-phase;i>=0;--i){
         // hallar siguiente vertice que no esta en A
         next = -1;

         for(int j=1;j<n;++j)
            if(!A[j] && (next==-1 || w[j]>w[next]))
               next = j;

         A[next] = true;

         if(i>0){
            prev = next;

            // actualiza los pesos
            for(int j=1;j<n;++j)
               if(!A[j]) w[j] += M[next][j];
         }
      }

      if(best>w[next]) best = w[next];

      // mezcla s y t
      for(int i=0;i<n;++i){
         M[i][prev] += M[next][i];
         M[prev][i] += M[next][i];
      }

      merged[next] = true;
   }

   return best;
}
```

2.19. Graph Facts.

```
Un grafo es bipartito si y solo si no contiene ciclos de longitud impar.

Todos los arboles son bipartitos.

Las aristas que forman un ciclo, se encuentran en una misma componente biconexa.

Minimum Vertex Cover: para V = (S,T)
DFS desde los vertices que no estan cubiertos por alguna arista del matching,
para moverse:

- De izq. a der. usar las aristas que no estan en el matching
- De der. a izq. usar las aristas que estan en el matching

Estan en el vertex cover(independent set):

- De S los no alcanzados(los alcanzados)
- De T los alcanzados(los no alcanzados)

Para usar Teorema de Dilworth colocar tambien aristas que resulten de la transitividad.

Un grafo con grados de vertices iguales a 1 o 2, consiste solo de caminos y ciclos.
```

3. Cadenas

3.1. Knuth-Morris-Pratt.

```
#define MAX_L 70
int f[MAX_L];

void prefixFunction(string P){
   int n = P.size(), k = 0;
   f[0] = 0;

   for(int i=1;i<n;++i){
      while(k>0 && P[k]!=P[i]) k = f[k-1];
      if(P[k]==P[i]) ++k;
      f[i] = k;
   }
}

int KMP(string P, string T){
   int n = P.size(), L = T.size(), k = 0, ans = 0;

   for(int i=0;i<L;++i){
      while(k>0 && P[k]!=T[i]) k = f[k-1];
      if(P[k]==T[i]) ++k;

      if(k==n){
         ++ans;
         k = f[k-1];
      }
   }

   return ans;
}
```

3.2. Aho-Corasick.

```
struct AhoCorasick{
   static const int UNDEF = 0;
   static const int MAXN = 360;
   static const int CHARSET = 26;

   int end,have;
   int tag[MAXN];
   int fail[MAXN];
   int trie[MAXN][CHARSET];

   void init(){
      tag[0] = UNDEF;
      fill(trie[0],trie[0] + CHARSET,-1);
      end = 1;
      have = 0;
   }

   void add(int len, const int* s){
      int p = 0;

      for(int i = 0; i < len; ++i){
         if(trie[p][*s] == -1) {
            tag[end] = UNDEF;
            fill(trie[end],trie[end] + CHARSET,-1);
            trie[p][*s] = end++;
         }

         p = trie[p][*s];
         ++s;
      }

      tag[p] = (1 << have);
      ++have;
   }

   void build(){
      queue<int> bfs;
      fail[0] = 0;

      for(int i = 0;i < CHARSET;++i){
         if(trie[0][i] != -1){
            fail[trie[0][i]] = 0;
            bfs.push(trie[0][i]);
         }else{
            trie[0][i] = 0;
         }
      }
```

```
      while(!bfs.empty()){
         int p = bfs.front();
         tag[p] |= tag[fail[p]];
         bfs.pop();

         for(int i = 0;i < CHARSET;++i){
            if(trie[p][i] != -1){
               fail[trie[p][i]] = trie[fail[p]][i];
                  bfs.push(trie[p][i]);
               }else{
                trie[p][i] = trie[fail[p]][i];
               }
            }
         }
      }
   }
};
```

3.3. Algoritmo Z.

```
int next[MAX_P_LEN];
// next[i] : lcp entre la cadena y su sufijo
// a partir del i-esimo caracter

void prefix_kmp(char *P){
   int L = strlen(P),p = 0,t;

   for(int i = 1;i < L;i++){
      if(i < p && next[i-t] < p-i) next[i] = next[i-t];
      else{
         int j = max(0, p-i);

         while(i+j < L && P[i+j] == P[j]) ++j;

         next[i] = j;
         p = i + j;
         t = i;
      }
   }
}

void LCP(char * P, char *T, int *lcp){
   int LP = strlen(P),LT = strlen(T);
   int p = 0,t;

   for(int i = 0;i < LT;i++){
      if(i < p && next[i-t] < p-i) lcp[i] = next[i-t];
      else{
         int j = max(0,p-i);

         while(i+j < LT && T[i+j] == P[j]) ++j;

         lcp[i] = j;
         p = i + j;
         t = i;
      }
   }
}
```

3.4. Palíndromos.

```
void manacher(int n, const char s[], int p[]){
  for (int i = 0, j = 0, k = 0; i <= 2 * (n - 1); ++i){
   int l = i < k ? min(p[j + j - i], (k - i) / 2) : 0;
   int a = i / 2 - l, b = (i + 1) / 2 + l;

   while(0 <= a && b < n && s[a] == s[b]){
    --a;
    ++b;
    ++l;
   }
   p[i] = l;

   if(k < 2 * b){
    j = i;
    k = 2 * b;
   }
  }
}
```

4. Geometría

4.1. Punto y Línea.

```
const double eps = 1e-9;

struct point{
   double x,y;

   point(){}
   point(double _x, double _y) : x(_x), y(_y){}

   double cross(point P){
      return x * P.y - y * P.x;
   }

   bool operator < (const point &p) const{
      if(fabs(x-p.x)>eps) return x<p.x;
      return y>p.y;
   }
};

double cross(point a, point b){
   return a.x * b.y - a.y * b.x;
}

bool polar_cmp(point a, point b){
   if(a.x >= 0 && b.x < 0) return true;
   if(a.x < 0 && b.x >= 0) return false;
   if(a.x == 0 && b.x == 0){
      if(a.y > 0 && b.y < 0) return false;
      if(a.y < 0 && b.y > 0) return true;
   }
   return cross(a,b) > 0;
}

struct line{
   point p1,p2;

   line(){}

   line(point _p1, point _p2){
      p1 = _p1; p2 = _p2;
      if(p1.x>p2.x) swap(p1,p2);
   }
};
```

4.2. Ángulo entre dos vectores.

```
double get_angle(point P1, point P2){
   double sina = P1.y / P1.abs(),cosa = P1.x / P1.abs();
   double sinb = P2.y / P2.abs(),cosb = P2.x / P2.abs();
   double sinc = sinb * cosa - sina * cosb;
   double cosc = cosb * cosa + sina * sinb;
   double x = atan2(sinc,cosc);
   if(x < 0) x += 2 * M_PI;

   return x;
}
```

4.3. Círculos.

```
point get_center(point A, point B, point C){
   point v1 = (B - A).perp(),v2 = C - A;
   point m1 = (A + B) * 0.5;
   point m2 = (A + C) * 0.5;
   double k = (m2 - m1).dot(v2) / v1.dot(v2);
   return m1 + v1 * k;
}
```

4.4. Polígonos.

```
//verdadero : sentido anti-horario, Complejidad : O(n)
bool ccw(const vector<point> &poly){
   //primero hallamos el punto inferior ubicado mas a la derecha
   int ind = 0,n = poly.size();
   double x = poly[0].x,y = poly[0].y;

   for(int i=1;i<n;i++){
      if (poly[i].y>y) continue;
      if (fabs(poly[i].y-y)<eps && poly[i].x<x) continue;
      ind = i;
      x = poly[i].x;
      y = poly[i].y;
   }

   if (ind==0) return ccw(poly[n-1],poly[0],poly[1]);
   return ccw(poly[ind-1],poly[ind],poly[(ind+1)%n]);
}

bool isInConvex(vector <Point> &A, const Point &P){
  int n = A.size(), lo = 1, hi = A.size() - 1;

  if(area(A[0], A[1], P) <= 0) return 0;
  if(area(A[n-1], A[0], P) <= 0) return 0;

  while(hi - lo > 1){
    int mid = (lo + hi) / 2;

    if(area(A[0], A[mid], P) > 0) lo = mid;
    else hi = mid;
  }

  return area(A[lo], A[hi], P) > 0;
}

bool PointInsidePolygon(const point &P, const vector<point> &poly){
   int n = poly.size();
   bool in = 0;

   for(int i = 0,j = n-1;i<n;j = i++){
      double dx = poly[j].x-poly[i].x;
      double dy = poly[j].y-poly[i].y;

      if((poly[i].y<=P.y+eps && P.y<poly[j].y) ||
         (poly[j].y<=P.y+eps && P.y<poly[i].y))
         if(P.x-eps<dx*(P.y-poly[i].y)/dy+poly[i].x)
            in ^= 1;
   }

   return in;
}

//valor positivo : vertices orientados en sentido antihorario
//valor negativo : vertices orientados en sentido horario
double signed_area(const vector<point> &poly){
  int n = poly.size();
  if(n<3) return 0.0;

  double S = 0.0;

  for(int i=1;i<=n;++i)
    S += poly[i%n].x*(poly[(i+1)%n].y-poly[i-1].y);

  S /= 2;
  return S;
}
```

4.5. Convex Hull (Monotone Chain).

```
vector<point> ConvexHull(vector<point> P){
   sort(P.begin(),P.end());
   int n = P.size(),k = 0;
   point H[2*n];

   for(int i=0;i<n;++i){
      while(k>=2 && !ccw(H[k-2],H[k-1],P[i])) --k;
      H[k++] = P[i];
   }

   for(int i=n-2,t=k;i>=0;--i){
      while(k>t && !ccw(H[k-2],H[k-1],P[i])) --k;
```

```
      H[k++] = P[i];
   }

   return vector<point> (H,H+k);
}
```

4.6. Teorema de Pick.

```
A = I + B/2 - 1, donde:
A = Area de un poligono de coordenadas enteras
I = Numero de puntos enteros en su interior
B = Numero de puntos enteros sobre sus bordes

int IntegerPointsOnSegment(const point &P1, const point &P2){
   point P = P1-P2;
   P.x = abs(P.x); P.y = abs(P.y);

   if(P.x == 0) return P.y;
   if(P.y == 0) return P.x;
   return (__gcd(P.x,P.y));
}

Se asume que los vertices tienen coordenadas enteras. Sumar el valor de esta
funcion para todas las aristas para obtener el numero total de punto en el borde
del poligono.
```

4.7. Par de puntos más cercano (Sweep Line).

```
#define MAX_N 100000
#define px second
#define py first
typedef pair<long long, long long> point;

int N;
point P[MAX_N];
set<point> box;

bool compare_x(point a, point b){ return a.px<b.px; }

inline double dist(point a, point b){
   return sqrt((a.px-b.px)*(a.px-b.px)+(a.py-b.py)*(a.py-b.py));
}

double closest_pair(){
   if(N<=1) return -1;

   sort(P,P+N,compare_x);

   double ret = dist(P[0],P[1]);
   box.insert(P[0]);

   set<point> :: iterator it;

   for(int i = 1,left = 0;i<N;++i){
      while(left<i && P[i].px-P[left].px>ret) box.erase(P[left++]);
      for(it = box.lower_bound(make_pair(P[i].py-ret,P[i].px-ret));
         it!=box.end() && P[i].py+ret>=(*it).py;++it)
            ret = min(ret, dist(P[i],*it));
      box.insert(P[i]);
   }

   return ret;
}
```

4.8. Par de puntos más cercano (Divide and Conquer).

```
void closest_pair(int l, int r){
   if(l == r) return;

   int mi = (l + r) >> 1;
   int X = p[mi].x;
```

```
   closest_pair(l,mi);
   closest_pair(mi + 1,r);

   int m = 0;

   for(int i = l;i <= r;++i)
      if(abs(X - p[i].x) <= best)
         aux[m++] = point(p[i].y,p[i].x,p[i].id);

   sort(aux,aux + m);

   for(int i = 0;i < m;++i){
      int e = i + 1;

         while(e < m && aux[e].x - aux[i].x <= best + EPS){
            double d = dist(aux[i],aux[e]);

            if(d < best){
               best = d;
               id1 = aux[i].id;
               id2 = aux[e].id;
            }

            ++e;
         }
      }
   }
```

4.9. Unión de rectángulos (Área).

```
#define MAX_N 10000

struct event{
   int ind;
   bool type;

   event(){};
   event(int ind, int type) : ind(ind), type(type) {};
};


struct point{
   int x,y;
};

int N;
point rects[MAX_N][2];
// rects[i][0] : esquina inferior izquierda
// rects[i][1] : esquina superior derecha
event events_v[2*MAX_N],events_h[2*MAX_N];
bool in_set[MAX_N];

bool compare_x(event a, event b){
   return rects[a.ind][a.type].x<rects[b.ind][b.type].x;
}
bool compare_y(event a, event b){
   return rects[a.ind][a.type].y<rects[b.ind][b.type].y;
}


long long union_area(){
   int e = 0;

   for(int i = 0;i<N;++i){
      events_v[e] = event(i,0);
      events_h[e] = event(i,0);
      ++e;
      events_v[e] = event(i,1);
      events_h[e] = event(i,1);
      ++e;
   }

   sort(events_v,events_v+e,compare_x);
   sort(events_h,events_h+e,compare_y);

   memset(in_set,false,sizeof(in_set));
   in_set[events_v[0].ind] = true;
   long long area = 0;

   int prev_ind = events_v[0].ind, cur_ind;
   int prev_type = events_v[0].type, cur_type;

   for(int i = 1;i<e;++i){
      cur_ind = events_v[i].ind; cur_type = events_v[i].type;
      int cont = 0, dx = rects[cur_ind][cur_type].x-rects[prev_ind][prev_type].x;
      int begin_y;
```

```
      if(dx!=0){
         for(int j = 0;j<e;++j){
            if(in_set[events_h[j].ind]){
               if(events_h[j].type==0){
                  if(cont==0) begin_y = rects[events_h[j].ind][0].y;
                  ++cont;
               }else{
                  --cont;
                  if(cont==0){
                     int dy = rects[events_h[j].ind][1].y-begin_y;
                     area += (long long)dx*dy;
                  }
               }
            }
         }
      }

      in_set[cur_ind] = (cur_type==0);
      prev_ind = cur_ind; prev_type = cur_type;
   }

   return area;
}
```

4.10. Geometría 3D.

```
struct XYZ{
   double x,y,z;

   XYZ(){}
   XYZ(double _x, double _y, double _z) :
      x(_x), y(_y), z(_z){}

   void normalize(){
      double r = sqrt(x * x + y * y + z * z);
      x /= r; y /= r; z /= r;
   }

   XYZ cross(XYZ p){
      return XYZ(y * p.z - z * p.y,z * p.x - x * p.z,x * p.y - y * p.x);
   }

   double dot(XYZ p){
      return x * p.x + y * p.y + z * p.z;
   }
};

// rotar p con eje de rotacion r
XYZ rotate(XYZ p, XYZ r, double theta){
   XYZ q(0,0,0);
   double costheta,sintheta;

   r.normalize();
   costheta = cos(theta);
   sintheta = sin(theta);

   q.x += (costheta + (1 - costheta) * r.x * r.x) * p.x;
   q.x += ((1 - costheta) * r.x * r.y - r.z * sintheta) * p.y;
   q.x += ((1 - costheta) * r.x * r.z + r.y * sintheta) * p.z;

   q.y += ((1 - costheta) * r.x * r.y + r.z * sintheta) * p.x;
   q.y += (costheta + (1 - costheta) * r.y * r.y) * p.y;
   q.y += ((1 - costheta) * r.y * r.z - r.x * sintheta) * p.z;

   q.z += ((1 - costheta) * r.x * r.z - r.y * sintheta) * p.x;
   q.z += ((1 - costheta) * r.y * r.z + r.x * sintheta) * p.y;
   q.z += (costheta + (1 - costheta) * r.z * r.z) * p.z;

   return q;
}
```

5. Matemática

5.1. GCD extendido.

```
// a*x + b*y = gcd(a,b)
int extGcd(int a, int b, int &x, int &y){
   if(b == 0){
      x = 1;
      y = 0;
      return a;
   }

   int g = extGcd(b,a % b,y,x);
   y -= a / b * x;
   return g;
}

// ASSUME: gcd(a, m) == 1
int modInv(int a, int m){
   int x,y;
   extGcd(a, m, x, y);
   return (x % m + m) % m;
}
```

5.2. Teorema chino del resto.

```
// rem y mod tienen el mismo numero de elementos
long long chinese_remainder(vector<int> rem, vector<int> mod){
   long long ans = rem[0],m = mod[0];
   int n = rem.size();

   for(int i=1;i<n;++i){
      int a = modular_inverse(m,mod[i]);
      int b = modular_inverse(mod[i],m);
      ans = (ans*b*mod[i]+rem[i]*a*m)%(m*mod[i]);
      m *= mod[i];
   }

   return ans;
}
```

5.3. Número combinatorio.

```
long long comb(int n, int m){
   if(m>n-m) m = n-m;

   long long C = 1;
   //C^{n}_{i} -> C^{n}_{i+1}
   for(int i=0;i<m;++i) C = C*(n-i)/(1+i);
   return C;
}
```

Cuando n y m son grandes y se pide comb(n,m)%MOD, donde MOD es un numero primo, se puede usar el Teorema de Lucas.

```
#define MOD 3571

int C[MOD][MOD];

void FillLucasTable(){
   memset(C,0,sizeof(C));
   for(int i=0;i<MOD;++i) C[i][0] = 1;
   for(int i=1;i<MOD;++i) C[i][i] = 1;
   for(int i=2;i<MOD;++i)
      for(int j=1;j<i;++j)
         C[i][j] = (C[i-1][j]+C[i-1][j-1])%MOD;
}

int comb(int n, int k){
   long long ans = 1;

   while(n!=0){
      int ni = n%MOD,ki = k%MOD;
      n /= MOD; k /= MOD;
      ans = (ans*C[ni][ki])%MOD;
   }

   return (int)ans;
}
```

5.4. Test de Miller-Rabin.

```
typedef unsigned long long ULL;

ULL mulmod(ULL a, ULL b, ULL c){
   ULL x = 0,y = a % c;

   while(b > 0){
      if(b & 1) x += y;
      y <<= 1;
      if(x >= c) x -= c;
      if(y >= c) y -= c;
      b >>= 1;
   }

   return x;
}

ULL pow(ULL a, ULL b, ULL c){
   ULL x = 1,y = a;

   while(b > 0){
      if(b & 1) x = mulmod(x,y,c);
      y = mulmod(y,y,c);
      b >>= 1;
   }

   return x;
}

bool miller_rabin(ULL p, int it){
   if(p < 2) return false;
   if(p == 2) return true;
   if((p & 1) == 0) return false;

   ULL s = p - 1;
   while(s % 2 == 0) s >>= 1;

   while(it--){
      ULL a = rand() % (p-1) + 1,temp = s;
      ULL mod = pow(a,temp,p);

      if(mod == -1 || mod == 1) continue;

      while(temp != p-1 && mod != p-1){
         mod = mulmod(mod,mod,p);
         temp <<= 1;
      }

      if(mod != p-1) return false;
   }

   return true;
}
```

5.5. Polinomios.

```
vector<int> add(vector<int> &a, vector<int> &b){
   int n = a.size(),m = b.size(),sz = max(n,m);
   vector<int> c(sz,0);

   for(int i = 0;i<n;++i) c[i] += a[i];
   for(int i = 0;i<m;++i) c[i] += b[i];

      // mejor no quitar si son reales
   while(sz>1 && c[sz-1]==0){
      c.pop_back();
      --sz;
   }

   return c;
}

vector<int> multiply(vector<int> &a, vector<int> &b){
   int n = a.size(),m = b.size(),sz = n+m-1;
   vector<int> c(sz,0);

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

      // mejor no quitar si son reales
   while(sz>1 && c[sz-1]==0){
```

```
      c.pop_back();
      --sz;
   }

   return c;
}

bool is_root(vector<int> &P, int r){
   int n = P.size();
   long long y = 0;

   for(int i = 0;i<n;++i){
      if(abs(y-P[i])%r!=0) return false;
      y = (y-P[i])/r;
   }

   return y==0;
}
```

5.6. Fast Fourier Transform.

```
struct Complex{
   double x,y;

   Complex(){}
   Complex(double _x, double _y):
      x(_x), y(_y){}

   void operator += (Complex &c){
      x += c.x; y += c.y;
   }

   Complex operator -= (Complex &c){
      x -= c.x; y -= c.y;
   }

   Complex operator * (Complex &c){
      return Complex(x * c.x - y * c.y,x * c.y + y * c.x);
   }
};

#define MAXN 262144
Complex A2[MAXN];

void fft(int n, Complex A[], int s){
   int p = __builtin_ctz(n);

   memcpy(A2,A,sizeof(Complex) * n);

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

      for(int j = 0;j < p;++j){
         rev <<= 1;
         rev |= ((i >> j) & 1);
      }

      A[i] = A2[rev];
   }

   Complex w,wn;
   int M = 2,K = 1;

   for(int i = 1;i <= p;++i,M <<= 1,K <<= 1){
      wn = Complex(cos(s * 2 * M_PI / M),sin(s * 2 * M_PI / M));

      for(int j = 0;j < n;j += M){
         w = Complex(1,0);

         for(int l = j;l < K+j;++l){
            Complex t = w * A[l + K],u = A[l];

            A[l] += t;
            u -= t;
            A[l + K] = u;
            w = w * wn;
         }
      }
   }

   if(s == -1)
      for(int i = 0;i < n;++i)
         A[i].x /= n, A[i].y /= n;;
}

Complex R[MAXN];
int nR;
```

```
void fft_mult(int nP, Complex P[], int nQ, Complex Q[]){
   nR = nP + nQ;
   while(__builtin_popcount(nR) > 1) nR += nR & -nR;

   for(int i = nP;i < nR;++i) P[i] = Complex(0,0);
   for(int i = nQ;i < nR;++i) Q[i] = Complex(0,0);

   fft(nR,P,1);
   fft(nR,Q,1);

   for(int i = 0;i < nR;i++) R[i] = P[i] * Q[i];

   fft(nR,R,-1);
}
```

5.7. Stern Brocott.

```
const int MAX_DEN = 3000;
vector<int> Fnum,Fden;

void build(int lnum = 0, int lden = 1, int rnum = 1, int rden = 1){
   int a = lnum+rnum,b = lden+rden;
   if(b>MAX_DEN) return;

   build(lnum,lden,a,b);

   Fnum.push_back(a);
   Fden.push_back(b);

   build(a,b,rnum,rden);
}
```

6. Estructuras de datos

6.1. Lowest Common Ancestor.

```
#define MAX_N 100000
#define LOG2_MAXN 16

// NOTA : memset(parent,-1,sizeof(parent));
int N,parent[MAX_N],L[MAX_N];
int P[MAX_N][LOG2_MAXN + 1];

int get_level(int u){
   if(L[u]!=-1) return L[u];
   else if(parent[u]==-1) return 0;
   return 1+get_level(parent[u]);
}

void init(){
      memset(L,-1,sizeof(L));
      for(int i = 0;i<N;++i) L[i] = get_level(i);

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

      for(int i = 0;i<N;++i) P[i][0] = parent[i];

      for(int j = 1;(1<<j)<N;++j)
         for(int i = 0;i<N;++i)
               if(P[i][j-1]!=-1)
                     P[i][j] = P[P[i][j-1]][j-1];
}

int LCA(int p, int q){
   if(L[p]<L[q]) swap(p,q);

   int log = 1;
   while((1<<log)<=L[p]) ++log;
   --log;

   for(int i = log;i>=0;--i)
      if(L[p]-(1<<i)>=L[q])
         p = P[p][i];

   if(p==q) return p;
```

```
   for(int i = log;i>=0;--i){
      if(P[p][i]!=-1 && P[p][i]!=P[q][i]){
         p = P[p][i];
         q = P[q][i];
      }
   }

   return parent[p];
}
```

6.2. Heavy-Light Descomposition.

```
struct HeavyLight{
   static const int MAXN = 100005;

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

   int nodedad[MAXN];
   int treesize[MAXN];

   int pos,cntchain;
   int chainleader[MAXN];
   int homechain[MAXN];
   int homepos[MAXN];

   void init(int n){
      N = n;
      for(int i = 0;i < n;++i) E[i].clear();
      pos = cntchain = 0;
   }

   void add_edge(int u, int v){
      E[u].push_back(v);
      E[v].push_back(u);
   }

   void explore(int x = 0, int dad = -1){
      nodedad[x] = dad;
      treesize[x] = 1;

      int sz = E[x].size();

      for(int i = 0;i < sz;++i){
         if(E[x][i] != dad){
            explore(E[x][i], x);
            treesize[x] += treesize[ E[x][i] ];
         }
      }
   }

   void heavy_light(int x = 0, int dad = -1, int k = -1, int p = 0){
      if(p == 0){
         k = cntchain++;
         chainleader[k] = x;
      }

      homechain[x] = k;
      homepos[x] = pos++;

      int mx = -1,sz = E[x].size();

      for(int i = 0;i < sz;++i)
         if(E[x][i] != dad && (mx == -1 || treesize[ E[x][i] ] > treesize[ E[x][mx] ]) )
            mx = i;

      if(mx != -1) heavy_light(E[x][mx], x, k, p + 1);

      for(int i = 0;i < sz;++i)
         if(E[x][i] != dad && i != mx)
            heavy_light(E[x][i], x, -1, 0);
   }
};
```

6.3. Treap.

```
typedef long long ptype;
```

```
ptype seed = 47;

ptype my_rand(){
   seed = (seed * 279470273) % 4294967291LL;
   return seed;
}

typedef struct node * pnode;

struct node{
  int x, y, cnt;
  pnode L, R;
  node() {}
  node(int x, int y): x(x), y(y), cnt(1), L(NULL), R(NULL) {}
};

pnode T;

inline int cnt(pnode &it){
  return it ? it->cnt : 0;
}

inline void upd_cnt(pnode &it){
  if (it){
    it->cnt = cnt(it->L) + cnt(it->R) + 1;
  }
}

// Split Treap
void split(pnode t, int x, pnode &L, pnode &R){
  if (!t) L = R = NULL;
  else{
    if (x < t->x)
      split (t->L, x, L, t->L), R = t;
    else
      split (t->R, x, t->R, R), L = t;
    upd_cnt(t);
  }
}

// Split Implicit Treap
void split(pnode t, pnode &L, pnode &R, int key){
  if (!t) L = R = NULL;
  else{
    int cntL = cnt(t->L);
    if (key <= cntL)
      split (t->L, L, t->L, key), R = t;
    else
      split (t->R, t->R, R, key - cntL - 1), L = t;
    upd_cnt(t);
  }
}

// For Treap & Implicit Treap
void merge(pnode &t, pnode L, pnode R){
  if (!L) t = R;
  else if(!R) t = L;
  else if (L->y > R->y)
    merge (L->R, L->R, R), t = L;
  else
    merge (R->L, L, R->L), t = R;
  upd_cnt(t);
}

// Combines 2 treaps
pnode unite (pnode l, pnode r) {
  if (!l || !r) return l? l: r;
  if (l->y > r->y) swap (l, r);
  pnode lt, rt;
  split (r, l->x, lt, rt);
  l->L = unite(l->L, lt);
  l->R = unite(l->R, rt);
  return l;
}

// Find in Treap
bool find (pnode &t, int x) {
  if(!t) return 0;
  else if (t->x == x) return 1;
  else return find (x < t->x ? t->L: t->R, x);
}

// Erase from Treap
void erase (pnode &t, int x) {
  if (t-> x == x)
    merge (t, t->L, t->R);
  else
    erase (x < t->x ? t->L: t->R, x);
}

// Insert into Treap
void insert(pnode &t, pnode it) {
```

```
 if (!t) t = it;
 else if (it->y > t->y)
  split (t, it->x, it->L, it->R), t = it;
 else insert (it->x < t->x ? t->L: t->R, it);
}

// Insert into Treap and return the # of greater elements
int insert(pnode &t, pnode it){
 int ret = 0;
 if (!t) t = it;
 else if (it->y > t->y)
  split (t, it->x, it->L, it->R), t = it, ret = cnt(t->R);
 else if(it->x < t->x)
   ret = 1 + cnt(t->R) + insert(t->L, it);
  else
   ret = insert(t->R, it);
  upd_cnt(t);
  return ret;
}

// Safely insert into Treap
void insert(int x)
{
 if(!find(T, x))
  insert(T, new node(x, rand()));
}
```

7. Matrices

8. Mathematical facts

8.1. **Números de Catalan.** están definidos por la recurrencia:

$$C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$$

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1}$$

8.2. **Números de Stirling de primera clase.** son el número de permutaciones de n elementos con exactamente k ciclos disjuntos.

$$\begin{bmatrix} n \\ k \end{bmatrix} = (n-1)\begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix}$$

8.3. **Números de Stirling de segunda clase.** son el número de particionar un conjunto de n elementos en k subconjuntos no vacíos.

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = k\begin{Bmatrix} n-1 \\ k \end{Bmatrix} + \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix}$$

Además:

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j} j^n$$

8.4. **Números de Bell.** cuentan el número de formas de dividir n elementos en subconjuntos.

$$\mathcal{B}_{n+1} = \sum_{k=0}^{n}\binom{n}{k}\mathcal{B}_k$$

x	0	1	2	3	4	5	6	7	8	9	10
$\mathcal{B}_x$	1	1	2	5	15	52	203	877	4.140	21.147	115.975

8.5. **Derangement.** permutación que no deja ningún elemento en su lugar original

$$!n = (n-1)(!(n-1)+!(n-2)); !1 = 0, !2 = 1$$

$$!n = n!\sum_{k=0}^{n}\frac{(-1)^k}{k!}$$

8.6. **Números armónicos.**

$$H_n = \sum_{k=1}^{n} \frac{1}{k}$$

$$\frac{1}{2n+1} < H_n - \ln n - \gamma < \frac{1}{2n}$$

$\gamma = 0.5772156649015328606065120900824024310421593359\ldots$

8.7. **Número de Fibonacci.** $f_0 = 0,\ f_1 = 1$:

$$f_n = \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}}{2})^n - \frac{1}{\sqrt{5}}(\frac{1-\sqrt{5}}{2})^n$$

$$f_{n+1}^2 + f_n^2 = f_{2n+1}, f_{n+2}^2 - f_n^2 = f_{2n+2}$$

$$f_n = \sum_{j=0}^{\lfloor \frac{n}{2} \rfloor} \binom{n-j}{j}$$

8.8. **Sumas de combinatorios.**

$$\sum_{i=n}^{m} \binom{i}{n} = \binom{m+1}{n+1}$$

$$\sum_{i=0}^{k} \binom{n}{i}\binom{m}{k-i} = \binom{n+m}{k}$$

8.9. **Funciones generatrices.** Una lista de funciones generatrices para secuencias útiles:

$(1,1,1,1,1,1,\ldots)$	$\frac{1}{1-z}$
$(1,-1,1,-1,1,-1,\ldots)$	$\frac{1}{1+z}$
$(1,0,1,0,1,0,\ldots)$	$\frac{1}{1-z^2}$
$(1,0,\ldots,0,1,0,1,0,\ldots,0,1,0,\ldots)$	$\frac{1}{1-z^2}$
$(1,2,3,4,5,6,\ldots)$	$\frac{1}{(1-z)^2}$
$(1,\binom{m+1}{m},\binom{m+2}{m},\binom{m+3}{m},\ldots)$	$\frac{1}{(1-z)^{m+1}}$
$(1,c,\binom{c+1}{2},\binom{c+2}{3},\ldots)$	$\frac{1}{(1-z)^c}$
$(1,c,c^2,c^3,\ldots)$	$\frac{1}{1-cz}$
$(0,1,\frac{1}{2},\frac{1}{3},\frac{1}{4},\ldots)$	$\ln\frac{1}{1-z}$

Truco de manipulación:

$$\frac{1}{1-z}G(z) = \sum_n \sum_{k\le n} g_k z^n$$

8.10. **The twelvefold way.** ¿Cuántas funciones $f\colon N \to X$ hay?

N	X	Any f	Injective	Surjective
dist.	dist.	x^n	$(x)_n$	$x!\left\{ {n \atop x} \right\}$
indist.	dist.	$\binom{x+n-1}{n}$	$\binom{x}{n}$	$\binom{n-1}{n-x}$
dist.	indist.	$\left\{ {n \atop 1} \right\} + \ldots + \left\{ {n \atop x} \right\}$	$[n \le x]$	$\left\{ {n \atop k} \right\}$
indist.	indist.	$p_1(n) + \ldots p_x(n)$	$[n \le x]$	$p_x(n)$

Where $\binom{a}{b} = \frac{1}{b!}(a)_b$ and $p_x(n)$ is the number of ways to partition the integer n using x summands.

8.11. **Teorema de Euler.** si un grafo conexo, plano es dibujado sobre un plano sin intersección de aristas, y siendo v el número de vértices, e el de aristas y f la cantidad de caras (regiones conectadas por aristas, incluyendo la región externa e infinita), entonces

$$v - e + f = 2$$

8.12. **Burnside's Lemma.** Si X es un conjunto finito y G es un grupo de permutaciones que actúa sobre X, sean $S_x = \{g \in G : g * x = x\}$ y $Fix(g) = \{x \in X : g * x = x\}$. Entonces el número de órbitas está dado por:

$$N = \frac{1}{|G|} \sum_{x \in X} |S_x| = \frac{1}{|G|} \sum_{g \in G} |Fix(g)|$$

8.13. **Ángulo entre dos vectores.** Sea α el ángulo entre $\vec{a}$ y $\vec{b}$:

$$\cos\alpha = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|}$$

8.14. **Proyección de un vector.** Proyección de $\vec{a}$ sobre $\vec{b}$:

$$\text{proy}_{\vec{b}}\vec{a} = (\frac{\vec{a} \cdot \vec{b}}{\vec{b} \cdot \vec{b}})\vec{b}$$

ACM ICPC Team Reference - Contenidos

Universidad Nacional de Ingeniería - FIIS

Contents