ACM ICPC
Regional 2010

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. Lectura rápida de enteros.

```
void readInt(int &n){
   int sign = 1;
   char c;
   bool found = false;
   n = 0;

   while(true){
      c = getc(stdin);
```

```
    switch(c){
      case '-' :
        sign = -1;
        found = true;
        break;
      case '␣':
        if(found) goto jump;
        break;
      case '\n':
        if(found) goto jump;
        break;
      default:
        if(c>='0' && c<='9'){
          n = n*10+c-'0';
          found = true;
        }else goto jump;
        break;
    }
  }

  jump:
    n *= sign;
}
```

1.4. 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;
}
```

1.5. 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. Union-Find.

```
#define MAX_SIZE 26
int parent[MAX_SIZE],rank[MAX_SIZE];

void Make_Set(const int x){
   parent[x] = x; rank[x] = 0;
}

int Find(const int x){
   if(parent[x]!=x) parent[x] = Find(parent[x]);
   return parent[x];
}

void Union(const int x, const int y){
   int PX = Find(x),PY = Find(y);

   if(rank[PX]>rank[PY]) parent[PY] = PX;
   else{
      parent[PX] = PY;
      if(rank[PX]==rank[PY]) ++rank[PY];
   }
}
```

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 biconnected_components(){
   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. 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;
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;
      int aux;

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

      C[num_scc] = com;
      ++num_scc;
   }
}

void build_scc(){
   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);
}
```

2.7. 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);
    }
}
```

2.8. 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;
}
```

2.9. Ordenamiento Topológico.

```
#define MAX_V 100000
#define MAX_E 100000

int V,E,indeg[MAX_V],topo_pos[MAX_V];
int last[MAX_V],next[MAX_E],to[MAX_E];
int Q[MAX_V],head,tail;

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

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

void topological_sort(){
   head = tail = 0;

   for(int i = 0;i<V;++i){
      if(indeg[i]==0){
         topo_pos[i] = tail;
         Q[tail++] = i;
      }
   }

   while(head!=tail){
      int u = Q[head++];

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

         if(indeg[v]==0){
            topo_pos[v] = tail;
            Q[tail++] = v;
         }
```

```
      }
   }
}
```

2.10. Diámetro de un árbol.

```
#define MAX_SIZE 100
bool visited[MAX_SIZE];
int prev[MAX_SIZE];

int most_distant(int s){
   queue<int> Q;
   Q.push(s);

   memset(visited,false,sizeof(visited));
   visited[s] = true;
   prev[s] = -1;

   int ans = s;

   while(!Q.empty()){
      int aux = Q.front();
      Q.pop();

      ans = aux;

      for(int i=L[aux].size()-1;i>=0;--i){
         int v = L[aux][i];
         if(visited[v]) continue;
         visited[v] = true;
         Q.push(v);
         prev[v] = aux;
      }
   }

   return ans;
}
```

2.11. 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;
            }
      }
}
```

2.12. Bipartite matching (Hopcroft Karp).

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

int V1,V2,l[MAX_V1],r[MAX_V2];
int E,to[MAX_E],next[MAX_E],last[MAX_V1];
bool visited[MAX_V1];

void init(){
      memset(last,-1,sizeof(int)*V1);
      E = 0;
}

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

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

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

      if(r[v]==-1 || pairup(r[v])){
         l[u] = v;
         r[v] = u;
         return true;
      }
   }

   return false;
}

int hopcroft_karp(){
   bool change = true;
   memset(l,-1,sizeof(int)*V1);
   memset(r,-1,sizeof(int)*V2);

   while(change){
      change = false;
      memset(visited,false,sizeof(bool)*V1);

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

   int ret = 0;

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

   return ret;
}
```

2.13. Algoritmo húngaro.

```
#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;
}
```

2.14. General matching (Gabow).

```
#define MAXV 200
#define MAXE 19900

int prev_edge[MAXE], v[MAXE], w[MAXE], last_edge[MAXV];
int type[MAXV], label[MAXV], first[MAXV], mate[MAXV], nedges;
bool g_flag[MAXV], g_souter[MAXV];

void g_init(){
   nedges = 0;
   memset(last_edge,-1,sizeof(last_edge));
}

void g_edge(int a, int b){
   prev_edge[nedges] = last_edge[a];
   v[nedges] = a;
   w[nedges] = b;
   last_edge[a] = nedges++;

   prev_edge[nedges] = last_edge[b];
   v[nedges] = b;
   w[nedges] = a;
   last_edge[b] = nedges++;
}

void g_label(int v, int join, int edge, queue<int> &outer){
   if(v==join) return;
   if(label[v]==-1) outer.push(v);

   label[v] = edge;
   type[v] = 1;
   first[v] = join;

   g_label(first[label[mate[v]]], join, edge, outer);
}

void g_augment(int _v, int _w){
   int t = mate[_v];
   mate[_v] = _w;
   if(mate[t]!=_v) return;
   if(label[_v]==-1) return;

   if(type[_v]==0){
      mate[t] = label[_v];
      g_augment(label[_v], t);
   }else if(type[_v]==1){
      g_augment(v[label[_v]], w[label[_v]]);
      g_augment(w[label[_v]], v[label[_v]]);
   }
}

int gabow(int n){
   memset(mate,-1,sizeof(mate));
   memset(first,-1,sizeof(first));

   int u = 0, ret = 0;

   for(int z = 0;z<n;++z){
      if(mate[z]!=-1) continue;

      memset(label,-1,sizeof(label));
      memset(type,-1,sizeof(type));
      memset(g_souter,0,sizeof(g_souter));

      label[z] = -1; type[z] = 0;

      queue<int> outer;
      outer.push(z);

      bool done = false;

      while(!outer.empty()){
         int x = outer.front(); outer.pop();

         if(g_souter[x]) continue;
         g_souter[x] = true;

         for(int i = last_edge[x];i!=-1;i = prev_edge[i]) {
```

```
            if(mate[w[i]]==-1 && w[i]!=z){
               mate[w[i]] = x;
               g_augment(x, w[i]);
               ++ret;

               done = true;
               break;
            }

            if(type[w[i]]==-1){
               int v = mate[w[i]];
               if(type[v] == -1){
                  type[v] = 0;
                  label[v] = x;
                  outer.push(v);
                  first[v] = w[i];
               }
               continue;
            }

            int r = first[x], s = first[w[i]];
            if(r==s) continue;

            memset(g_flag,0,sizeof(g_flag));
            g_flag[r] = g_flag[s] = true;

            while(true){
               if(s!=-1) swap(r, s);
               r = first[label[mate[r]]];
               if(g_flag[r]) break;
               g_flag[r] = true;
            }

            g_label(first[x], r, i, outer);
            g_label(first[w[i]], r, i, outer);

            for(int c = 0;c<n;++c)
               if(type[c]!=-1 && first[c]!=-1 && type[first[c]]!=-1)
                  first[c] = r;
         }
         if(done) break;
      }
   }
   return ret;
}
```

2.15. Flujo máximo (Dinic).

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

   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], q = new int[MAX_V];
      dist = new int[MAX_V], now = new int[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++;
   }

   bool bfs(){
      fill(dist,dist+MAX_V,-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]>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;
   }

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

      while(bfs()){
         for(int i = 0;i<MAX_V;++i) now[i] = last[i];

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

      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 capacity, cost_type cst){
   to[E] = v, cap[E] = capacity;
   cost[E] = cst, next[E] = last[u];
   last[u] = E++;
   to[E] = u, cap[E] = 0;
   cost[E] = -cst, next[E] = last[v];
   last[v] = E++;
}

bool 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];
    return stop;
}

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

    if(!BellmanFord(s,t)){
        printf("Ciclo negativo de capacidad infinita");
        return;
    }

    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 vrtice 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 (No dirigidos).

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.

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. Suffix array.

```
#define MAX_LEN 40000
#define ALPH_SIZE 123

char A[MAX_LEN+1];
int N,pos[MAX_LEN],rank[MAX_LEN];
int cont[MAX_LEN],next[MAX_LEN];
bool bh[MAX_LEN+1],b2h[MAX_LEN+1];

void build_suffix_array(){
   N = strlen(A);

   memset(cont,0,sizeof(cont));

   for(int i = 0;i<N;++i) ++cont[A[i]];
   for(int i = 1;i<ALPH_SIZE;++i) cont[i] += cont[i-1];
   for(int i = 0;i<N;++i) pos[--cont[A[i]]] = i;

   for(int i = 0;i<N;++i){
      bh[i] = (i==0 || A[pos[i]]!=A[pos[i-1]]);
      b2h[i] = false;
   }

   for(int H = 1;H<N;H <<= 1){
      int buckets = 0;

      for(int i = 0,j;i<N;i = j){
         j = i+1;

         while(j<N && !bh[j]) ++j;
         next[i] = j;
         ++buckets;
      }

      if(buckets==N) break;

      for(int i = 0;i<N;i = next[i]){
         cont[i] = 0;
         for(int j = i;j<next[i];++j)
            rank[pos[j]] = i;
      }

      ++cont[rank[N-H]];
      b2h[rank[N-H]] = true;
```

```
      for(int i = 0;i<N;i = next[i]){
         for(int j = i;j<next[i];++j){
            int s = pos[j]-H;

            if(s>=0){
               int head = rank[s];
               rank[s] = head+cont[head];
               ++cont[head];
               b2h[rank[s]] = true;
            }
         }

         for(int j = i;j<next[i];++j){
            int s = pos[j]-H;

            if(s>=0 && b2h[rank[s]]){
               for(int k = rank[s]+1;!bh[k] && b2h[k];++k)
                  b2h[k] = false;
            }
         }
      }

      for(int i = 0;i<N;++i){
         pos[rank[i]] = i;
         bh[i] |= b2h[i];
      }
   }

   for(int i = 0;i<N;++i) rank[pos[i]] = i;
}

int height[MAX_LEN];
// height[i] = lcp(pos[i],pos[i-1])

// Complejidad : O(n)
void getHeight(){
   height[0] = 0;

   for(int i = 0,h = 0;i<N;++i){
      if(rank[i]>0){
         int j = pos[rank[i]-1];

         while(i+h<N && j+h<N && A[i+h]==A[j+h]) ++h;
         height[rank[i]] = h;
         if(h>0) --h;
      }
   }
}

// Queries para el Longest Common Prefix usando una Sparse Table.

#define LOG2_LEN 16

int RMQ[MAX_LEN][LOG2_LEN];

// Complejidad : O(nlgn)
void initialize_rmq(){
   for(int i = 0;i<N;++i) RMQ[i][0] = height[i];

   for(int j = 1;(1<<j)<=N;++j){
      for(int i = 0;i+(1<<j)-1<N;++i){
         if(RMQ[i][j-1]<=RMQ[i+(1<<(j-1))][j-1])
            RMQ[i][j] = RMQ[i][j-1];
         else
            RMQ[i][j] = RMQ[i+(1<<(j-1))][j-1];
      }
   }
}

// lcp(pos[x],pos[y])
int lcp(int x, int y){
   if(x==y) return N-rank[x];
   if(x>y) swap(x,y);

   int log = 0;
   while((1<<log)<=(y-x)) ++log;
   --log;

   return min(RMQ[x+1][log],RMQ[y-(1<<log)+1][log]);
}
```

3.3. Trie.

```
const int ALPH_SIZE = 58;

struct Node{
   int words; // numero de palabras que terminan en el nodo
```

```
   int prefixes; // numero de palabras que tienen como prefijo el camino al nodo
   vector<Node*> links; // enlaces a los nodos hijos

   Node(){
      words = prefixes = 0;
      links.resize(ALPH_SIZE,NULL);
   }
};

class Trie{
   public :

   Trie(){
      myRoot = new Node();
      myCount = 1;
   }

   bool contains(const string& s) const;
   int nodeCount() const;
   int countWords(const string& s) const;
   int countPrefixes(const string& s) const;
   int countRepeated(Node* t) const;
   void printAllWords(const Node* t, const string& s) const;

   void insert(const string s);

   private :

   Node* myRoot; // raiz del trie
   int myCount; // # nodos del trie
};

bool Trie::contains(const string& s) const{
   Node* t = myRoot;
   int len = s.size();

   for(int k=0;k<len;++k){
         if(t==NULL) return false;
      t = t->links[s[k]-'A'];
   }

   if(t==NULL) return false;
   return (t->words > 0);
}

int Trie::nodeCount() const{
   return myCount;
}

int Trie::countWords(const string& s) const{
   int len = s.size();
   Node* t = myRoot;

   for(int k=0;k<len;++k){
      if(t->links[s[k]-'A']==NULL) return 0;
      t = t->links[s[k]-'A'];
   }

   return t->words;
}

int Trie::countPrefixes(const string& s) const{
   int len = s.size();
   Node* t = myRoot;

   for(int k=0;k<len;++k){
      if(t->links[s[k]-'A']==NULL) return 0;
      t = t->links[s[k]-'A'];
   }

   return t->prefixes;
}

void Trie::printAllWords(const Node* t = myRoot, const string& s = "") const{
   if(t->words > 0) cout<<s<<endl;

   for(int k=0;k<ALPH_SIZE;++k)
      if(t->links[k]) printAllWords(t->links[k],s+char(k+'A'));
}

void Trie::insert(const string s){
   int len = s.size();
   Node* t = myRoot;

   for(int k=0;k<len;++k){
         if(t->links[s[k]-'A']==NULL){
            t->links[s[k]-'A'] = new Node();
            ++myCount;
         }
         t = t->links[s[k]-'A'];
         ++(t->prefixes);
   }
```

```
   ++(t->words);
}

int Trie::countRepeated(Node* t = myRoot) const{
      int aux = 0;

      if((t->words)>1) ++aux;

      for(int k=0;k<ALPH_SIZE;++k)
      if(t->links[k]) aux += countRepeated(t->links[k]);

      return aux;
}

string test[] = {"tree","trie","algo","assoc","all","also"};
```

```
int main(){
   Trie* myTrie;
   myTrie = new Trie();

   for(int i=0;i<6;++i){
      myTrie->insert(test[i]);
      cout<<myTrie->nodeCount()<<endl;
      myTrie->printAllWords();
      cout<<endl;
   }

   delete myTrie;

   return 0;
}
```

3.4. Aho-Corasick.

```
struct No {
   int fail;
   vector< pair<int,int> > out; // num e tamanho do padrao
   //bool marc; // p/ decisao
   map<char, int> lista;
   int next; // aponta para o proximo sufixo que tenha out.size > 0
};
No arvore[1000003]; // quantida maxima de nos
//bool encontrado[1005]; // quantidade maxima de padroes, p/ decisao
int qtdNos, qtdPadroes;

// Funcao para inicializar
void inic() {
   arvore[0].fail = -1;
   arvore[0].lista.clear();
   arvore[0].out.clear();
   arvore[0].next = -1;
   qtdNos = 1;
   qtdPadroes = 0;
   //arvore[0].marc = false; // p/ decisao
   //memset(encontrado, false, sizeof(encontrado)); // p/ decisao
}

// Funcao para adicionar um padrao
void adicionar(char *padrao) {
   int no = 0, len = 0;
```

```
   for (int i = 0 ; padrao[i] ; i++, len++) {
      if (arvore[no].lista.find(padrao[i]) == arvore[no].lista.end()) {
         arvore[qtdNos].lista.clear(); arvore[qtdNos].out.clear();
         //arvore[qtdNos].marc = false; // p/ decisao
         arvore[no].lista[padrao[i]] = qtdNos;
         no = qtdNos++;
      } else no = arvore[no].lista[padrao[i]];
   }
   arvore[no].out.push_back(pair<int,int>(qtdPadroes++,len));
}

// Ativar Aho-corasick, ajustando funcoes de falha
void ativar() {
   int no,v,f,w;
   queue<int> fila;
   for (map<char,int>::iterator it = arvore[0].lista.begin();
       it != arvore[0].lista.end() ; it++) {
      arvore[no = it->second].fail = 0;
      arvore[no].next = arvore[0].out.size() ? 0 : -1;
      fila.push(no);
   }
   while (!fila.empty()) {
      no = fila.front(); fila.pop();
      for (map<char,int>::iterator it=arvore[no].lista.begin();
          it!=arvore[no].lista.end(); it++) {
         char c = it->first;
```

```
            v = it->second;
            fila.push(v);
            f = arvore[no].fail;
            while (arvore[f].lista.find(c) == arvore[f].lista.end()) {
               if (f == 0) { arvore[0].lista[c] = 0; break; }
               f = arvore[f].fail;
            }
            w = arvore[f].lista[c];
            arvore[v].fail = w;
            arvore[v].next = arvore[w].out.size() ? w : arvore[w].next;
         }
      }
}

// Buscar padroes no aho-corasik
void buscar(char *input) {
   int v, no = 0;
   for (int i = 0 ; input[i] ; i++) {
      while (arvore[no].lista.find(input[i]) == arvore[no].lista.end()) {
         if (no == 0) { arvore[0].lista[input[i]] = 0; break; }
         no = arvore[no].fail;
      }
      v = no = arvore[no].lista[input[i]];
      // marcar os encontrados
      while (v != -1 /* && !arvore[v].marc */ ) { // p/ decisao
         //arvore[v].marc = true; // p/ decisao: nao continua a lista
         for (int k = 0 ; k < arvore[v].out.size() ; k++) {
            //encontrado[arvore[v].out[k].first] = true; // p/ decisao
            printf("Padrao %d na posicao %d\n", arvore[v].out[k].first,
                  i-arvore[v].out[k].second+1);
         }
         v = arvore[v].next;
      }
   }
   // for (int i = 0 ; i < qtdPadroes ; i++)
   //printf("%s\n", encontrado[i]?"y":"n"); // p/ decisao
}
```

3.5. Rotación lexicográfica mínima.

```
char s[100001];
scanf("%s",s);

int N = strlen(s),ans = 0,p = 1,l = 0;

while(p<N && ans+l+1<N){
   if(s[ans+l]==s[(p+l)%N]) ++l;
   else if(s[ans+l]<s[(p+l)%N]){
      p = p+l+1;
      l = 0;
   }else{
      ans = max(ans+l+1,p);
      p = ans+1;
      l = 0;
   }
}

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

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;
   }

   point operator + (const point &p) const{
      return point(x+p.x,y+p.y);
   }
```

```
  point operator - (const point &p) const{
    return point(x-p.x,y-p.y);
  }

  point operator * (double v) const{
    return point(x*v,y*v);
  }

  point perp(){
    return point(-y,x);
  }

  point normal(){
    return point(-y/abs(),x/abs());
  }

  double dot(const point &p) const{
    return x*p.x+y*p.y;
  }

  double abs2() const{
    return dot(*this);
  }

  double abs() const{
    return sqrt(abs2());
  }

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

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. Área y orientación de un triángulo.

```
double signed_area(const point &p1, const point &p2, const point &p3){
  return (p1.x*p2.y+p2.x*p3.y+p3.x*p1.y-p1.y*p2.x-p2.y*p3.x-p3.y*p1.x)/2;
}

bool ccw(const point &p1, const point &p2, const point &p3){
  return signed_area(p1,p2,p3)>-eps;
}
```

4.3. Fórmulas de triángulos.

```
double AreaHeron(double const &a, double const &b, double const &c){
  double s=(a+b+c)/2;
  return sqrt(s*(s-a)*(s-b)*(s-c));
}

double Circumradius(const double &a, const double &b, const double &c){
  return a*b*c/4/AreaHeron(a,b,c);
}

double Circumradius(const point &P1, const point &P2, const point &P3){
  return (P2-P1).abs()*(P3-P1).abs()*(P3-P2).abs()/4/fabs(signed_area(P1,P2,P3));
}

double Inradius(const double &a, const double &b, const double &c){
  return 2*AreaHeron(a,b,c)/(a+b+c);
}
```

4.4. Orientación de un polígono.

```
//verdadero : sentido anti-horario, Complejidad : O(n)
bool ccw(const vector<point> &poly){
   //primero hallamos el punto inferior ubicado ms 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]);
}
```

4.5. Área con signo.

```
//valor positivo : vrtices orientados en sentido antihorario
//valor negativo : vrtices 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;
}
```

4.6. Punto dentro de un polígono.

```
bool PointInsideConvexPolygon(const point &P, vector<point> &poly){
   int n = poly.size();
   if(!ccw(poly)) reverse(poly.begin(),poly.end());

   for(int i=1;i<=n;++i)
      if(!ccw(poly[i-1],poly[i%n],P))
         return false;

   return true;
}

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;
}
```

4.7. Distancia desde un punto.

```
//Distancia de un punto a una recta infinita
double PointToLineDist(const point &P, const line &L){
   return 2*fabs(signed_area(L.p1,L.p2,P))/(L.p2-L.p1).abs();
}

//Distancia de un punto a un segmento de recta
double PointToSegmentDist(const point &P, const line &L){
   point v=L.p2-L.p1,w=P-L.p1;
   double aux1=w.dot(v);
   if(aux1<eps) return (P-L.p1).abs();

   double aux2=v.dot(v);
   if(aux2<=aux1+eps) return (P-L.p2).abs();

   return PointToLineDist(P,L);
}
```

4.8. Intersección de líneas.

```
//verdadero : s se intersectan, I : punto de interseccin
bool lineIntersection(line &L1, line &L2, P &I){
   point n = (L2.p2-L2.p1).perp();

   double denom = n.dot(L1.p2-L1.p1);
   if(fabs(denom)<eps) return false; // las rectas son paralelas
   double t = n.dot(L2.p1-L1.p1)/denom;

   I = L1.p1 + (L1.p2-L1.p1)*t;

   return true;
}
```

4.9. 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);
}
```

4.10. Teorema de Pick.

```
El Teorema de Pick nos dice que : A=I+B/2-1, donde,

A = Area de un poligono de coordenadas enteras
I = Nmero de puntos enteros en su interior
B = Nmero de puntos enteros sobre sus bordes

Haciendo un cambio en la frmula : I=(2A-B+2)/2, tenemos una forma de calcular
el numero de puntos enteros en el interior del poligono

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.

4.11. Par de puntos más cercano.

```
#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;
}
```

4.12. 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;
}
```

5. Matemática

5.1. Algoritmo de Euclides.

```
struct EuclidReturn{
   int u,v,d;

   EuclidReturn(int _u, int _v, int _d){
      u = _u; v = _v; d = _d;
   }
};

EuclidReturn Extended_Euclid(int a, int b){
   if(b==0) return EuclidReturn(1,0,a);
   EuclidReturn aux = Extended_Euclid(b,a%b);
   int v = aux.u-(a/b)*aux.v;
   return EuclidReturn(aux.v,v,aux.d);
}

// ax = b (mod n)
int solveMod(int a,int b,int n){
   EuclidReturn aux = Extended_Euclid(a,n);
   if(b%aux.d==0) return ((aux.u * (b/aux.d))%n+n)%n;
   return -1;// no hay solucuin
}

// ax = 1(mod n)
int modular_inverse(int a, int n){
   EuclidReturn aux = Extended_Euclid(a,n);
   return ((aux.u/aux.d)%n+n)%n;
}
```

5.2. Criba para la función phi de Euler.

```
fill(factors,factors+N+1,0);
phi[1] = 1;

for(int i = 2;i<=N;i++){
   if(factors[i]==0){
      factors[i] = i;
      phi[i] = i-1;

      if(i<=sqrt(N)) for(int j = i*i;j<=N;j += i) factors[j] = i;
   }else{
      int aux = i,exp = 0;

      while(aux%factors[i]==0){
         aux /= factors[i];
         ++exp;
      }

      phi[i] = 1;

      for(int j = 0;j<exp;++j) phi[i] *= factors[i];
      phi[i] -= phi[i]/factors[i];
      phi[i] *= phi[aux];
   }
}
```

5.3. Teorema chino del resto.

```
// rem y mod tienen el mismo nmero 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;
}
```

5.4. 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;
}
```

5.5. 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 = (x+y)%c;
        y = (y<<1)%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;
}
```

5.6. 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];

    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];

   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;
}
```

5.7. Fast Fourier Transform.

```
#define lowbit(x) (((x) ^ (x-1)) & (x))
typedef complex<long double> Complex;

void FFT(vector<Complex> &A, int s){
   int n = A.size(), p = 0;

   while(n>1){
      ++p;
      n >>= 1;
   }

   n = (1<<p);

   vector<Complex> a = A;

   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] = a[rev];
   }

   Complex w,wn;

   for(int i = 1;i<=p;++i){
      int M = (1<<i), K = (M>>1);
      wn = Complex(cos(s*2.0*M_PI/(double)M), sin(s*2.0*M_PI/(double)M));

      for(int j = 0;j<n;j += M){
         w = Complex(1.0, 0.0);
         for(int l = j;l<K+j;++l){
            Complex t = w;
            t *= A[l + K];
            Complex u = A[l];
            A[l] += t;
            u -= t;
            A[l + K] = u;
            w *= wn;
         }
      }
   }

   if(s==-1){
      for(int i = 0;i<n;++i)
         A[i] /= (double)n;
   }
}

vector<Complex> FFT_Multiply(vector<Complex> &P, vector<Complex> &Q){
   int n = P.size()+Q.size();
   while(n!=lowbit(n)) n += lowbit(n);

   P.resize(n,0);
   Q.resize(n,0);

   FFT(P,1);
   FFT(Q,1);
```

```
   vector<Complex> R;
   for(int i=0;i<n;i++) R.push_back(P[i]*Q[i]);

   FFT(R,-1);

   return R;
}

// Para multiplicacin de enteros grandes
const long long B = 100000;
const int D = 5;
```

6. Estructuras de datos

6.1. BIT.

```
#define MAX_SIZE 20001
//los indices que se pueden usar van desde 1 hasta MAX_SIZE-1

void update(long long T[], int idx, int val){
   for(;idx<MAX_SIZE;idx+=(idx & -idx)) T[idx]+=val;
}

long long f(long long T[], int idx){
   long long sum = T[idx];

   if(idx>0){
      int z = idx-(idx & -idx);
      --idx;
      while(idx!=z){
         sum -= T[idx];
         idx -= (idx & -idx);
      }
   }

   return sum;
}

long long F(long long T[], int idx){
   long long sum = 0;
   for(;idx>0;idx -= (idx & -idx)) sum += T[idx];
   return sum;
}
```

6.2. Range Minimum Query.

```
#define MAX_N 100000
#define LOG2_MAXN 16
long long A[MAX_N];
int N,ind[(1<<(LOG2_MAXN+2))];

void initialize(int node, int s, int e){
   if(s==e) ind[node] = s;
   else{
      initialize(2*node+1,s,(s+e)/2);
      initialize(2*node+2,(s+e)/2+1,e);
      if(A[ind[2*node+1]]<=A[ind[2*node+2]]) ind[node] = ind[2*node+1];
      else ind[node] = ind[2*node+2];
   }
}

int query(int node, int s, int e, int a, int b){
   if(b<s || a>e) return -1;
   if(a<=s && e<=b) return ind[node];

   int ind1 = query(2*node+1,s,(s+e)/2,a,b);
   int ind2 = query(2*node+2,(s+e)/2+1,e,a,b);

   if(ind1==-1) return ind2;
   if(ind2==-1) return ind1;
   if(A[ind1]<=A[ind2]) return ind1;
   return ind2;
}
```

6.3. 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];

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];
}
```

6.4. Maximum Sum Segment Query.

```
#define MAX_N 100000
#define LOG2_MAXN 16
const long long INF = 10000000001LL;

int N,a[MAX_N];
long long c[MAX_N+1],int_min[1<<(LOG2_MAXN+2)],int_max[1<<(LOG2_MAXN+2)];
long long int_best[1<<(LOG2_MAXN+2)];

void build_tree(int node, int lo, int hi){
   if(lo==hi){
      if(lo!=0){
         int_min[node] = c[lo-1];
         int_max[node] = c[lo];
         int_best[node] = c[lo]-c[lo-1];
      }else{
         int_min[node] = 0;
         int_max[node] = 0;
         int_best[node] = 0;
      }
   }else{
      int mi = (lo+hi)>>1;
      build_tree(2*node+1,lo,mi);
      build_tree(2*node+2,mi+1,hi);

      int_min[node] = min(int_min[2*node+1],int_min[2*node+2]);
      int_max[node] = max(int_max[2*node+1],int_max[2*node+2]);
      int_best[node] = max(int_max[2*node+2]-int_min[2*node+1],
                  max(int_best[2*node+1],int_best[2*node+2]));
```

```
    }
}

void init(){
    c[0] = 0;
    for(int i = 0;i<N;++i) c[i+1] = c[i]+a[i];
    build_tree(0,0,N);
}

long long minPrefix;
int s,e;

long long tree_query(int node, int lo, int hi) {
    if (s<=lo && hi<=e) {
        long long ret = int_best[node];
        if (minPrefix!=INF) ret = max(ret,int_max[node]-minPrefix);
        minPrefix = min(minPrefix,int_min[node]);
        return ret;
    }else{
        int mi = (lo+hi)>>1;

        if(mi<s) return tree_query(2*node+2,mi+1,hi);
        else if(mi>=e) return tree_query(2*node+1,lo,mi);
        else{
            long long val1 = tree_query(2*node+1,lo,mi);
            long long val2 = tree_query(2*node+2,mi+1,hi);
            return max(val1,val2);
        }
    }
}

// Los indices van de 1 a N
long long solve_msq(int x, int y){
    minPrefix = INF;
    s = x; e = y;
    return tree_query(0,0,N);
}
```

6.5. Treap.

```
long long seed = 47;

long long rand(){
    seed = (seed * 279470273) % 4294967291LL;
    return seed;
}

typedef int treap_type;

class treap{
    public:

    treap_type value;
    long long priority;
    treap *left, *right;
    int sons;

    treap(treap_type value) : left(NULL), right(NULL), value(value), sons(0){
        priority = rand();
    }

    ~treap(){
        if(left) delete left;
        if(right) delete right;
    }
};

treap* find(treap* t, treap_type val){
    if(!t) return NULL;
    if(val == t->value) return t;

    if(val < t->value) return find(t->left, val);
    if(val > t->value) return find(t->right, val);
}

inline void rotate_to_right(treap* &t){
    treap* n = t->left;
    t->left = n->right;
    n->right = t;
    t = n;
}

inline void rotate_to_left(treap* &t){
    treap* n = t->right;
    t->right = n->left;
    n->left = t;
```

```
   t = n;
}

void fix_augment(treap* t){
   if(!t) return;
   t->sons = (t->left ? t->left->sons + 1 : 0) +
      (t->right ? t->right->sons + 1 : 0);
}

void insert(treap* &t, treap_type val){
   if(!t) t = new treap(val);
   else insert(val <= t->value ? t->left : t->right, val);

   if(t->left && t->left->priority > t->priority)
      rotate_to_right(t);
   else if(t->right && t->right->priority > t->priority)
      rotate_to_left(t);

   fix_augment(t->left); fix_augment(t->right); fix_augment(t);
}

inline long long get_priority(treap* t){
   return t ? t->priority : -1;
}

void erase(treap* &t, treap_type val){
   if(!t) return;
   if(t->value != val) erase(val < t->value ? t->left : t->right, val);
   else{
      if(!t->left && !t->right){
         delete t;
         t = NULL;
      }else{
         if(get_priority(t->left) < get_priority(t->right))
            rotate_to_left(t);
         else
            rotate_to_right(t);

         erase(t, val);
      }
   }

   fix_augment(t->left); fix_augment(t->right); fix_augment(t);
}

int getKth(treap* &t, int K){
   int left = (t->left==NULL? 0 : 1+t->left->sons);
   int right = (t->right==NULL? 0 : 1+t->right->sons);

   if(1+left==K) return t->value;
   else if(left<K) return getKth(t->right,K-1-left);
   return getKth(t->left,K);
}
```

7. Matrices

7.1. Exponenciación de matrices.

```
#define MAX_SIZE 100
#define MOD 10000
int size;

struct M{
   long long X[MAX_SIZE][MAX_SIZE];

   M(){}
}M0,aux1,aux2;

void mult(M &m, M &m1, M &m2){
   memset(m.X,0,sizeof(m.X));
   for(int i=0;i<size;i++)
      for(int j=0;j<size;j++)
         for(int k=0;k<size;k++)
            m.X[i][k]=(m.X[i][k]+m1.X[i][j]*m2.X[j][k])%MOD;
}

M exp(int n){
   if(n==1) return M0;
   if(n==0){
      for(int i=0;i<size;i++) fill(aux1.X[i],aux1.X[i]+size,0);
      for(int i=0;i<size;i++) aux1.X[i][i]=1;
      return aux1;
   }
```

```
    aux1=exp(n/2);

    for(int i=0;i<size;i++) fill(aux2.X[i],aux2.X[i]+size,0);
    for(int i=0;i<size;i++) aux2.X[i][i]=1;

    mult(aux2,aux1,aux1);

    if(n%2==1){
        mult(aux1,aux2,M0);
        return aux1;
    }

    return aux2;
}

// para exponente n escrito en base 2<=b<=10
M exp(string &n, int b){
    M P[b+1];
    for(int i=0;i<=b;++i) P[i] = exp(i);

    int L = n.size();
    M ret;
    memset(ret.X,0,sizeof(ret.X));
    for(int i=0;i<size;++i) ret.X[i][i] = 1;

    int aux = 0;
    for(int i=0;i<L;++i){
        int x = n[i]-'0';
        M0 = ret;
        ret = exp(b);

        aux1 = ret;
        mult(ret,aux1,P[x]);
    }

    return ret;
}
```

7.2. Determinante.

```
#define MAX_SIZE 500
int size;

struct M{
    double X[MAX_SIZE][MAX_SIZE];
    M(){}
};

const double eps = 1e-7;

double determinant(M M0){
    double ans = 1,aux;
    bool found;

    for(int i = 0,r = 0;i<size;i++){
        found = false;

        for(int j = r;j<size;j++)
            if(fabs(M0.X[j][i])>eps){
                found = true;
                if(j>r) ans = -ans;
                else break;

                for(int k = 0;k<size;k++) swap(M0.X[r][k],M0.X[j][k]);
                break;
            }

        if(found){
            for(int j = r+1;j<size;j++){
                aux = M0.X[j][i]/M0.X[r][i];
                for(int k = i;k<size;k++) M0.X[j][k] -= aux*M0.X[r][k];
            }

            r++;
        }else return 0;
    }

    for(int i = 0;i<size;i++) ans *= M0.X[i][i];
    return ans;
}
```

7.3. Elimación gaussiana módulo MOD.

```
#define MAX_R 500
#define MAX_C 500
int R,C,MOD;

struct M{
   int X[MAX_R][MAX_C];
   M(){}
};

//cuidado con overflow
int exp(int a, int n){
   if(n==0) return 1;
   if(n==1) return a;

   int aux=exp(a,n/2);
   if(n&1) return ((long long)a*(aux*aux)%MOD)%MOD;
   return (aux*aux)%MOD;
}
void GaussianElimination(M &M0){
   int aux;
   bool found;

   for(int I = 0,r = 0;r<R && i<C;++i){
      found=false;

      for(int j = r;j<R;++j){
         if(M0.X[j][i]>0){
            found=true;
            if(j==r) break;

            for(int k = i;k<C;++k) swap(M0.X[r][k],M0.X[j][k]);
            break;
         }
      }

      if(found){
         aux = modular_inverse(M0.X[r][i],MOD);

         for(int j = i;j<C;++j) M0.X[r][j] = (M0.X[r][j]*aux)%MOD;

         for(int j = r+1;j<R;++j){
            aux = MOD-M0.X[j][i];
            for(int k = i;k<C;++k)
               M0.X[j][k] = (M0.X[j][k]+aux*M0.X[r][k])%MOD;
         }

         ++r;
      }
   }

   //Recuciendo hacia atrs
   for(int I = R-1;i>0;--i)
      for(int j = 0;j<i;++j)
         M0.X[j][R] = (M0.X[j][R]+(MOD-M0.X[j][i])*M0.X[i][R])%MOD;
}
```

8. Mathematical facts

8.1. Números de Catalan.

están definidos por la recurrencia:

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

Una fórmula cerrada para los números de Catalán es:

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

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 formas de dividir n elementos en k conjuntos.

$$\left\{ {n \atop k} \right\} = k \left\{ {n-1 \atop k} \right\} + \left\{ {n-1 \atop k-1} \right\}$$

Además:

$$\left\{ {n \atop k} \right\} = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^n$$

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	5	6	7	8	9	10	11	12
$\mathcal{B}_x$	52	203	877	4.140	21.147	115.975	678.570	4.213.597

8.5. **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:

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

8.6. **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.	$\left\{ {n \atop 1} \right\} + \ldots + \left\{ {n \atop x} \right\}$	$[n \leq x]$	$\left\{ {n \atop k} \right\}$
indist.	indist.	$p_1(n) + \ldots p_x(n)$	$[n \leq x]$	$p_x(n)$

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.

8.7. **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

$$v - e + f = 2$$

ACM ICPC Team Reference - Contenidos

Universidad Nacional de Ingeniería - FIIS

Contents