17 de Setembro de 2009 20:15

Conteúdo

1 Grafos

1.1 Arvore Geradora Minima (Kruskal)

```
Autor: Davi Costa
Complexidade: O(m*logm)
Tempo de implementacao: ?
Testes: uva.10034
Dependencias: Nenhuma
Descricao: Encontra Arvore Geradora Minima

#define MAXN 101
#define MAXM MAXN*MAXN

int orig[MAXM]; //origem da aresta i
int dest[MAXM]; //destino da aresta i
```

```
int custo[MAXM]; //custo da aresta i
//bool intree[MAXM]; //caso precise da arvore ao final
int ord[MAXM]; //auxiliar
int id[MAXN], sz[MAXN]; //uf auxiliar

void ufinit(int n) {
  for (int i = 0; i < n; i++)
    id[i] = i, sz[i] = 1;
}

int uffind(int i) {
  if (i == id[i]) return i;
  return id[i] = uffind(id[i]);
}

void ufunion(int v, int w) {
  v = uffind(v); w = uffind(w);
  if (v == w) return;
  if (sz[v] > sz[w]) swap(v,w);
  id[v] = w;
  if (sz[v] == sz[w]) sz[w]++;
}

bool edgecmp(int i, int j) {
  return (custo[i] < custo[j]);
}

int kruskal(int n, int ne) {
  int cost = 0;
  //memset(intree,0,sizeof(intree));
  for (int i = 0; i < ne; i++) ord[i] = i;
  sort(ord,&ord[ne],edgecmp);
  ufinit(n);
  int k = 0;
  for (int i = 0; i < ne && k < n; i++) {
    int v = ord[i];
    if (uffind(orig[v]) != uffind(dest[v])) {
      //intree[v] = true;
      ufunion(orig[v],dest[v]);
      cost += custo[v]; k++;
    }
  }
  return cost;
}
```

```
/*Exemplo de Uso*/
int main() {
  int n, m;
  while (scanf("%d %d",&n,&m) == 2 && n != 0) {
    for (int i = 0; i < m; i++) {
      scanf("%d %d %d",&orig[m],&dest[m],&custo[m]);
    }
    printf("%d\n",kruskal(n,m));
  }
}
```

1.2 Arvore Geradora Minima (Prim)

```
Autor: Davi Costa
Complexidade: O(m*logn)
Tempo de implementacao: ?
Testes: uva.10034
Dependencias: Nenhuma
Descricao: Encontra Arvore Geradora Minima

#include <iostream>
#include <queue>
#include <limits.h>

using namespace std;

#define MAXN 101 //numero maximo de vertices
#define INF INT_MAX //nao ha perigo de overflow

int adj[MAXN][MAXN]; //lista de adj
int custo[MAXN][MAXN]; //tamanho das arestas de adj
int nadj[MAXN]; //grau de cada vertice
int pai[MAXN]; //para reconstruir o caminho
int dist[MAXN]; //distancia de cada vertice a arvore
bool used[MAXN];

/*
n: numero de vertices, s: origem (opcional)
retorna peso total da arvore
*/
int prim(int n, int s = 0) {
  priority_queue<pair<int, int> > q;
  int a,v;
```

```
  int cost, nv = 0;
  int ret = 0;
  memset(pai,-1,sizeof(pai));
  memset(used,0,sizeof(used));
  for (int i = 0; i < n; i++) dist[i] = INF;
  dist[s] = 0;
  pai[s] = s;
  q.push(make_pair(0,s));
  while(!q.empty() && nv < n) {
    a = q.top().second;
    q.pop();
    if (used[a]) continue;
    ret += dist[a];
    used[a] = true;
    nv++;
    for (int i = 0; i < nadj[a]; i++) {
      v = adj[a][i];
      if (used[v]) continue;
      cost = custo[a][i];
      if (cost >= dist[v]) continue;
      dist[v] = cost;
      q.push(make_pair(-1*cost,v));
      pai[v] = a;
    }
  }
  return ret;
}

/**** Exemplo simples de uso ****/
int main() {
  int n, m;
  int from, to, cost;
  while (scanf("%d %d",&n,&m) == 2 && n != 0) {
    memset(nadj,0,sizeof(nadj));
    for (int i = 0; i < m; i++) {
      scanf("%d %d %d",&from,&to,&cost);
      custo[from][nadj[from]] = custo[to][nadj[to]] = cost;
      adj[from][nadj[from]++] = to;
      adj[to][nadj[to]++] = from;
    }
    printf("%d\n",prim(n));
  }
  return 0;
}
```

1.3 Bellman Ford

```
Autor: Igor Assis/Davi Costa
Complexidade: O(n*m)
Tempo de implementacao: ?
Testes: uva.10806, uva.423
Dependencias: Sem dependencias
Descricao: Caminho minimo com pesos negativos

#define MAXN 100 //Numero maximo de vertices
#define MAXM 10000 //Numero maximo de arestas
#define INF 0x3f3f3f3f


/* aresta (u,v) com peso w:
   orig[i] = u, dest[i] = v, peso[i] = w
   d[u], distancia da origem s ao vertice u
*/

int orig[MAXM], dest[MAXM], peso[MAXM], d[MAXN], pai[MAXN];
/*
s: origem, n: numero de vertices, m: numero de arestas
retorna 1 se o grafo nao tem ciclo negativo, 0 c.c
*/

int bellman_ford(int s, int n, int m) {
  int i, j;
  memset(pai,-1,sizeof(pai));
  pai[s] = s;
  for (i = 0; i < n; i++)
    d[i] = INF;
  d[s] = 0;
  for (i = 0; i < n-1; i++)
    for (j = 0; j < m; j++) {
      int u = orig[j], v = dest[j], w = peso[j];
      if (d[v] > d[u]+w) {
d[v] = d[u]+w;
pai[v] = u;
      }
    }
  for (j = 0; j < m; j++) {
    int u = orig[j], v = dest[j], w = peso[j];
    if (d[v] > d[u]+w) return 0;
  }
```

```
  return 1;
}

/*Exemplo de Uso*/

int main() {
  int n, m;
  int from, to, cost;
  int origem, destino;
  while (scanf("%d %d",&n,&m) == 2 && n != 0) {
    int k = 0;
    for (int i = 0; i < m; i++) {
      scanf("%d %d %d",&from,&to,&cost);
      orig[k] = from;
      dest[k] = to;
      peso[k] = cost;
      k++;
      orig[k] = to;
      dest[k] = from;
      peso[k] = cost;
      k++;
    }
    scanf("%d %d",&origem,&destino);
    bellman_ford(origem,n,m);
    printf("%d\n",d[destino]);
  }
  return 0;
}
```

1.4 Pontes e Pontos de Articulação

```
Autor: Igor Assis
Complexidade: O(n+m)
Tempo de implementacao: ?
Testes: uva.796 spojbr.TUBOS
Dependencias: Nenhuma
Descricao: Encontra as pontes, os pontos de articulacao e as
componentes biconexas (comecando em 1)

#include <cstring>
#include <stack>
#include <algorithm>
using namespace std;

#define MAXN 1024
#define MAXM 1024*1024

#define VIZ(u, i) (orig[adj[u][i]] != (u) ? \
   orig[adj[u][i]] : dest[adj[u][i]])

int n, m;
int adj[MAXN][MAXN], nadj[MAXN];
int orig[MAXM], dest[MAXM];

int low[MAXN], part[MAXN], ponte[MAXM], ncomp, comp[MAXM];
int dt;

int vis[MAXN];
stack<int> stck;

int dfsbcc(int u, int p) {
  int ch = 0;
  vis[u] = dt++;
  low[u] = vis[u];
  for (int i = 0; i < nadj[u]; i++) {
    int e = adj[u][i], v = VIZ(u, i);
    if (!vis[v]) {
      stck.push(e);
      dfsbcc(v, u); ch++;
      low[u] = min(low[u], low[v]);
      if (low[v] >= vis[u]) {
part[u] = 1;
ncomp++;
while (stck.top() != e) {
  comp[stck.top()] = ncomp;
  stck.pop();
}
comp[stck.top()] = ncomp; stck.pop();
      }
      if (low[v] == vis[v]) ponte[e] = 1;
    } else if (v != p) {
      if (vis[v] < vis[u]) stck.push(e);
      low[u] = min(low[u], vis[v]);
    }
  }
  return ch;
```

```
}

void bcc() {
  memset(low, 0, sizeof(low));
  memset(vis, 0, sizeof(vis));
  memset(part, 0, sizeof(part));
  memset(ponte, 0, sizeof(ponte));
  memset(comp, 0, sizeof(comp));
  dt = 1;
  ncomp = 0;
  for (int i = 0; i < n; i++)
    if (!vis[i])
      part[i] = dfsbcc(i, -1) >= 2;
}

/**** Exemplo simples de uso ****/
#include <cstdio>

int main(void){
  int i, j;

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

  memset(nadj, 0, sizeof(nadj));
  for (i = 0; i < m; i++) {
    int u, v;
    scanf("%d%d", &u, &v);
    orig[i] = u; dest[i] = v;
    adj[u][nadj[u]++] = adj[v][nadj[v]++] = i;
  }

  bcc();

  printf("Pontos de Articulacao:");
  for (i = 0; i < n; i++)
    if (part[i]) printf(" %d", i);
  printf("\n");

  printf("Pontes:");
  for (i = 0; i < n; i++)
    for (j = 0; j < nadj[i]; j++)
      if (ponte[adj[i][j]] && i < VIZ(i, j))
printf(" (%d %d)", i, VIZ(i, j));
  printf("\n");

  printf("Componentes:\n");
  for (i = 0; i < m; i++)
    printf("comp[%d] = %d\n", i, comp[i]);
  printf("\n");

  return 0;
}
```

1.5 Emparelhamento máximo e cobertura mínima

```
Autor: Igor Assis
Complexidade: O(n*m)
Tempo de implementacao: ?
Testes: uva.11419 uva.10080
Descricao: Encontra um emparelhamento máximo e
uma cobertura de aresta por vértice mínima em grafo
bipartido.

#include <cstring>

/* O numero maximo total de vertices, somando as duas particoes */
#define MAXN 2024

int nU, nV, n;
int adj[MAXN][MAXN], nadj[MAXN];

int conj[MAXN], cover[MAXN], vis[MAXN];

/* Emparelhamento maximo em grafo bipartido
 *
 * +---+      +---+
 * | U |======| V |
 * +---+      +---+
 *
 * U = {0...nU-1}
 * V = {nU...nU+nV-1}
 *
 * n = nU + nV
 */
int aumenta(int u) {
  int i;
  for (i = 0; i < nadj[u]; i++) {
    int v = adj[u][i];
```

```
    if (vis[v]) continue; vis[v] = 1;
    if (conj[v] == -1 || aumenta(conj[v])) {
      conj[v] = u;
      conj[u] = v;
      return 1;
    }
  }
  return 0;
}

int maxbpm() {
  int i;
  int res = 0;
  memset(conj, -1, sizeof(conj));
  for (i = 0; i < nU; i++) {
    memset(vis, 0, sizeof(vis));
    if (aumenta(i)) res++;
  }
  return res;
}

/*
 * Cobertura minima de arestas em grafo bipartido
 */
void reach(int u) {
  int i;
  vis[u] = 1;
  for (i = 0; i < nadj[u]; i++) {
    int v = adj[u][i];
    if (!vis[v] && ((conj[u] != v && u < nU) ||
    (conj[u] == v && u >= nU))) reach(v);
  }
}

int minbpec() {
  int i;
  int res = maxbpm();
  memset(vis, 0, sizeof(vis));
  for (i = 0; i < nU; i++)
    if (!vis[i] && conj[i] == -1) reach(i);
  /* C = (U \ Rm) \/ (V /\ Rm) */
  memset(cover, 0, sizeof(cover));
  for (i = 0; i < nU; i++)
    if (!vis[i]) cover[i] = 1;
  for (i = nU; i < nU+nV; i++)
    if (vis[i]) cover[i] = 1;
  return res;
}

/**** Exemplo simples de uso ****/
#include <cstdio>

int main() {
  int i, u, v, m;

  scanf("%d%d%d", &nU, &nV, &m);
  for (i = 0; i < m; i++) {
    scanf("%d%d", &u, &v);
    v += nU;
    adj[u][nadj[u]++] = v;
    adj[v][nadj[v]++] = u;
  }

  printf("Matching Maximo/Cobertura Minima: %d\n", minbpec());
  printf("Arestas do Matching:\n");
  for (i = 0; i < nU; i++)
    if (conj[i] != -1)
      printf("%d %d ", i, conj[i]);
  printf("\nVertices na cobertura:\n");
  for (i = 0; i < n; i++)
    if (cover[i])
      printf("%d ", i);
  printf("\n");

  return 0;
}
```

1.6 Dijkstra STL

```
Autor: Davi Costa
Complexidade: O(m*logn)
Tempo de implementacao: ?
Testes: spoj.SHPATH, uva.423
Dependencias: Nenhuma
Descricao: Encontra caminho minimo em grafos com pesos positivos

#include <iostream>
#include <queue>
```

```
#include <limits.h>
#include <cstring>

using namespace std;

#define MAXN 101 //numero maximo de vertices
#define INF INT_MAX //nao ha perigo de overflow

int adj[MAXN][MAXN]; //lista de adj
int custo[MAXN][MAXN]; //tamanho das arestas de adj
int nadj[MAXN]; //grau de cada vertice
int pai[MAXN]; //para reconstruir o caminho
int dist[MAXN]; //distancia da origem ateh cada vertice
bool used[MAXN];

/*
n: numero de vertices, s: origem, to: destino (opcional)
retorna: distancia ate "to"
caso "to" nao seja passado retorna a maior distancia
*/
int dijkstra(int n, int s, int to = -1) {
  priority_queue<pair<int, int> > q;
  int a,v;
  int cost, nv = 0;
  memset(pai,-1,sizeof(pai));
  memset(used,0,sizeof(used));
  for (int i = 0; i < n; i++) dist[i] = INF;
  dist[s] = 0;
  pai[s] = s;
  q.push(make_pair(0,s));
  while(!q.empty() && nv < n) {
    a = q.top().second;
    q.pop();
    if (used[a]) continue;
    //if (to == a) return dist[a]; se soh quiser dist[to]
    used[a] = true;
    nv++;
    for (int i = 0; i < nadj[a]; i++) {
      v = adj[a][i];
      if (used[v]) continue;
      cost = dist[a] + custo[a][i];
      if (cost >= dist[v]) continue;
      dist[v] = cost;
      q.push(make_pair(-1*cost,v));
      pai[v] = a;
    }
  }
  if (to == -1) to = a;
  return dist[to];
}

/**** Exemplo simples de uso ****/
int main() {
  int n, m;
  int origem,destino;
  int from, to, cost;
  while (scanf("%d %d",&n,&m) == 2 && n != 0) {
    scanf("%d %d",&origem,&destino);
    memset(nadj,0,sizeof(nadj));
    for (int i = 0; i < m; i++) {
      scanf("%d %d %d",&from,&to,&cost);
      custo[from][nadj[from]] = custo[to][nadj[to]] = cost;
      adj[from][nadj[from]++] = to;
      adj[to][nadj[to]++] = from;
    }
    printf("%d\n",dijkstra(n,from,to));
  }
  return 0;
}
```

1.7 Dijkstra com heap explícito

```
Autor: shygypsy
Complexidade: O(m*logn)
Tempo de implementacao: ?
Testes: spoj.SHPATH, uva.423, uva.10594 (mfmc)
Dependencias: Nenhuma
Descricao: Encontrar caminho minimo em grafos com pesos positivs
Se demonstrou ligeiramente mais rapido que a outra
versao (2.4s -> 3.8s), porem bem mais confusa


#include <iostream>
#include <cstring>

using namespace std;

#define NN 110
```

```
int custo[NN][NN], adj[NN][NN], nadj[NN];
int d[NN], q[NN], inq[NN], pai[NN], qs;

#define CLR( x, v ) memset( x, v, sizeof( x ) )
#define BUBL { \
    t = q[i]; q[i] = q[j]; q[j] = t; \
    t = inq[q[i]]; inq[q[i]] = inq[q[j]]; inq[q[j]] = t; }

int dijkstra( int n, int s, int t )
{
    CLR( d, 9 ); CLR( inq, -1 ); CLR( pai, -1 );
    d[s] = qs = 0;
    inq[q[qs++] = s] = 0;
    pai[s] = -2;

    while( qs )
    {
        // get the minimum from the q
        int u = q[0]; inq[u] = -1;

        // bubble down
        q[0] = q[--qs];
        if( qs ) inq[q[0]] = 0;
        for(int i = 0,j = 2*i + 1,t;j < qs; i = j,j = 2*i + 1)
        {
            if( j + 1 < qs && d[q[j + 1]] < d[q[j]] ) j++;
            if( d[q[j]] >= d[q[i]] ) break;
            BUBL;
        }

        // relax neighbours
        for(int k = 0,v = adj[u][k]; k < nadj[u];v = adj[u][++k])
        {
            int newd = d[u] + custo[u][k];
            if( newd < d[v] )
            {
                d[v] = newd;
                pai[v] = u;
                if( inq[v] < 0 ) { inq[q[qs] = v] = qs; qs++; }

                // bubble up
                for( int i = inq[v], j = ( i - 1 )/2, t;
                     d[q[i]] < d[q[j]]; i = j, j = ( i - 1 )/2 )
                    BUBL;
            }
        }
    }

    return d[t];
}

//------- TESTING --------
#include <stdio.h>
int main()
{
    int n, m;
    while( scanf( " %d %d\n", &n, &m ) == 2 )
    {
        memset( nadj, 0, sizeof( nadj ) );
        while( m-- )
        {
            int u, v, w;
            scanf( " %d %d %d", &u, &v, &w );
            custo[u][nadj[u]] = adj[v][nadj[v]] = w;
            adj[u][nadj[u]++] = v;
            adj[v][nadj[v]++] = u;
        }

        int ans = dijkstra( n, 0, n - 1 );

        printf( "%d\n", ans );
    }
    return 0;
}
```

1.8 Emparelhamento Maximo Geral (Edmonds)

```
Autor: Davi Costa
Complexidade: O(n^3)
Uso: CUIDADO - Nao utilizar o vertice 0
- Para cada aresta 'i' (sem direcao) u-v,
faca from[i] = u, to[i] = v e coloque i
na lista de adjacencia de ambos u e v.
- n e m devem ser utilizados obrigatoriamente.
- E() retorna o tamanho do emparalhamento (# de casais).
- mate[v] quando diferente de 0 indica que o vertice v esta
casado com mate[v]
```

```
Testes:
- uva-11439 (#4 melhor tempo)
- nuevo-4130 (#2 melhor tempo)
- spojbr-ENGENHAR

#include <iostream>
#include <algorithm>
#include <queue>
#include <cstring>

using namespace std;

#define MAXN 110
#define MAXM MAXN*MAXN

int n,m;

int mate[MAXN], first[MAXN], label[MAXN];
int adj[MAXN][MAXN], nadj[MAXN], from[MAXM], to[MAXM];

queue<int> q;

#define OUTER(x) (label[x] >= 0)

void L(int x, int y, int nxy) {
  int join, v, r = first[x], s = first[y];
  if (r == s) return;
  nxy += n + 1;
  label[r] = label[s] = -nxy;
  while (1) {
    if (s != 0) swap(r,s);
    r = first[label[mate[r]]];
    if (label[r] != -nxy) label[r] = -nxy;
    else {
      join = r;
      break;
    }
  }
  v = first[x];
  while (v != join) {
    if (!OUTER(v)) q.push(v);
    label[v] = nxy; first[v] = join;
    v = first[label[mate[v]]];
  }
  v = first[y];
  while (v != join) {
    if (!OUTER(v)) q.push(v);
    label[v] = nxy; first[v] = join;
    v = first[label[mate[v]]];
  }
  for (int i = 0; i <= n; i++) {
    if (OUTER(i) && OUTER(first[i])) first[i] = join;
  }
}

void R(int v, int w) {
  int t = mate[v]; mate[v] = w;
  if (mate[t] != v) return;
  if (label[v] >= 1 && label[v] <= n) {
    mate[t] = label[v];
    R(label[v],t);
    return;
  }
  int x = from[label[v]-n-1];
  int y = to[label[v]-n-1];
  R(x,y); R(y,x);
}

int E() {
  memset(mate,0,sizeof(mate));
  int r = 0;
  bool e7;
  for (int u = 1; u <= n; u++) {
    memset(label,-1,sizeof(label));
    while (!q.empty()) q.pop();
    if (mate[u]) continue;
    label[u] = first[u] = 0;
    q.push(u); e7 = false;
    while (!q.empty() && !e7) {
      int x = q.front(); q.pop();
      for (int i = 0; i < nadj[x]; i++) {
int y = from[adj[x][i]];
if (y == x) y = to[adj[x][i]];
if (!mate[y] && y != u) {
  mate[y] = x; R(x,y);
  r++; e7 = true;
  break;
}
```

```
else if (OUTER(y)) L(x,y,adj[x][i]);
else {
  int v = mate[y];
  if (!OUTER(v)) {
    label[v] = x; first[v] = y;
    q.push(v);
  }
}
      }
    }
    label[0] = -1;
  }
  return r;
}


/*Exemplo simples de uso*/
int main() {
  int f,t;
  while (scanf("%d %d",&n,&m) == 2 && (n || m)) {
    memset(nadj,0,sizeof(nadj));
    for (int i = 0; i < m; i++) {
      scanf("%d %d",&f,&t);
      f++; t++; //nao utilizar o vertice 0
      adj[f][nadj[t]++] = i;
      adj[t][nadj[f]++] = i;
      from[i] = f; to[i] = t;
    }
    printf("O emparelhamento tem tamanho %d\n",E());
    for (int i = 1; i <= n; i++) {
      if (mate[i] > i) {//para nao imprimir 2 vezes
printf("%d casa com %d\n",i-1,mate[i]-1);
      }
    }
  }
  return 0;
}
```

1.9 Estrutura para grafos grandes, mas esparsos

```
Autor: Alexandre Kunieda
Complexidade: espaco O(V+E)
Tempo de implementacao: 3 minutos
Testes:
- OBI Batuira - utilizado com Dijkstra
- SPOJ PT07Z  - utilizado com Diametro de Arvore
Dependencias: Nenhuma
Descricao:
Uma estrutura que utiliza um vetor e da acesso as listas
de adjacencias dos vertices do grafo e gasta espaco O(V+E)
ATENCAO: Esta NAO eh a estrutura padrao para grafos!


#include <stdio.h>
#include <stdlib.h>

#define MAXn 10123
#define MAXm 1000123

struct grafo_t {
  /* lista de adjacencias, num vetor */
  struct adj_t {
    int v,c; /* vert,custo */
    int next;
  } adj[MAXn+MAXm];
  int cont; /* ultimo elemento de adj[] utilizado */

  /*inicializa um grafo de n vertices*/
  void init(int n) { /* O(n) */
    cont = n;
    for(int i=0 ; i<n ; i++){
      adj[i].v = i;
      adj[i].next = -1;
    }
  }

  /*adiciona vertice y aa lista de x, com custo c*/
  void insert(int x, int y, int c) { /* O(1) */
    int i = adj[x].v;

    i = adj[i].next = adj[x].v = cont++;

    adj[i].v = y;
    adj[i].c = c;
    adj[i].next = -1;
  }
```

```
  int next(int i) { /* O(1) */
    return adj[i].next;
  }
};


/**** Exemplo de uso ****/
grafo_t G;
int main() {
  int vert,cost;

  /* inicializando o grafo */
  G.init(3);

  /* inserindo elementos na lista de adjacencias */
  G.insert(0,2,5);
  G.insert(2,1,4);
  G.insert(0,1,3);

  /* percorrendo a lista de adjacencias */
  puts("> adjacentes de 0:");
  for(int i = G.next(0) ; i!=-1 ; i = G.next(i)) {
    vert = G.adj[i].v;
    cost = G.adj[i].c;

    printf("( %d-%d, custo %d )\n", 0, vert, cost);
  }

  return 0;
}
```

1.10 Emparalhamento Bipartido de Custo Maximo

```
Autor: Chines/Davi Costa
Complexidade: O(n^3)
Tempo de implementacao: ?
Testes: nuevo-3987
Dependencias: Sem dependencias
Descricao: Encontra o emparelhamento maximo
de custo maximo, para custo minimo insira as
arestas com peso negativo. Se uma aresta nao
existe o valor na matriz deve ser -1*INF.
CUidado: NAO UTILIZAR MEMSET PARA O -1*INF
```

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <vector>
#include <cstring>

using namespace std;

#define INF 0x3f3f3f3f
#define MAXN 351

int n, m; //# de vertices em cada lado
int adj[MAXN][MAXN]; /*Matriz de Adj*/
int labelx[MAXN], usedx[MAXN], lnk[MAXN];
int labely[MAXN], usedy[MAXN];
int mat; //Tamanho to match


//Auxiliar Caminho Aumentante
bool path(int i) {
  usedx[i] = 1;
  for (int j = 0; j < m; j++) {
    if (!usedy[j] && adj[i][j] != -INF &&
!abs(adj[i][j] - labelx[i] - labely[j])) {
      usedy[j] = 1;
      if (lnk[j] == -1 || path(lnk[j])) {
lnk[j] = i;
return true;
      }
    }
  }
  return false;
}

//Apos preencher adj chamar match()
int match() {
  mat = 0;
  memset(lnk,-1,sizeof(lnk));
  memset(labely,0,sizeof(labely));
  for (int i = 0; i < n; i++) {
    labelx[i] = 0;
    for (int j = 0; j < m; j++)
      if (adj[i][j] > labelx[i]) labelx[i] = adj[i][j];
  }
```

```
  for (int k = 0; k < n; k++) {
    while (1) {
      memset(usedx,0,sizeof(usedx));
      memset(usedy,0,sizeof(usedy));
      if (path(k)) { mat++; break; }
      int delta = INF;
      for (int i = 0; i < n; i++)
if (usedx[i])
  for (int j = 0; j < m; j++)
    if (!usedy[j] && adj[i][j] != -INF)
      delta = min(delta,labelx[i] + labely[j] - adj[i][j]);
      if (delta == 0 || delta == INF) break;
      for (int i = 0; i < n; i++)
if (usedx[i]) labelx[i] -= delta;
      for (int j = 0; j < m; j++)
if (usedy[j]) labely[j] += delta;
    }
  }
  int sum = 0;
  for (int i = 0; i < n; i++) sum += labelx[i];
  for (int j = 0; j < m; j++) sum += labely[j];
  return sum;
}

/*Exemplo de uso com custo minimo*/
int main() {
  int k, e;
  int from, to, cost;
  scanf("%d",&k);
  for (int z = 0; z < k; z++) {
    if (z != 0) printf("\n");
    scanf("%d %d",&n,&m);
    for (int i = 0; i < n; i++)
      for (int j = 0; j < m; j++)
adj[i][j] = -INF;
    scanf("%d",&e);
    for (int i = 0; i < e; i++) {
      scanf("%d %d %d",&from,&to,&cost);
      adj[from][to] = -cost;
    }
    int r = -match();
    printf("%d\n",r);
  }
  return 0;
}
```

1.11 LCA (Menor Ancestral Comum)

```
Autor: Igor Assis
Complexidade: O(n) + O(1) por query
Tempo de implementacao: ?
Testes: nuevo.2045
Dependencias: Range Minimum Query
Descricao: Dada uma arvore preprocessa de forma a
realizar querys da forma LCA(u,v) que retornam
o menor ancestral (mais longe da raiz) comum
de u e v na arvore.

#include <cstring>
#include <vector>

using namespace std;

#define MAXN 1024

int n, nE;
int adj[MAXN][MAXN], nadj[MAXN];

vector<int> L;
int E[2*MAXN], R[MAXN], vis[MAXN];

void euler(int u, int p, int el) {
  E[nE++] = u; L.push_back(el);
  vis[u] = 1;
  for (int i = 0; i < nadj[u]; i++)
    if (!vis[adj[u][i]] && adj[u][i] != p) {
      euler(adj[u][i], u, el+1);
      E[nE++] = u; L.push_back(el);
    }
}

void preprocess(int root) {
  int i;
  nE = 0;
  L.clear();
  memset(vis, 0, sizeof(vis));
  euler(root, -1, 0);
  for (i = 2*n-2; i >= 0; i--) R[E[i]] = i;
```

```
  init(L);
}

int lca(int u, int v) {
  return E[query(min(R[u],R[v]), max(R[u], R[v]))];
}

#include <cstdio>

/**** Exemplo simples de uso ****/
int main(){
  int i, u, v;
  scanf("%d", &n);
  memset(nadj, 0, sizeof(nadj));
  for (i = 0; i < n-1; i++) {
    scanf("%d%d", &u, &v);
    adj[u][nadj[u]++] = v;
    adj[v][nadj[v]++] = u;
  }

  preprocess(0);
  printf("%d\n", lca(2, 3));

  return 0;
}
```

1.12 Intersecção de Matróides

```
Autor: Igor Assis
Complexidade: O(|I|*B(n,m)), |I| = tamanho interseccao
B(n,m) = complexidade da busca
Tempo de implementacao: ?
Testes: spojbr.HONESTID (|I| = n, B(n,m) = O(mnlogˆ*n))
Dependencias: Nenhuma
Descricao: Encontra a interseccao de dois matroides.
Esta implementado o caso especifico de
matroide floresta e matroide cor de aresta unica.

#include <cstdio>
#include <cstring>
#include <queue>
#include <algorithm>

using namespace std;

#define MAXN 128
#define MAXM 128*128

#define MAXK 2*MAXN

int X2[MAXM], X1[MAXM], inX2[MAXM], inX1[MAXM];
int Y[MAXM], X_Y[MAXM], inY[MAXM], inX_Y[MAXM];

int n, m, ncor;
int nX1, nX2, nY, nX_Y;

int p[MAXM], mark[MAXM], comp[MAXN], rank[MAXN];

int orig[MAXM], dest[MAXM], empresa[MAXM], cor[MAXK];

int find(int u) {
  if (u == comp[u])
    return u;
  return comp[u] = find(comp[u]);
}

void unite(int u, int v)  {
  if (rank[u] > rank[v])
    comp[v] = u;
  else {
    comp[u] = v;
    if (rank[u] == rank[v])
      rank[v]++;
  }
}

int caminho() {
  int i;
  queue<int>Q;

  memset(mark, 0, sizeof(mark));
  for (i = 0; i < nX1; i++) {
    p[X1[i]] = -1;
    mark[X1[i]] = 1;
    if (inX2[X1[i]] != -1)
      return X1[i];
    Q.push(X1[i]);
  }
```

```
  while (!Q.empty()) {
    int u = Q.front(); Q.pop();
    if (inX2[u] != -1)
      return u;
    if (inY[u] == -1) {
      /* monta ciclo em uma componente */
      if (find(orig[u]) == find(dest[u])) {

for (i = 0; i < nY; i++)
  if (!mark[Y[i]] && find(orig[Y[i]]) == find(orig[Y[u]])) {
    p[Y[i]] = u;
    mark[Y[i]] = 1;
    if (inX2[Y[i]] != -1)
      return Y[i];
    Q.push(Y[i]);
  }
      } else {
for (i = 0; i < nY; i++)
  if (!mark[Y[i]]) {
    p[Y[i]] = u;
    mark[Y[i]] = 1;
    if (inX2[Y[i]] != -1)
      return Y[i];
    Q.push(Y[i]);
  }
      }
    } else {
      for (i = 0; i < nX_Y; i++)
if (!mark[X_Y[i]] &&
    (empresa[u] == empresa[X_Y[i]] || !cor[empresa[u]])) {
  p[X_Y[i]] = u;
  mark[X_Y[i]] = 1;
  if (inX2[X_Y[i]] != -1)
    return X_Y[i];
  Q.push(X_Y[i]);
}
    }
  }

  return -1;
}

int matroide() {
  int i, u, res;
  nX2 = nX1 = m; nY = 0;
  for (i = 0; i < m; i++) {
    inX2[i] = X2[i] = inX1[i] = X1[i] = i;
    inY[i] = -1;
  }
  for (i = 0; i < n; i++) {comp[i] = i; rank[i] = 1;}
  memset(cor, 0, sizeof(cor));

  res = 0;
  while ((u = caminho()) != -1) {
    while (u != -1) {
      /* ou-exclusivo */
      if (inY[u] == -1) {
Y[nY] = u;
inY[u] = nY++;
X_Y[inX_Y[u]] = X_Y[--nX_Y];
inX_Y[X_Y[nX_Y]] = inX_Y[u];
cor[empresa[u]] = 1; /* marca empresa */
      } else {
X_Y[nX_Y] = u;
inX_Y[u] = nX_Y++;
Y[inY[u]] = Y[--nY];
inY[Y[nY]] = inY[u];
cor[empresa[u]] = 0; /* desmarca empresa */
      }
      u = p[u];
    }
    /* atualiza componentes */
    for (i = 0; i < n; i++) {comp[i] = i; rank[i] = 1;}
    for (i = 0; i < nY; i++)
      unite(find(orig[Y[i]]), find(dest[Y[i]]));
    /* atualiza X2 e X1 */
    nX2 = nX1 = 0;
    memset(inX2, -1, sizeof(inX2));
    memset(inX1, -1, sizeof(inX1));
    for (i = 0; i < m; i++) {
      if (inY[i] == -1 && find(orig[i]) != find(dest[i])) {
X2[nX2] = i;
inX2[i] = nX2++;
      }
      if (inY[i] == -1 && !cor[empresa[i]]) {
X1[nX1] = i;
inX1[i] = nX1++;
```

```
      }
    }
    res++;
  }
  return res;
}

/* Exemplo e' o problema HONESTID do spojbr */
int main() {
  int i, cases = 1;

  while (scanf("%d%d%d", &n, &m, &ncor) == 3) {
    for (i = 0; i < m; i++) {
      scanf("%d%d%d", &orig[i], &dest[i], &empresa[i]);
      orig[i]--; dest[i]--;
    }
    printf("Instancia %d\n", cases++);
    if (matroide() == n-1)
      printf("sim\n\n");
    else printf("nao\n\n");
  }

  return 0;
}
```

1.13 Fluxo Máximo

```
Autor: Felipe Sodré, Igor Assis
Complexidade: O(n^3)
Tempo de implementacao: ?
Testes: uva.820 uva.10330 uva.10480
Dependencias: Nenhuma
Descricao: Algoritmo para encontrar o fluxo máximo de s a t.
#include <cstdio>

#include <cstring>
#include <queue>

using namespace std;

#define MAXN 100

int adj[MAXN][MAXN], nadj[MAXN], n, m;
int x[MAXN][MAXN], cap[MAXN][MAXN], r[MAXN][MAXN];
int e[MAXN], d[MAXN], s, t;

queue<int> Q;

#define adm(u, v) (d[u] == d[v] + 1)

void push(int u, int v, int c) {
  x[u][v] += c;  x[v][u] -= c;
  r[u][v] -= c;  r[v][u] += c;
  e[u] -= c;     e[v] += c;
}

void preprocess() {
  memset(x, 0, sizeof(x));
  memset(e, 0, sizeof(e));
  memset(d, 0, sizeof(d));
  for (int i = 0; i < nadj[s]; i++) {
    int v = adj[s][i];
    push(s, v, cap[s][v]);
    if (v != s && v != t) Q.push(v);
  }
  d[s] = n;
}

void push_relabel(int u) {
  int j = -1;
  for (int i = 0; i < nadj[u]; i++) {
    int v = adj[u][i];
    if (e[u] <= 0) break;
    if (adm(u, v) && r[u][v] > 0) {
      int delta = min(e[u], r[u][v]);
      push(u, v, delta);
      if (e[v] > 0 && v != s && v != t) Q.push(v);
    }
    if (r[u][v] > 0 && (j == -1 || d[v] < d[j])) j = v;
  }
  if (e[u] > 0) {
    d[u] = d[j] + 1;
    Q.push(u);
  }
}

int maxflow() {
```

```
  int flow = 0;
  memcpy(r, cap, sizeof(r));
  preprocess();
  while (!Q.empty()) {
    int u = Q.front(); Q.pop();
    push_relabel(u);
  }
  for (int i = 0; i < nadj[s]; i++)
    flow += x[s][adj[s][i]];
  return flow;
}

/* funcoes para encontar um s-t-corte minimo */
#define MAXM MAXN*MAXN

int mark[MAXN], cut[MAXM];

void dfs(int u) {
  mark[u] = 1;
  for (int i = 0; i < nadj[u]; i++)
    if (!mark[adj[u][i]] && r[u][adj[u][i]] > 0)
      dfs(adj[u][i]);
}

void mincut() {
  memset(mark, 0, sizeof(mark));
  dfs(s);
  for (int i = 0; i < n; i++)
    if (mark[i])
      for (int j = 0; j < nadj[i]; j++)
if (!mark[adj[i][j]]) printf("%d %d\n", i, adj[i][j]);
}

/**** Exemplo simples de uso ****/
int main(void){
  return 0;
}
```

1.14 Fluxo Máximo Edmonds-Karp

```
Autor: Igor Assis
Complexidade: O(m*F) F = fluxo maximo
Tempo de implementacao: ?
Testes: uva.820 uva.10330 uva.10480
Descricao: Encontra o fluxo maximo de s-t utilizando
o algoritmo de caminhos aumentantes minimos.
```

```
#include <cstdio>
#include <cstring>
#include <queue>

using namespace std;

#define MAXN 128
#define INF 0x3f3f3f3f

int adj[MAXN][MAXN], nadj[MAXN], n, m;

int x[MAXN][MAXN], cap[MAXN][MAXN], r[MAXN][MAXN];
int vis[MAXN], pred[MAXN], delta[MAXN];
int s, t;

int aumenta() {
  int u;
  queue<int> Q;
  memset(vis, 0, sizeof(vis));
  vis[s] = 1; pred[s] = -1; delta[s] = INF;
  Q.push(s);
  while (!Q.empty()) {
    u = Q.front(); Q.pop();
    if (u == t) break;
    for (int i = 0; i < nadj[u]; i++) {
      int v = adj[u][i];
      if (!vis[v] && r[u][v] > 0) {
vis[v] = 1; pred[v] = u;
delta[v] = min(delta[u], r[u][v]);
Q.push(v);
      }
    }
  }
  if (u != t) return 0;
  while (pred[u] != -1) {
    x[pred[u]][u] += delta[t]; r[pred[u]][u] -= delta[t];
    x[u][pred[u]] -= delta[t]; r[u][pred[u]] += delta[t];
    u = pred[u];
  }
  return delta[t];
}
```

```
int maxflow() {
  int inc, res = 0;
  /* constroi lista de adjacencias */
  memset(nadj, 0, sizeof(nadj));
  memset(x, 0, sizeof(x));
  memcpy(r, cap, sizeof(r));
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      if (cap[i][j] || cap[j][i]) adj[i][nadj[i]++] = j;
  while ((inc = aumenta()) > 0)
    res += inc;
  return res;
}

/**** Exemplo simples de uso ****/
int main() {
  int i, j;
  int u, v, c;

  for (;;) {
    scanf("%d", &n);
    if (!n) break;
    memset(cap, 0, sizeof(cap));
    scanf("%d%d%d", &s, &t, &m); s--; t--;
    for (i = 0; i < m; i++) {
      scanf("%d%d%d", &u, &v, &c); u--; v--;
      cap[u][v] += c;
      cap[v][u] += c;
    }
    printf("Fluxo maximo %d.\n\n", maxflow());
    printf("Fluxo:\n");
    for (i = 0; i < n; i++)
      for (j = 0; j < nadj[i]; j++)
if (x[i][adj[i][j]] > 0)
  printf("(%d %d) %d\n", i, adj[i][j], x[i][adj[i][j]]);
    printf("Arestas do corte:\n");
    mincut(); /* ver mincut() no algoritmo de preflow */
    printf("\n");
  }

  return 0;
}
```

1.15 Árvore Geradora de Diâmetro Mínimo (mais eficiente)

```
Autor: Igor Assis
Complexidade: O(n^3 + n^2logn)
Tempo de implementacao: ?
Testes: spoj.MDST
Dependencias: Nenhuma
Descricao: Encontra o diametro da arvore geradora de
diametro minimo, mais rapido se só precisa
do diametro.

#include <cstdio>
#include <cstring>
#include <utility>
#include <algorithm>
#include <queue>
#include <vector>

using namespace std;

#define MAXN 1123
#define INF 0x3f3f3f3f

int n, m;
int adj[MAXN][MAXN], peso[MAXN][MAXN], nadj[MAXN];
int d[MAXN][MAXN], p[MAXN], t[MAXN];
int mark[MAXN];

int mdst() {
  int i, j, k, u;

  if (n == 1)
    return 0;

  // all-pairs-shortest-path
  for (k = 0; k < n; k++) {
    queue<int> Q;
    memset(mark, 0, sizeof(mark));
    d[k][k] = 0;
    mark[k] = 1;
    Q.push(k);
    while (!Q.empty()) {
      u = Q.front(); Q.pop();
```

```
      for (i = 0; i < nadj[u]; i++)
if (!mark[adj[u][i]]) {
  Q.push(adj[u][i]);
  d[k][adj[u][i]] = d[k][u] + 1;
  mark[adj[u][i]] = 1;
}
    }
  }

  // absolute local 1-center
  int H = INF;
  i = 0;
  memset(t, 0, sizeof(t));
  for (u = 0; u < n; u++)
    for (j = 0; j < nadj[u]; j++)
      if (u < adj[u][j]) {
for (k = 0; k < n; k++)
  t[i] = max(t[i], min(d[u][k], d[adj[u][j]][k]));
H = min(H, peso[u][j] + 2*t[i++]);
      }

  int value = INF;
  i = 0;
  for (u = 0; u < n; u++)
    for (j = 0; j < nadj[u]; j++)
      if (u < adj[u][j] && 2*t[i++] <= H) {
vector<pair<int, int> > L;
for (int k = 0; k < n; k++)
  L.push_back(make_pair(d[u][k], d[adj[u][j]][k]));
sort(L.begin(), L.end(), greater< pair<int, int> > ());
int p = 0;
value = min(value, 2*L[0].first);
for (int k = 0; k < n; k++)
  if (L[p].second < L[k].second)
    value = min(value,
peso[u][j]+L[p].second+L[k].first), p = k;
value = min(value, 2*L[p].second);
      }

  return value;
}

/**** Exemplo simples de uso ****/
int main(){
  return 0;
}
```

1.16 Árvore Geradora de Diâmetro Mínimo

```
Autor: Igor Assis
Complexidade: O(n^2 + mn^2)
Tempo de implementacao: ?
Testes: spoj.MDST
Dependencias: Nenhuma
Descricao: Encontra o diametro da arvore geradora de
diametro minimo e um absolute 1-center a partir do qual
da para se obter a arvore.

#include <cstdio>
#include <cstring>
#include <utility>
#include <algorithm>
#include <queue>
#include <vector>

using namespace std;

#define MAXN 1012
#define INF 0x3f3f3f3f

int n, m;
int adj[MAXN][MAXN], peso[MAXN][MAXN], nadj[MAXN];

int d[MAXN][MAXN], pred[MAXN][MAXN], t[2][MAXN], k[MAXN];
int U, L;

int mark[MAXN];

/* a arvore geradora de diametro minimo é
   a arvore de caminhos minimos a partir do vértices CAC: que é
   um vértice criado a distancia d/2 de _j_ na aresta (i,j) */
struct _CAC {int i, j, d;} CAC;

/* trocar para floyd-warshall se tem pesos nas arestas */
int asp() {
  // all-pairs-shortest-path
  int vc = -1;
  memset(d, 0, sizeof(d));
```

```
  memset(k, -1, sizeof(k));
  for (int j = 0; j < n; j++) {
    queue<int> Q;
    memset(mark, 0, sizeof(mark));
    mark[j] = 1;
    pred[j][j] = -1;
    Q.push(j);
    while (!Q.empty()) {
      int u = Q.front(); Q.pop();
      if (k[u] == -1 || d[j][u] > d[j][k[j]]) k[j] = u;
      for (int i = 0; i < nadj[u]; i++)
if (!mark[adj[u][i]]) {
  Q.push(adj[u][i]);
  pred[j][adj[u][i]] = u;
  d[j][adj[u][i]] = d[j][u] + 1;
  mark[adj[u][i]] = 1;
}
    }
    if (vc == -1 || d[j][k[j]] < d[vc][k[vc]]) vc = j;
  }
  return vc;
}

int update(int i, int j, int k) {
  int c, delta = t[j][k];
  for (c = 0; c < n; c++) t[j][c] -= delta;
  for (c = 0; c < n; c++)
    if (t[j][c] > 0 && t[i][c] > 0) break;
  if (c == n) {
    U = L;
    CAC.i = i; CAC.j = j; CAC.d = abs(U-2*d[j][k]);
    return 1;
  }
  return 0;
}

int mdst() {
  int j, vc;

  vc = asp();
  U = 2*d[vc][k[vc]];
  CAC.i = -1; CAC.j = vc; CAC.d = 0;

  for (int r = 0; r < n; r++)
    for (int u = 0; u < nadj[r]; u++) if (r < adj[r][u]) {
int s = adj[r][u];
if (k[r] == k[s]) continue;
if ((L = peso[r][u] + d[r][k[s]] + d[s][k[r]]) >= U)
  continue;
memcpy(t[0], d[r], sizeof(t[0]));
memcpy(t[1], d[s], sizeof(t[1]));
if (update(1, 0, k[s]) || update(0, 1, k[r])) continue;
for (;;) {
  int maxv = -1, maxi, maxj;
  for (j = 0; j < n; j++)
    if ((t[0][j] > 0 && t[1][j] > 0)) {
      if (maxv == -1 || t[0][j] > maxv)
maxv = t[0][j], maxi = 0, maxj = j;
      if (maxv == -1 || t[1][j] > maxv)
maxv = t[1][j], maxi = 1, maxj = j;
      }
  L = L + t[1-maxi][maxj];
  if (L >= U) break;
  if(update(maxi, 1-maxi, maxj)) break;
}
      }
  return U;
}

/**** Exemplo simples de uso ****/
int main(){
}
```

1.17 Fluxo máximo de custo mínimo

```
Autor: Frank Chu, Igor Naverniouk, Igor Assis
Complexidade: O(n^2*flow  <?  n^3*fcost)
Tempo de implementacao: ?
Testes: uva.10594 uva.10806
Dependencias: Nenhuma
Descricao: Fluxo maximo de custo minimo entre dois vertices
s e t usando algoritmo de caminhos aumentantes
minimos.

#include <cstring>
#include <climits>
#include <algorithm>
```

```
using namespace std;

// the maximum number of vertices + 1
#define MAXN 1024

// adjacency matrix (fill this up)
int cap[MAXN][MAXN];

// cost per unit of flow matrix (fill this up)
int cost[MAXN][MAXN];

// flow network and adjacency list
int n, s, t;
int x[MAXN][MAXN], adj[MAXN][MAXN], nadj[MAXN];

// Dijkstra's successor and depth
int par[MAXN], d[MAXN]; // par[source] = source;

// Labelling function
int pi[MAXN];

#define INF (0x3f3f3f3f)

// Dijkstra's using non-negative edge weights (cost + potential)
#define Pot(u,v) (d[u] + pi[u] - pi[v])
bool dijkstra( int n, int s, int t ) {
  for( int i = 0; i < n; i++ ) d[i] = INF, par[i] = -1;
  d[s] = 0;
  par[s] = -n - 1;

  for (;;) {
    // find u with smallest d[u]
    int u = -1, bestD = INF;
    for(int i = 0; i < n; i++) if (par[i] < 0 && d[i] < bestD)
 bestD = d[u = i];
    if(bestD == INF) break;

    // relax edge (u,i) or (i,u) for all i;
    par[u] = -par[u] - 1;
    for (int i = 0; i < nadj[u]; i++)
      {
// try undoing edge v->u
int v = adj[u][i];
if (par[v] >= 0 ) continue;
if (x[v][u] && d[v] > Pot(u,v) - cost[v][u])
  d[v] = Pot( u, v ) - cost[v][u], par[v] = -u-1;

// try edge u->v
if( x[u][v] < cap[u][v] && d[v] > Pot(u,v) + cost[u][v] )
  d[v] = Pot(u,v) + cost[u][v], par[v] = -u - 1;
      }
  }

  for (int i = 0; i < n; i++ ) if (pi[i] < INF) pi[i] += d[i];

  return par[t] >= 0;
}
#undef Pot

int mfmc(int &fcost) {
  // build the adjacency list
  memset(nadj, 0, sizeof(nadj));
  for(int i = 0; i < n; i++)
    for(int j = 0; j < n; j++)
      if(cap[i][j] || cap[j][i]) adj[i][nadj[i]++] = j;

  memset(x, 0, sizeof(x));
  memset(pi, 0, sizeof(pi));
  int flow = fcost = 0;

  // repeatedly, find a cheapest path from s to t
  while(dijkstra(n, s, t)) {
    // get the bottleneck capacity
    int bot = INT_MAX;
    for(int v = t, u = par[v]; v != s; u = par[v = u])
      bot = min(bot, x[v][u] ? x[v][u] : (cap[u][v] - x[u][v]));

    // update the flow network
    for(int v = t, u = par[v]; v != s; u = par[v = u] )
      if(x[v][u]) { x[v][u] -= bot; fcost -= bot * cost[v][u]; }
      else { x[u][v] += bot; fcost += bot * cost[u][v]; }

    flow += bot;
  }

  return flow;
}
```

```
/**** Exemplo simples de uso ****/
#include <iostream>
#include <stdio.h>

using namespace std;

/*
 * PARAMETERS:
 *      - cap (global): adjacency matrix where
 *                cap[u][v] is the capacity of the edge u->v.
 *                cap[u][v] is 0 for non-existent edges.
 *      - cost (global): a matrix where cost[u][v] is the cost
 *                per unit of flow along the edge u->v.
 *                If cap[u][v] == 0, cost[u][v] is
 *                ignored. ALL COSTS MUST BE NON-NEGATIVE!
 *      - n: the number of vertices
 *      - s: source vertex.
 *      - t: sink.
 * RETURNS:
 *      - the flow
 *      - the total cost through 'fcost'
 *      - fnet contains the flow network. Careful:
 *           both fnet[u][v] and fnet[v][u] could be positive.
 *           Take the difference.
 */

int main() {
  int numV;
  //  while ( cin >> numV && numV ) {
  cin >> numV;
  memset( cap, 0, sizeof( cap ) );

  int m, a, b, c, cp;
  int s, t;
  cin >> m;
  cin >> s >> t;

  // fill up cap with existing capacities.
  // if the edge u->v has capacity 6, set cap[u][v] = 6.
  // for each cap[u][v] > 0, set cost[u][v] to  the
  // cost per unit of flow along the edge u->v
  for (int i=0; i<m; i++) {
    cin >> a >> b >> cp >> c;
    cost[a][b] = c; // cost[b][a] = c;
    cap[a][b] = cp; // cap[b][a] = cp;
  }

  int fcost;
  int flow = mcmf3( numV, s, t, fcost );
  cout << "flow: " << flow << endl;
  cout << "cost: " << fcost << endl;

  return 0;
}
```

1.18 Componentes Fortemente Conexas

```
Autor: Igor Assis
Complexidade: O(n+m)
Tempo de implementacao: ?
Testes: uva.247 uva.10510 SPOJ.CARPDAPIO
Dependencias: Nenhuma
Descricao: Encontra as componentes fortemente conexas de um grafo
orientado. Componentes nomeadas de 1 à ncomp.

#include <cstring>
#include <algorithm>

using namespace std;

#define MAXN 1024

int adj[MAXN][MAXN], nadj[MAXN];
int comp[MAXN], vis[MAXN], stck[MAXN], t, high[MAXN], num, ncomp;

void dfscc(int u) {
  int i, v;
  high[u] = vis[u] = num--;
  stck[t++] = u;
  for (i = 0; i < nadj[u]; i++) {
    v = adj[u][i];
    if (!vis[v]) {
      dfscc(v);
      high[u] = max(high[u], high[v]);
    } else if (vis[v] > vis[u] && !comp[v])
      high[u] = max(high[u], vis[v]);
  }
  if (high[u] == vis[u]) {
```

```
    ncomp++;
    do {
      v = stck[--t];
      comp[v] = ncomp;
    } while (v != u);
  }
}

void scc(int n) {
  int i;
  ncomp = t = 0; num = n;
  memset(vis, 0, sizeof(vis));
  memset(comp, 0, sizeof(comp));
  for (i = 0; i < n; i++)
    if (!vis[i])
      dfscc(i);
}

/**** Exemplo simples de uso ****/
#include <cstdio>

int main(void){
  int i, u, v;

  scanf("%d%d", &n, &m);
  memset(nadj,0,sizeof(nadj));
  for (i = 0; i < m; i++) {
    scanf("%d%d", &u, &v);
    adj[u][nadj[u]++] = v;
  }

  scc();

  printf("Numero componentes: %d\n", ncomp);
  for (i = 0; i < n; i++)
    printf("componente[%d] = %d\n", i, comp[i]);

  return 0;
}
```

1.19 Caminho Mínimo em um DAG

```
Autor: Alexandre Kunieda
Complexidade: O(E + V)
Tempo de implementacao: ?
Testes: uva-10350
Dependencias: Nenhuma
Descricao: Caminho minimo em DAG

int pai[MAXN]; /* se precisar reconstruir o caminho */
int adj[MAXN][MAXN]; /* lista de adj */
int custo[MAXN][MAXN]; /* refere-se a aresta de adj */
int nadj[MAXN]; /* grau de cada vertice */

int D[MAXN]; /* distancia de cada vertice até b */
int foi[MAXN];

void minDFS(int k) {
  int i,j,c;
  foi[k]=1;
  for(j=0 ; j<nadj[k] ; j++) {
    i = adj[k][j];
    c = custo[k][j];

    if(!foi[i]) minDFS(i);
    if(D[k] > D[i]+c) {
      D[k] = D[i]+c;
      pai[k] = i;
    }
  }
}

/* D é preenchido ao contrário: D[u] é a distância
de u até o vértice final b */
int minDAG(int a, int b, int n) {
  memset(D, 0x3f, n*sizeof(int));
  memset(foi, 0, n*sizeof(int));
  D[b] = 0;
  minDFS(a);
  return D[a];
}

/* Para obter o caminho mínimo de um vértice a até
todos os outros, gerar as arestas invertidas, e chamar
esta função */
/*
void minDAG2(int a, int n) {
  int i;
```

```
  memset(D, 0x3f, n*sizeof(int));
  memset(foi, 0, n*sizeof(int));
  D[a] = 0;
  for(i=0 ; i<n ; i++)
    if(!foi[i]) minDFS(i);
}
*/

/*Exemplo de uso*/
int main() {
  int n, m;
  int origem,destino;
  int from, to, cost;
  int i;
  while (scanf("%d %d",&n,&m) == 2 && n != 0) {
    scanf("%d %d",&origem,&destino);
    memset(nadj,0,sizeof(nadj));
    for (i = 0; i < m; i++) {
      scanf("%d %d %d",&from,&to,&cost);
      custo[from][nadj[from]] = cost;
      adj[from][nadj[from]++] = to;
    }
    printf("%d\n",minDAG(origem,destino,n));
  }
  return 0;
}
```

1.20 Caminho Mínimo em um DAG

```
Autor: NBMundial / Davi Costa
Complexidade: O(E + V)
Tempo de implementacao: ?
Testes: uva-10350
Dependencias: Topological Sort
Descricao: Caminho minimo em DAG


#define MAXN 100
#define INF 0x3f3f3f

int pai[MAXN]; //se precisar reconstruir o caminho
int adj[MAXN][MAXN]; //lista de adj.
int custo[MAXN][MAXN]; //refere-se a aresta de adj
int nadj[MAXN]; //grau de cada vertice
int d[MAXN]; //distancia de s ateh cada vertice
int tops[MAXN]; //topological sort
int path[MAXN]; //caminho ate t
bool used[MAXN];
int ip;

/*
n: numero de vertices, s: origem
*/
int calc_path(int n, int s) {
  topsort(n);
  memset(pai,-1,sizeof(pai));
  for (int i = 0; i < n; i++) d[i] = INF;
  d[s] = 0;
  pai[s] = 0;
  for (int i = 0; i < n; i++) {
    int x = tops[i];
    if (pai[x] == -1) continue;
    for (int j = 0; j < nadj[x]; j++) {
      int v = adj[x][j];
      int cost = custo[x][j];
      if (d[v] > d[x] + cost) {
d[v] = d[x] + cost;
pai[v] = x;
      }
    }
  }
}

/*Exemplo de uso*/

int main() {
  int n, m;
  int origem,destino;
  int from, to, cost;
  while (scanf("%d %d",&n,&m) == 2 && n != 0) {
    scanf("%d %d",&origem,&destino);
    memset(nadj,0,sizeof(nadj));
    for (int i = 0; i < m; i++) {
      scanf("%d %d %d",&from,&to,&cost);
      custo[from][nadj[from]] = custo[to][nadj[to]] = cost;
      adj[from][nadj[from]++] = to;
      adj[to][nadj[to]++] = from;
    }
```

```
    shortdag(n,origem);
    printf("%d\n",d[destino]);
  }
  return 0;
}
```

1.21 Stable Marriage

```
Autor: Igor Naverniouk, Igor Assis
Complexidade: O(m^2), m = numero de homens
Tempo de implementacao: ?
Testes: uva.11119
Dependencias: Nenhuma
Descricao:
Takes a set of m men and n women, where each person has
an integer preference for each of the persons of opposite
sex. Produces a matching of each man to some woman.
The matching will have the following properties:
- Each man is assigned a different woman
(n must be at least m).
- No two couples M1W1 and M2W2 will be unstable.
- The solution is man-optimal.
Two couples are unstable if
- M1 prefers W2 over W1 and
- W1 prefers M2 over M1.

#include <cstring>

#define MAXM 1024
#define MAXN 1024

int m, n;
int L[MAXM][MAXN], R[MAXN][MAXM];
int L2R[MAXM], R2L[MAXN];

int p[MAXM];

void stableMarriage() {
  memset( R2L, -1, sizeof( R2L ) );
  memset( p, 0, sizeof( p ) );

  // Each man proposes...
  for( int i = 0; i < m; i++ ) {
    int man = i;
    while( man >= 0 ) {
      // to the next woman on his list in order
      // of decreasing preference, until one of them accepts;
      int wom;
      while( 1 ) {
wom = L[man][p[man]++];
if(R2L[wom] < 0 || R[wom][man] > R[wom][R2L[wom]]) break;
      }

      // Remember the old husband of wom.
      int hubby = R2L[wom];

      // Marry man and wom.
      R2L[L2R[man] = wom] = man;

      // If a guy was dumped in the process, remarry him now.
      man = hubby;
    }
  }
}

/**** Exemplo simples de uso ****/
int main() {
  /* INPUTS:
   *      - m:       number of men.
   *      - n:       number of women
   *                 (must be at least as large as m).
   *      - L[i][]:  the list of women in order of
   *                 decreasing preference of man i.
   *      - R[j][i]: the attractiveness level of man i to woman j.
   * OUTPUTS:
   *      - L2R[]:   the mate of man i (always between 0 and n-1)
   *      - R2L[]:   the mate of woman j (or -1 if single)
   */
}
```

1.22 Corte mínimo geral (Stoer-Wagner)

```
Autor: Igor Naverniouk, Igor Assis
Complexidade: O(n^3)
Tempo de implementacao: ?
Testes: uva.10989 spojbr.EINSTEIN
Dependencias: Nenhuma
Descricao: Algoritmo que encontra o valor do corte mínimo
```

```
dentre todos de um grafo nao-orientado com peso na aresta.

#include <algorithm>

// Maximum number of vertices in the graph
#define MAXN 256

// Maximum edge weight (MAXW * NN * NN must fit into an int)
#define MAXW 1000

// Adjacency matrix and some internal arrays
int adj[MAXN][MAXN], v[MAXN], w[MAXN], na[MAXN];
bool a[MAXN];

int mincut(int n) {
  // init the remaining vertex set
  for(int i = 0; i < n; i++) v[i] = i;

  // run Stoer-Wagner
  int best = MAXW * n * n;
  while(n > 1) {
    // initialize the set A and vertex weights
    a[v[0]] = true;
    for( int i = 1; i < n; i++ ) {
      a[v[i]] = false;
      na[i - 1] = i;
      w[i] = adj[v[0]][v[i]];
    }

    // add the other vertices
    int prev = v[0];
    for(int i = 1; i < n; i++) {
      // find the most tightly connected non-A vertex
      int zj = -1;
      for(int j = 1; j < n; j++)
if(!a[v[j]] && (zj < 0 || w[j] > w[zj]))
  zj = j;

      // add it to A
      a[v[zj]] = true;

      // last vertex?
      if(i == n - 1) {
// remember the cut weight
best = min(best, w[zj]);

// merge prev and v[zj]
for(int j = 0; j < n; j++)
  adj[v[j]][prev] = adj[prev][v[j]] += adj[v[zj]][v[j]];
v[zj] = v[--n];
break;
      }
      prev = v[zj];

      // update the weights of its neighbours
      for(int j = 1; j < n; j++)
if(!a[v[j]])
  w[j] += adj[v[zj]][v[j]];
    }
  }
  return best;
}

/**** Exemplo simples de uso ****/
int main() {
  // preencha a matriz de adjacencias adj[][]
  // se nao existe a aresta (u,v) coloque *zero*
  int n, answer = mincut( n );
  return 0;
}
```

1.23 Topological Sort

```
Autor: Alexandre Kunieda
Complexidade: O(E + V)
Tempo de implementacao: ?
Testes: uva-10350 auxiliar para shortdag, spojbr-ORKUT
Dependencias: Nenhuma
Descricao: Ordena Topologicamente, ou verifica que não há
ordenação topológica. Para verificação de existência da
ordenação, implemente os trechos comentados do código;
além disso, as duas funções podem ser declaradas como void.

int n;
int adj[MAXN][MAXN]; /* lista de adj */
int nadj[MAXN]; /* grau de cada vertice */

int foi[MAXN], ip; /* auxiliar */
```

```
/* int foi2[MAXN]; */
int tops[MAXN]; /* resposta */

int DFS(int k) {
  int i,j;

  foi[k] = /* foi2[k] = */ 1;
  for(j=0 ; j<nadj[k] ; j++) {
    i = adj[k][j];
    /* if(foi2[i]) return 0; */
    if(!foi[i] && !DFS(i)) return 0;
  }

  tops[--ip] = k;
  /* foi2[k] = 0; */

  return 1;
}

/*
popular n: numero de vertices
apos chamar ord_top() "tops" tera a solucao
*/
int ord_top() {
  int i;

  memset(foi, 0, n*sizeof(int));
  /* memset(foi2, 0, n*sizeof(int)); */
  ip = n;

  for(i=0 ; i<n ; i++)
    if(!foi[i] && !DFS(i)) return 0;

  return 1;
}


/** Exemplo de uso **/

int main() {
  int m,i, from,to;
  while (scanf("%d %d",&n,&m) == 2 && n != 0) {
    memset(nadj,0,sizeof(nadj));
    for (i = 0; i < m; i++) {
      scanf("%d %d",&from,&to);
      adj[from][nadj[from]++] = to;
    }
    if(!ord_top()) puts("não há ordenação topológica");
    else for (i = 0; i < n; i++) printf("%d ",tops[i]);
  }
  return 0;
}
```

1.24 Problema do caixeiro viajante

```
Complexidade: O(n^2 * 2^n)

Descrição: encontra o circuito hamiltoniano
de custo mínimo.

#include <iostream>
#include <vector>
#include <algorithm>
#include <cstdlib>

using namespace std;

#define REP(i,n) for(int i=0; i<n; ++i)

const int M = 20;
const int INF = 1<<29;
int best[1<<M][M];
int prev[1<<M][M];

void buildPath(int S, int i, vector<int> &path) {
  if (!S) return;
  buildPath(S^(1<<i), prev[S][i], path);
  cout << i << " ";
  path.push_back(i);
}

int shortestHamiltonPath(int w[M][M],int n,int s,
 vector<int> &path) {
  int N = 1 << n;
  REP(S,N) REP(i,n) best[S][i] = INF;
  best[1<<s][s] = 0;
  REP(S,N) REP(i,n) if (S&(1<<i)) REP(j,n)
```

```
    if (best[S|(1<<j)][j] > best[S][i] + w[i][j])
      best[S|(1<<j)][j] = best[S][i] + w[i][j],
      prev[S|(1<<j)][j] = i;
  int t = min_element(best[N-1], best[N-1]+n) - best[N-1];
  path.clear(); buildPath(N-1, t, path);
  return best[N-1][t];
}

int main()
{
  return 0;
}
```

1.25 Two Sat

```
Autor: Davi Costa
Complexidade: O(E + V)
Tempo de implementacao: ?
Testes: spoj.CARDAPIO nuevo-2886
Dependencias: SCC
Descricao:
Determina se existe uma atribuicao
que satisfaca a expressao (Xi v Xk)^(Xj v !Xl)...
Para cada clausula deve haver uma aresta
no grafo da forma !Xi -> Xk e !Xk -> Xi
A funcao "clau" gera as arestas automaticamente
dada a clausula, ver exemplo.
No grafo a literal P esta na posicao 2*p e
sua negacao em 2*p + 1. (checklist: alocando o suficiente?)

#define N(x) (2*x + 1)
#define Y(x) (2*x)
#define NEG(x) (x%2 == 1 ? x-1 : x+1)

/*n deve ser o numero total de literais p (nao 2*p))*/
bool two_sat(int n) {
  n *= 2;
  bool ok = true;
  int i;
  scc(n);
  for (i=0; i<n/2 && ok; i++)
    ok &= (comp[2*i] != comp[2*i+1]);
  return ok;
}

/*Insira a clausula como descrito na main*/
void clau(int x, int y) {
  int negx = NEG(x), negy = NEG(y);
  adj[negx][nadj[negx]++] = y;
  adj[negy][nadj[negy]++] = x;
}

int main() {
  /* A ou B */
  clau(Y(A),Y(B));
  /* A ou NAO(B) */
  clau(Y(A),N(B));
  /* NAO(A) ou NAO(B) */
  clau(N(A),N(B));
  /* NAO(NAO(A)) ou B */
  clau(Y(A),Y(B));
  /*ou*/
  clau(NEG(N(A)),Y(B));
  /* Nunca use N e Y recursivamente, se precisar
     use NEG apos a primeira negacao ou afirmacao */
  /* Exemplo de uso, cada pessoa entra com x, y, w, z,
     sendo que x ou y devem ser atendidos e w ou z nao podem */
  int k, n;
  memset(nadj,0,sizeof(nadj));
  scanf("%d %d",&k,&n);
  for (int i = 0; i < k; i++) {
    int x,y,w,z;
    scanf("%d %d %d %d",&x,&y,&w,&z);
    clau(Y(x),Y(y));
    clau(N(w),N(z));
  }
  if (two_sat(n)) printf("yes\n");
  else printf("no\n");
}
```

2 Programação Dinâmica

2.1 Hash polinomial

```
Autor: André Linhares
Complexidade: O(nm)
```

```
Teste: UVA 11019

Descrição: aplica uma função de hash em todas
as submatrizes de dimensões determinadas.

#include <cstdio>
#include <iostream>
#include <algo.h>
#include <algorithm>
#include <vector>

using namespace std;

#define for_to(i,j,k) for(i=j; i<=k; ++i)
#define all(v) v.begin(),v.end()

#define MAX 1010
#define ui unsigned int

template <class Q>
void hash(Q T[MAX][MAX],ui h[MAX][MAX],int n,int m,int a,int b)
{
  ui p,pot;
  int i,j;
  static ui g[MAX][MAX];

  p=0x9e6fe013;
  pot=power(p,b-1);
  for_to(i,0,n-1)
  {
    g[i][0]=T[i][0];
    for_to(j,1,b-1)
      g[i][0]=g[i][0]*p+T[i][j];
    for_to(j,1,m-b)
      g[i][j]=(g[i][j-1]-T[i][j-1]*pot)*p+T[i][j+b-1];
  }
  p=31;
  pot=power(p,a-1);
  for_to(j,0,m-b)
  {
    h[0][j]=g[0][j];
    for_to(i,1,a-1)
      h[0][j]=h[0][j]*p+g[i][j];
    for_to(i,1,n-a)
      h[i][j]=(h[i-1][j]-g[i-1][j]*pot)*p+g[i+a-1][j];
  }
}

char T[MAX][MAX],P[MAX][MAX];
ui h[MAX][MAX],H[MAX][MAX];
int i,j,k,n,m;
int n_tests,test,x,y,ans;

int main()
{
  scanf("%d",&n_tests);
  for_to(test,1,n_tests)
  {
    scanf("%d %d",&n,&m);
    gets(T[0]);
    for_to(i,0,n-1)
      gets(T[i]);

    scanf("%d %d",&x,&y);
    gets(P[0]);
    hash(T,h,n,m,x,y);
    for_to(i,0,x-1)
      gets(P[i]);
    hash(P,H,x,y,x,y);
    ui v=H[0][0];
    ans=0;
    for_to(i,0,n-x)
      for_to(j,0,m-y)
        if (h[i][j]==v)
          ++ans;
    printf("%d\n",ans);
  }
  return 0;
}
```

2.2 Longest Common Subsequence (LCS)

```
Autor: Davi Costa/Marcelo Galvao Povoa
Complexidade: O(n*m)-calculo e O(n+m)-reconstrucao
Tempo de implementacao: 5 min
Testes: UVA.10066, UVA.10405
```

```
Dependencias: Nenhuma
Descricao: Calcula o tamanho de uma LCS entre duas strings e
reconstroi uma de tamanho qualquer (nao maior que
o maximo).
#include <stdio.h>
#include <string.h>
#define MAX 1234

/*Primeira e segunda strings (tamanhos m e n)*/
char seq[2][MAX+1];
int pd[MAX+1][MAX+1];

/*guarda o caminho*/
enum { cima, lado, diag } way[MAX+1][MAX+1];

int lcs(int m, int n) {
  int i,j;
  for (i = 0; i <= m; i++) pd[i][0] = 0;
  for (i = 0; i <= n; i++) pd[0][i] = 0;

  for (i = 1; i <= m; i++)
    for (j = 1; j <= n; j++) {
      if (seq[0][i-1] == seq[1][j-1]) {
pd[i][j] = pd[i-1][j-1] + 1;
way[i][j] = diag;
      }
      else if (pd[i-1][j] > pd[i][j-1]) {
pd[i][j] = pd[i-1][j];
way[i][j] = cima;
      }
      else {
pd[i][j] = pd[i][j-1];
way[i][j] = lado;
      }
    }
  return pd[m][n];
}

/*reconstroi uma CS, deve ser chamada com (m-1,n-1,tam desejado)*/
void printway(int i, int j, int k) {
  if (i==0 || j==0 || k==0) printf("\n");
  else if (way[i][j] == diag) {
    printway(i-1, j-1, k-1);
    printf("%c",seq[0][i]);
  }
  else if (way[i][j]==cima) printway(i-1, j, k-1);
  else printway(i, j-1, k-1);
}

/**** Exemplo simples de uso ****/
int main(){
  int n,m;
  while (1) {
    scanf(" %s",seq[0]);
    scanf(" %s",seq[1]);
    m=strlen(seq[0]);
    n=strlen(seq[1]);

    printf("%d\n",lcs(m,n));
  }
  return 0;
}
```

2.3 Longest Increasing Subsequence (LIS)

```
Autor: Marcelo Galvao Povoa
Complexidade: O(n*lg k), sendo k o tamanho da LIS
Tempo de implementacao: 5 min
Testes: UVA.231
Dependencias: Nenhuma
Descricao: Determina o tamanho da LIS do vetor v,
que pode ter numeros negativos, inclusive. Os trechos
de codigo comentados são  relativos apenas a parte
de reconstrucao de uma LIS. Esse algoritmo so funciona
quando a relacao entre dois elementos eh transitiva
(a < b e b < c => a < c), como acontece com
numeros, strings, etc.

#include <stdio.h>
#include <string.h>
#define MAXN 1000
#define INF 0x3f3f3f3f

int v[MAXN+1] /*,ant[MAXN+1],li[MAXN+1]*/ ;
int pd[MAXN+1] /*,ipd[MAXN+1]*/ ;
/*pd armazena o menor elemento que lide-
  ra uma IS de tamanho i ate o momento*/
```

```
int lis(int n) {
  int i,es,di,m,mx=0;

  memset(pd,0x3f,sizeof(pd));
  pd[0]=-INF;

  for (i=0;i<n;i++) {
    es=0; di=i;
    while (es<di) {
      m=(es+di+1)/2;
      if (pd[m]<v[i]) es=m;
      else di=m-1;
    }

    if (pd[es]<v[i] && pd[es+1]>v[i]) {
      pd[es+1]=v[i];
      if (es+1>mx) mx=es+1;
      /* ipd[es+1]=i;
         ant[i]=ipd[es];*/
    }
  }
  return mx;
}

/*reconstroi uma IS de tamanho tam depois de chamar lis(n)*/
/*void build(int tam) {
  int i,p;

  p=ipd[tam];

  if (pd[tam]==INF) printf("-1\n");
  else if (tam>0) {
  for (i=0;i<tam;i++) {
  li[i]=v[p];
  p=ant[p];
  }

  for (i=tam-1;i>0;i--) printf("%d ",li[i]);
  printf("%d\n",li[0]);
  }
  else printf("\n");
  }*/
```

```
/**** Exemplo simples de uso ****/
int main() {
  int n,i,k;

  scanf(" %d",&n);
  for (i=0;i<n;i++) scanf(" %d",&v[i]);

  k=lis(n);
  printf("%d\n",k);
  /*build(k);*/

  return 0;
}
```

2.4 Mochila

```
Autor: André Linhares
Complexidade: O(nC)

Teste: UVA 10130

Descrição: dado um conjunto de objetos, cada um com
um valor e um peso, achar o valor máximo que se pode
colocar numa mochila de determinada capacidade. Itens
repetidos podem ser permitidos pela alteração de uma
linha.
```

```
#include <iostream>
#include <cstring>

using namespace std;

#define MAXN 1000
#define MAXP 100
#define MAXCAP 30

int val[MAXN],weight[MAXN];
int cap[MAXP];
int T[MAXCAP+1];

void solve(int n_obj,int C)
```

```
{
  int i,k;
  memset(T,0,sizeof(T));
  for (k=0; k<n_obj; ++k)
    for (i=C; i >= weight[k]; --i)
      // se puder haver repetição, trocar por
      // for (i=weight[k]; i<=C; ++i)
      T[i] = max(T[i],T[i - weight[k]] + val[k]);;
}

int n_tests,test,n_obj,n_people;
int i,j,k,ans;

int main()
{
  scanf("%d",&n_tests);
  for (test=1; test<=n_tests; ++test)
  {
    scanf("%d",&n_obj);
    for (i=0; i<n_obj; ++i)
      scanf("%d %d",&val[i],&weight[i]);
    scanf("%d",&n_people);
    for (i=0; i<n_people; ++i)
      scanf("%d",&cap[i]);
    solve(n_obj,MAXCAP);
    ans=0;
    for (i=0; i<n_people; ++i)
      ans+=T[cap[i]];
    printf("%d\n",ans);
  }
  return 0;
}
```

2.5 Segmento de soma maxima

```
Autor: Marcelo Galvao Povoa
Complexidade: O(n)
Tempo de implementacao: 4 min
Testes: SPOJ.BAPOSTAS
Dependencias: Nenhuma
Descricao: Determina qual o segmento (subsequencia continua)
com a maior soma em um vetor. Indexado em 0.

#include <stdio.h>
#define MAXN 10000
#define INF 0x3f3f3f3f

int v[MAXN];
int mxe,mxd;

int segmax(int *v, int n) {
  int i,acc,esq,mx;

  acc=esq=0;
  mx=-INF;
  for (i=0;i<n;i++) {
    if (acc<0) {
      esq=i;
      acc=v[i];
    }
    else acc+=v[i];

    if (acc>mx) {
      mx=acc;
      mxe=esq;
      mxd=i;
    }
  }

  return mx;
}

/**** Exemplo simples de uso ****/
int main(){
  int n,i,r;

  scanf(" %d",&n);
  for (i=0;i<n;i++) scanf(" %d",&v[i]);

  r=segmax(v,n);
  printf("Um SegMax tem soma %d e vai de %d a %d\n",r,mxe,mxd);

  return 0;
}
```

3 Geométricos

3.1 Algoritmos basicos para Geometricos

```
Autor: Alexandre Kunieda (+ PUC-Rio)
Complexidade:
Tempo de implementacao: 2 minutos
Testes: ?
Dependencias: Nenhuma
Descricao:
Contem algoritmos simples para geometricos
```

```
/**
Estrutura de ponto e poligono aqui
**/

double polyarea(poly& p){ /* area com sinal */
  int i, n=p.size();
  double area = 0.0;

  for(i=0 ; i<n ; i++)
    area += p[i]%p[(i+1)%n];

  return area/2.0; /* area>0 = ccw ; area<0 = cw */
}

/* ponto p entre segmento [qr] */
int between3(point p, point q, point r){
  if(cmp((q-p)%(r-p)) == 0) /* colinear */
    if(cmp((q-p)*(r-p)) <= 0) /* < se os extremos nao contam */
      return 1;

  return 0;
}

int between(point q, reta r){ /*ponto em segmento de reta*/
  return between3(q, r.ini, r.fim);
}

/* rotaciona vetor 'p' em 'ang' radianos, em torno do ponto 'q' */
/* se 'q' nao especificado, rotaciona em torno da origem */
point rotate(point p, double ang, point q = point(0,0)) {
  double s = sin(ang), c = cos(ang);
  p = p-q;
  return q + point( p.x*c - p.y*s, p.x*s + p.y*c );
}

/**** Exemplo de uso ****/
int main() {
  vetor v,w;

  while(scanf(" %lf %lf", &v.x,&v.y)==2){
    w = rotate(v,pi/4);
    printf("%lf %lf\n", w.x,w.y);

    w = rotate(v,pi/4, point(1,1));
    printf("%lf %lf\n", w.x,w.y);
  }

  return 0;
}
```

3.2 Par de Pontos Mais Proximos

```
Autor: Notebook Unicamp (Mundial)
Complexidade: O(n*lg(n))
Tempo de implementacao: 2 minutos
Testes:
- UVa 10245 (n<=10000) t=0.290s
Dependencias:
- norma()
Descricao:
Obtem a menor distancia entre pontos de um dado conjunto de pontos
Eh preciso que o conjunto contenha pelo menos 2 pontos para o
algoritmo funcionar
```

```
#include <set>
#define foreach(it, a,b) for(typeof(a)it=(a) ; it!=(b) ; it++)
#define all(x) (x).begin(), (x).end()

bool ycmp(point a, point b) {
  if (a.y!=b.y) return a.y<b.y;
  return a.x<b.x;
}
```

```
double closest_pair (poly &P) {
  int n = P.size();
  double d = norma(P[0]-P[1]);
  set<point, bool(*)(point,point)> s(&ycmp);

  sort(all(P));
  for(int i=0,j=0 ; i<n ; i++) {
    point lower(0, P[i].y - d) , upper(0, P[i].y + d);
    while(P[i].x - P[j].x > d)
      s.erase(P[j++]);

    foreach(p, s.lower_bound(lower), s.upper_bound(upper))
      /* os pontos mais proximos sao tirados de P[i] e *p */
      d = min(d, norma(P[i] - *p));
    s.insert(P[i]);
  }
  return d;
}

/**** Exemplo de uso ****/
int main(){
  /**** especifico para o problema UVa 10245 ****/
  point i;
  poly p;
  int k,n;
  double d;

  while(scanf(" %d", &n)==1 && n) {
    p.clear();

    for(k=0 ; k<n ; k++) {
      scanf(" %lf %lf", &i.x,&i.y);
      p.push_back(i);
    }

    if(n==1) d = 15000.0;
    else d = closest_pair(p);

    if(d>10000) puts("INFINITY");
    else printf("%.4lf\n", d);
  }

  return 0;
}
```

3.3 Estrutura e base para Geometricos

```
Autor: Alexandre Kunieda (+ PUC-Rio)
Complexidade:
Tempo de implementacao: 5 minutos
Testes: ?
Dependencias: Nenhuma
Descricao:
Contem estrutura de ponto, reta, poligono e algumas
operacoes-base para os algoritmos geometricos


#include <math.h>
#include <vector>

using namespace std;

#define EPS (1e-8)
const double pi = acos(-1);

inline int cmp(double a, double b = 0){
  if(fabs(a-b)<=EPS) return 0;
  if(a<b) return -1;
  return 1;
}

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

  point operator +(point q){ return point(x + q.x, y + q.y); }
  point operator -(point q){ return point(x - q.x, y - q.y); }
  point operator *(double t){ return point(x * t, y * t); }
  point operator /(double t){ return point(x / t, y / t); }
  double operator *(point q){ return x * q.x + y * q.y; }
  double operator %(point q){ return x * q.y - y * q.x; }

  int cmp(point q) const {
    if (int t = ::cmp(x, q.x)) return t;
    return ::cmp(y, q.y);
  }
  bool operator ==(point q) const { return cmp(q) == 0; }
  bool operator !=(point q) const { return cmp(q) != 0; }
```

```
  bool operator < (point q) const { return cmp(q) < 0; }
};

struct reta {
  point ini,fim;
  reta(){}
  reta(point ini, point fim): ini(ini), fim(fim) {}
};
typedef vector<point> poly;
typedef point vetor;

vetor normal(vetor v){ return vetor(-v.y,v.x); }
double norma(vetor v){ return hypot(v.x, v.y); }
vetor versor(vetor v){ return v/norma(v); }
double anglex(vetor v){ return atan2(v.y, v.x); }
double angle(vetor v1, vetor v2){ /* angulo orientado ccw */
  return atan2(v1%v2 , v1*v2);
}
double triarea(point a, point b, point c){ /* area com sinal */
  return ((b-a)%(c-a))/2.0; /* area>0 = ccw ; area<0 = cw */
}
int ccw(point a, point b, point c){ /* b-a em relacao a c-a */
  return cmp((b-a)%(c-a)); /* ccw=1 ; cw=-1 ; colinear=0 */

  /* equivalente a cmp(triarea(a,b,c)), mas evita divisao */
}
vetor projecao(vetor v, vetor w){ /* proj de v em w */
  double alfa = (v*w)/(w*w);
  return w*alfa;
}

/**** Exemplo de uso ****/
int main() {
  point v,w,o;

  o = point(0,0);
  v = point(1,1);
  w = point(8,0);
  printf("norma(v)     : %lf\n", norma(v));
  printf("v*normal(v)  : %lf\n", v*normal(v));
  printf("anglex(v)    : %lf\n", anglex(v));
  printf("angle(v,w)   : %lf\n", angle(v,w));

  vetor p = projecao(v,w);
  printf("projecao(v,w): x=%lf, y=%lf\n", p.x, p.y);

  printf("triarea      : %lf\n", triarea(o, v,w));
  printf("ccw vw       : %d\n", ccw(o, v,w));
  printf("ccw wv       : %d\n", ccw(o, w,v));

  return 0;
}
```

3.4 Algoritmo de Graham (Convex Hull)

```
Autor: Alexandre Kunieda (+ PUC-Rio)
Complexidade: O(n*lg(n))
Tempo de implementacao: 4 minutos
Testes:
- UVa 218 (n<=?) t=0.110s
- UVa 596 (n<=400) t=0.010s
- UVa 10065 (n<=100) t=0.010s
- UVa 11096 (n<=100) t=0.000s
Dependencias:
- norma()
- ccw()
Descricao:
Algoritmo de Graham, para obter o Convex Hull de um dado
conjunto de pontos

/**
Estrutura de ponto e poligono aqui
**/

point pivo;
bool cmp_radial(point a, point b){ /* ordena em sentido horario */
  int aux = ccw(pivo, a,b);
  return ((aux<0) || (aux==0 && norma(a-pivo)<norma(b-pivo) ));
}
bool cmp_pivo(point p, point q){ /* pega o de menor x e y */
  int aux = cmp( p.x, q.x );
  return ((aux<0) || (aux==0 && cmp( p.y, q.y )<0));
}
/* colocar poly& p reduz tempo, mas desordena o conj de pontos */
poly graham(poly p){
  int i,j,n = p.size();
  poly g;
```

```
  /* ordena, tornando o conj de pontos um poligono estrelado */
  pivo = *min_element(p.begin(), p.end(), cmp_pivo);
  sort(p.begin(), p.end(), cmp_radial);

  /* para pegar colineares do final do poligono
   *
   * for(i=n-2 ; i>=0 && ccw(p[0], p[i], p[n-1])==0 ; i--);
   * reverse(p.begin()+i+1, p.end());
   */

  for(i=j=0 ; i<n ; i++) {
    /* trocar ccw>=0 por ccw>0 para pegar colineares */
    while( j>=2 && ccw(g[j-2], g[j-1], p[i]) >= 0 ){
      g.pop_back(); j--;
    }
    g.push_back(p[i]); j++;
  }

  return g;
}

/**** Exemplo de uso ****/
int main(){
  int i,n;
  poly p;
  point k;

  scanf(" %d", &n);
  while(n--){
    scanf(" %lf %lf", &k.x, &k.y);
    p.push_back(k);
  }

  poly g = graham(p);

  puts("poligono convexo:");
  n = g.size();
  for(i=0 ; i<n ; i++)
    printf("(%lf,%lf)\n", g[i].x, g[i].y);
  putchar('\n');

  puts("e nao re-ordenou o conjunto de pontos original:");
  n = p.size();
  for(i=0 ; i<n ; i++)
    printf("(%lf,%lf)\n", p[i].x, p[i].y);
  putchar('\n');

  return 0;
}
```

3.5 Verificacoes de Pontos em Poligonos

```
Autor: Alexandre Kunieda (+ PUC-Rio + note da Mundial)
Complexidade: O(n), O(n), O(lg(n))
Tempo de implementacao: 1 minuto (cada)
Testes:
- inpoly() e inpoly_big():
+ UVa 634 (n<=1000) [inpoly]t=0.010s, [inpoly_big]t=0.010s
- inpoly_convex():
+ para o poligono, foi criado um conjunto de pontos (n=200.000),
pertencentes aa borda de uma circunferencia de raio R=50.000,
entao foram gerados 100 pontos aleatorios no quadrado de lado 2R
que contem a circunferencia. os resultados foram:
+ inpoly()        tempo= 1m35.434s
+ inpoly_big()    tempo= 47.431s
+ inpoly_convex() tempo= 0.000s
Dependencias:
- angle() - soh inpoly()
- ccw()
- between3()
- intri() - soh inpoly_convex()
Descricao:


/**
Estrutura de ponto e poligono aqui
**/

int intri(point k, point a, point b, point c){
  int a1,a2,a3;

  a1 = ccw(a,k,b);
  a2 = ccw(b,k,c);
  a3 = ccw(c,k,a);

  if((a1*a2)>0 && (a2*a3)>0) return 1; /*dentro*/
```

```
  if(between3(k,a,b) || between3(k,b,c) || between3(k,c,a))
    return 2; /*borda*/
  return 0; /*fora*/
}

/* tempo= 1m35.434s */
/* verificacao a partir de calculo de angulos */
int inpoly(point k, poly& p){
  int i, n = p.size();
  double ang = 0.0;

  for(i=1 ; i<=n ; i++){
    ang += angle(p[i-1]-k , p[i%n]-k);
    if(between3(k, p[i-1],p[i%n])) return 2; /*borda*/
  }

  return cmp(fabs(ang),pi)<0 ? 0 : 1;
}

/* tempo= 47.431s */
/* para evitar erro de precisao com calculo de angulos */
int inpoly_big (point k, poly &p){
  int n = p.size();
  int cross = 0;

  for(int i=1; i<=n ; i++) {
    point q=p[i-1], r=p[i%n];

    if( between3(k,q,r) ) return 2;
    if( q.y>r.y ) swap(q,r);
    if( q.y<k.y && r.y>=k.y  && ccw(k,q,r)>0 ) cross++;
  }

  return cross%2;
}

/* tempo= 0.000s */
/* O(lg(n)) - soh para poligonos convexos */
int inpoly_convex(point k, poly& p){
  /* 'val' indica o sentido do poligono */
  int val = ccw(p[0],p[1],p[2]);
  /* tomar cuidado para o caso em que o poligono
     comeca com pontos colineares, 'val' recebera 0 */
  int esq,dir,meio, n = p.size();

  esq = 1; dir = n-1;
  while(dir>esq+1) {
    meio = (esq+dir)/2;

    if(ccw(p[0],p[meio],k) == val) esq = meio;
    else dir = meio;
  }

  return intri(k, p[0],p[esq],p[dir]);

  /* caso seja preciso verificar se esta na borda,
   * substituir o return por:
   *
   * if(between3(k,p[esq],p[dir]) ||
   *    between3(k,p[0],p[1])     ||
   *    between3(k,p[0],p[n-1])) return 2;  //BORDA
   * return intri(k, p[0],p[esq],p[dir])?1:0;  //DENTRO:FORA
   */
}

/**** Exemplo de uso ****/
int main() {
  point k;
  poly p;
  int n;
  int val;

  scanf(" %d", &n);
  while(n--) {
    scanf(" %lf %lf", &k.x,&k.y);
    p.push_back(k);
  }

  scanf(" %lf %lf", &k.x,&k.y);

  val = inpoly(k,p);
  /*val = inpoly_convex(k,p);*/
  /*val = inpoly_big(k,p);*/
  printf("%d\n", val);

  return 0;
}
```

3.6 Algoritmos de interseccoes

```
Autor: Alexandre Kunieda + (PUC-Rio)
Complexidade:
Tempo de implementacao: ?
Testes:
- UVa 11068 [intersect] [acha] t=0.010s
- UVa 866 [intersect_seg] [intersect_seg_2] [acha] t=0.000s
- UVa 378 [intersect] [acha] t=0.010s
- UVa 191 [intersect_seg] [intersect_seg_2] t=0.000s
Dependencias:
- comparacoes na estrutura de ponto - soh intersect_seg()
- norma() - soh distPR()
- projecao() - soh distPR()
- between() - soh distPR()
- between3() - soh intersect_seg_2()
- ccw() - soh intersect_seg_2()
Descricao:
Determina se ha interseccao ou o ponto de interseccao
entre segmentos de reta ou retas.
Tambem contem funcao que devolve a distancia de um ponto
a uma reta.


/**
Estrutura de ponto e poligono aqui
**/

struct eq_reta {
  double A,B,C;  /* Ax + By + C = 0 */

  void init(reta p){
    point aux = p.ini - p.fim;
    A = aux.y;
    B = -aux.x;
    C = -A*p.ini.x - B*p.ini.y;
  }
  eq_reta(reta p){ init(p); }
};

int intersect(reta p0, reta q0){ /*interseccao de retas*/
  eq_reta p(p0), q(q0);
  if(cmp(p.A*q.B , p.B*q.A)==0){ /*paralelos*/
    if(cmp(p.A*q.C , p.C*q.A)==0 &&
       cmp(p.B*q.C , p.C*q.B)==0) return 2; /*reta*/
    else return 0; /*nada*/
  }
  return 1; /*ponto*/
}

/* interseccao nos extremos dos segmentos tbm eh contada! */
bool intersect_seg(point p, point q, point r, point s) {
  point A = q - p, B = s - r, C = r - p, D = s - q;
  int a = cmp(A % C) + 2 * cmp(A % D);
  int b = cmp(B % C) + 2 * cmp(B % D);
  if (a == 3 || a == -3 || b == 3 || b == -3) return false;
  if (a || b || p == r || p == s || q == r || q == s) return true;
  int t = (p < r) + (p < s) + (q < r) + (q < s);
  return t != 0 && t != 4;
}

/* interseccao nos extremos dos segmentos tbm eh contada! */
bool intersect_seg_2(point p, point q, point r, point s) {
  int a = ccw(p,q,r)*ccw(p,q,s);
  int b = ccw(r,s,p)*ccw(r,s,q);

  if(a>0 || b>0) return false;
  if(a<0 && b<0) return true;

  return ( between3(p,r,s) ||
   between3(q,r,s) ||
   between3(r,p,q) ||
   between3(s,p,q) );
}

/*acha interseccao de duas retas*/
point acha(point a, point b, point c, point d){
  /* pressupoe que haja interseccao! */
  eq_reta p(reta(a,b)), q(reta(c,d));
  point k;

  k.x = (q.C*p.B - p.C*q.B)/(p.A*q.B - q.A*p.B);
  k.y = (q.C*p.A - p.C*q.A)/(p.B*q.A - q.B*p.A);
  return k;
}
```

```
/*acha interseccao de duas retas - da PUC*/
point acha_(point p, point q, point r, point s){
  point a = q-p, b = s-r, c = point(p%q,r%s);
  return point(point(a.x, b.x)%c, point(a.y, b.y)%c) / (a%b);
}

/* distancia de um ponto a uma reta */
double distPR(point p, reta r){
  vetor v = p - r.ini;
  vetor w = r.fim - r.ini;

  vetor proj = projecao(v,w);
  /* (proj+r.ini) eh o ponto mais proximo de p,
     e que pertencente aa reta r */

  /* para segmentos de reta
   *
   * if( !between(proj+r.ini, r) )
   *   return min( norma(p-r.ini), norma(p-r.fim) );
   */

  return norma(v - proj);
}

/**** Exemplo de uso ****/
int main(){
  /**** Especifico do problema UVa 378 ****/
  int n, i,aux;
  reta r[2];
  point p;

  puts("INTERSECTING LINES OUTPUT");

  scanf(" %d", &n);
  while(n--){
    for(i=0 ; i<2 ; i++)
      scanf(" %lf %lf %lf %lf",
    &r[i].ini.x, &r[i].ini.y,
    &r[i].fim.x, &r[i].fim.y);

    aux = intersect(r[0], r[1]);

    if(aux == 0) puts("NONE");
    if(aux == 1){
      p = acha(r[0].ini,r[0].fim, r[1].ini,r[1].fim);
      printf("POINT %.2lf %.2lf\n", p.x,p.y);
    }
    if(aux == 2) puts("LINE");
  }

  puts("END OF OUTPUT");

  return 0;
}
```

3.7 Algoritmo de Intersecao de Poligonos Convexos

```
Autor: PUC-Rio
Complexidade: O(n+m)
Tempo de implementacao: 8 min
Testes:
- UVa 137 (n<=100) t=0.002s
Dependencias:
- ccw()
- between3()
- intersect_seg()
- acha()
- inpoly()
Descricao:
O algoritmo devolve a interseccao de dois poligonos convexos,
orientados em sentido anti-horario


#define all(x) (x).begin(),(x).end()

/* os poligonos P e Q devem estar orientados em
   sentido anti-horario! */
poly poly_intersect(poly& P, poly& Q) {
  int m = Q.size(), n = P.size();
  int a = 0, b = 0, aa = 0, ba = 0, inflag = 0;
  poly R;
  while ((aa < n || ba < m) && aa < 2*n && ba < 2*m) {
    point p1 = P[a], p2 = P[(a+1) % n],
      q1 = Q[b], q2 = Q[(b+1) % m];
    point A = p2 - p1, B = q2 - q1;
    int cross = cmp(A % B), ha = ccw(p2, q2, p1),
      hb = ccw(q2, p2, q1);
```

```
    if (cross == 0 && ccw(p1, q1, p2) == 0 && cmp(A * B) < 0) {
      if (between3(p1, q1, p2)) R.push_back(q1);
      if (between3(p1, q2, p2)) R.push_back(q2);
      if (between3(q1, p1, q2)) R.push_back(p1);
      if (between3(q1, p2, q2)) R.push_back(p2);
      if (R.size() < 2) return poly();
      inflag = 1; break;
    } else if (cross != 0 && intersect_seg(p1, p2, q1, q2)) {
      if (inflag == 0) aa = ba = 0;
      R.push_back(acha(p1, p2, q1, q2));
      inflag = (hb > 0) ? 1 : -1;
    }
    if (cross == 0 && hb < 0 && ha < 0) return R;
    bool t = cross == 0 && hb == 0 && ha == 0;
    if (t ? (inflag == 1) : (cross >= 0) ? (ha <= 0) : (hb > 0)) {
      if (inflag == -1) R.push_back(q2);
      ba++; b++; b %= m;
    } else {
      if (inflag == 1) R.push_back(p2);
      aa++; a++; a %= n;
    }
  }
  if (inflag == 0) {
    if (inpoly(P[0], Q)) return P;
    if (inpoly(Q[0], P)) return Q;
  }
  R.erase(unique(all(R)), R.end());
  if (R.size() > 1 && R.front() == R.back()) R.pop_back();
  return R;
}

/**** Exemplo simples de uso ****/
int main() {
  return 0;
}
```

3.8 Circulo Gerador Minimo

```
Autor: PUC-Rio
Complexidade: O(n^3)
Tempo de implementacao: 3 min
Testes:
- SPOJbr ICPC (n<=100) t=0.35s
- UVa 10005 (n<=100) t=0.002s
Dependencias:
- norma()
Descricao:
O algoritmo devolve o circulo de raio minimo que contem
todos os pontos dados
```

```
#define all(x) (x).begin(),(x).end()
typedef pair<point,double> circle;

bool in_circle(circle C, point p){
  return cmp(norma(p - C.first), C.second) <= 0;
}
point circumcenter(point p, point q, point r) {
  point a = p - r, b = q - r,
    c = point(a * (p + r) / 2, b * (q + r) / 2);

  return point(c % point(a.y, b.y),point(a.x, b.x) % c) / (a % b);
}

circle spanning_circle(vector<point>& T) {
  int n = T.size();
  random_shuffle(all(T));

  circle C(point(), -INFINITY);
  for (int i = 0; i < n; i++) if (!in_circle(C, T[i])) {
    C = circle(T[i], 0);
    for (int j = 0; j < i; j++) if (!in_circle(C, T[j])) {
      C = circle((T[i] + T[j]) / 2, norma(T[i] - T[j]) / 2);
      for (int k = 0; k < j; k++) if (!in_circle(C, T[k])) {
point o = circumcenter(T[i], T[j], T[k]);
C = circle(o, norma(o - T[k]));
}
      }
    }

  return C;
}
```

4 Numéricos

4.1 Crivo de Erastotenes

```
Autor: Felipe Sodre
Complexidade: O(N log log N)
Tempo de implementacao: 2 minutos
Testes: todo(fsodre)
Dependencias: Nenhuma
Descricao:
Popula o array pr, de tal forma que pr[i] eh
verdadeiro se i eh primo.
```

```
#include <iostream>
#include <cstdio>

// Numero maximo a ser analisado
#define MAXN 1123123

// se pr[i] == true, i eh primo
bool pr[MAXN+1];

// algum divisor primo de i. Para fatoracao.
int divisor[MAXN+1];

// Analisa primalidade no intervalo [1,n]
void crivo(int n) {
  memset(pr, true, n * sizeof(int));
  pr[0] = pr[1] = false;
  for(int i = 2; i*i <= n; i++){
    if( !pr[i] || !(i&1) && i > 2) continue;
    int k = i*i;
    divisor[i] = i;
    while(k <= n){
      pr[k] = false;
      divisor[k] = i;
      k += i;
    }
  }
}


/**** Exemplo simples de uso ****/
int main(void){
  crivo(500);
  if(pr[2]) printf("2 eh primo\n");
  if(pr[9]) printf("9 eh primo\n");

  return 0;
}
```

4.2 Algoritmo de Euclides Extendido

```
Autor: NU 2/Marcelo Galvão Póvoa
Complexidade: O(lg x)
Tempo de implementacao: 1 min
Testes: SPOJ.DIOFANTO
Dependencias: Nenhuma
Descricao: Calcula um par x,y tal que a*x+b*y=mdc(a,b)
#include <algorithm>
using namespace std;

typedef pair<int,int> pii;

pii mdc(int a, int b){
  if (b == 0) return pii(1,0);
  pii u = mdc (b,a%b);
  return pii(u.second, u.first - (a/b)*u.second);
}

/**** Exemplo simples de uso ****/
int main(){
  int a,b;
  pii euext;

  scanf(" %d %d",&a,&b);
  euext=mdc(a,b);

  printf("%d %d\n",euext.first,euext.second);
  return 0;
}
```

4.3 Fatoracao de numero inteiro

```
Autor: Felipe Sodre
Complexidade: O(log N)
Tempo de implementacao: 1 minuto
Testes: todo(fsodre)
```

```
Dependencias:
- Crivo de Erastotenes
Descricao:
fatora(n, arr) coloca no array arr todos os fatores
primos de n, nao necessariamente em ordem. Retorna
a quantidade de fatores primos.

#include <iostream>
#include <cstdio>

/**
   Crivo aqui
**/

inline int div(int n){
  if(pr[n]) return n;
  return divisor[n];
}

int fatora(int n, int fatores[]){
  if(n <= 1){
    fatores[0] = n;
    return 0;
  }

  int k = 0;
  while(n > 1){
    fatores[k++] = div(n);
    n /= div(n);
  }
  return k;
}

/**** Exemplo simples de uso ****/
int main(void){
  int nums[15];

  crivo(500);

  int qt = fatora(444, nums);
  for(int i = 0; i < qt; i++){
    printf("%d eh um fator de 444.\n",nums[i]);
  }
  return 0;
}
```

4.4 Fibonacci (logaritmico)

```
Autor: Marcelo Galvão Póvoa
Complexidade: O(lg*lg x)
Tempo de implementacao: 4 min
Testes: SPOJ.RABBIT1
Dependencias: Nenhuma
Descricao: Determina f(x) % MD com f(1)=1 e f(2)=1
Chamar init() antes de usar a fib(x)
#include <map>
#define MD 1000000
using namespace std;

typedef long long lli;

map<lli,lli> M;

void init() {
   M.clear();
   M[0]=0;
   M[1]=M[2]=1;
}

lli fib(lli x) {
   lli k,a,b;

   if (M.find(x)!=M.end()) return M[x];

   k=(x+1)/2;
   a=fib(k);
   b=fib(k-1);

   if (x % 2 == 0) return M[x]=((a*a) % MD + (2*a*b) % MD) % MD;
   return M[x]=((a*a) % MD + (b*b) % MD) % MD;
}


/**** Exemplo simples de uso ****/
int main() {
   lli x;
```

```
   init();
   scanf(" %lld",&x);
   printf("%lld\n",fib(x));
   return 0;
}
```

4.5 Inverso Modular

```
Autor: NU2/Marcelo Galvão Póvoa
Complexidade: O(lg x)
Tempo de implementacao: 3 min
Testes: SPOJ.DIOFANTO
Dependencias: Euclides Extendido
Descricao: Calcula um x tal que a*x === 1 (mod M)
Para a e M coprimos, eh garantido que x eh unico
Nesse caso, pode ser usado para determinar
a divisao modular como exemplificado.
#include <algorithm>
using namespace std;

typedef pair<int,int> pii;

pii mdc(int a, int b){
  if (b == 0) return pii(1,0);
  pii u = mdc (b,a%b);
  return pii(u.second, u.first - (a/b)*u.second);
}

int invmod(int a, int M) {
  pii r=mdc(a,M);
  if (r.first * a + r.second * M == 1)
    return (r.first + M) % M;
  return 0;
}

/**** Exemplo simples de uso ****/
int main(){
  int x,m;

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

  /*retorna 36/x (mod m), se x eh divisor de 36*/
  //printf("%d\n",36*invmod(x,m) % m);
  printf("%d\n",invmod(x,m));
}
```

4.6 Sistema de equações lineares

```
Complexidade: O(n^3)
Tempo de implementacao: 5 minutos
Descricao:
Resolve o sistema mat*x=b, guardando a solução em ans.
Detecta se há soluções múltiplas ou
se o sistema não tem solução.
m equações e n variaveis

#include <iostream>
#include <vector>
#include <cmath>

#define FOR(i,n) for(int i=0; i<n; ++i)

using namespace std;

typedef vector<double> seq_t;
bool GaussJordan(int n, int m, vector<seq_t> mat,
 seq_t b, seq_t &ans)
{
  #define M(i,j) (mat[idx[i]][j])
  vector<int> idx(m);
  FOR(i,m) idx[i]=i;
  FOR(i,m) mat[i].push_back(b[i]);
  int step=0;
  FOR(i,n){
    int p = -1;  double pivot = 1e-8;
    for(int j=step;j<m;j++)
      if(abs(M(j,i)) > abs(pivot)){
        p = j;
        pivot = M(j,i);
      }
    if(p==-1) continue;
    swap(idx[step],idx[p]);
    FOR(j,n+1)  M(step,j) /= pivot;
    FOR(j,m) if(step!=j){
      double w=M(j,i);
      FOR(k,n+1) M(j,k) -= w * M(step,k);
    }
    step++;
```

```
  }
  for(int i=step;i<m;i++)
    if(abs(M(i,n))>1e-8) throw "No Sol.";
  if(step==n){
    ans.resize(n); FOR(i,n) ans[i] = M(i,n);
    return true;
  }else{
    return false; // there are Multiple(n-step) Solution.
  }
}

int main()
{
  return 0;
}
```

4.7 Simplex

```
Descricao:
Resolve o problema de programação linear:
minimizar cx
sujeito a Ax = b
x >= 0

// UVA 10498, Happiness
#include <iostream>
#include <cstdio>
#include <cmath>
#include <vector>

using namespace std;

typedef vector<double> array;
typedef vector<array> matrix;

const double EPS = 1e-8;
enum { OPTIMAL, UNBOUNDED, NOSOLUTION, UNKNOWN };
struct two_stage_simplex {
  int N, M, st;
  matrix a;
  vector<int> s;
  two_stage_simplex(const matrix &A, const array &b,
    const array &c)
    : N(A.size()), M(A[0].size()), a(N+2, array(M+N+1)),
      s(N+2), st(UNKNOWN) {
    for (int j = 0; j < M; ++j) a[N+1][j] = c[j];
    // make simplex table
    for (int i = 0; i < N; ++i)
      for (int j = 0; j < M; ++j) a[i+1][j] = A[i][j];
    for (int i = 0; i < N; ++i) a[i+1][M+N] = b[i];
    // add helper table
    for (int i = 0; i < N; ++i) a[ 0 ][i+M] = 1;
    for (int i = 0; i < N; ++i) a[i+1][i+M] = 1;
    for (int i = 0; i < N; ++i) s[i+1]      = i+M;
    for (int i = 1; i <= N; ++i)
      for (int j = 0; j <= N+M; ++j) a[0][j] += a[i][j];
    st = solve();
  }
  int status() const { return st; }
  double solution() const { return -a[0][M]; }
  double solution(array &x) const {
    x.resize(M, 0);
    for (int i = 0; i < N; ++i)
      x[s[i+1]] = a[i+1].back();
    return -a[0][M];
  }
  int solve() {
    M += N; N += 1;
    solve_sub(); // solve stage one
    if (solution() > EPS) return NOSOLUTION;
    N -= 1; M -= N;
    swap(a[0], a.back()); a.pop_back(); // modify table
    for (int i = 0; i <= N; ++i) {
      swap(a[i][M], a[i].back());
      a[i].resize(M+1);
    }
    return solve_sub(); // solve stage two
  }
  int solve_sub() {
    int p, q;
    while (1) {
      //print();
      for (q = 0; q <= M && a[0][q] >= -EPS; ++q);
      for (p = 0; p <= N && a[p][q] <=  EPS; ++p);
      if (q >= M || p > N) break;
      for (int i = p+1; i <= N; ++i)
```

```
        // bland's care for cyclation
        if (a[i][q] > EPS)
          if (a[i][M]/a[i][q] < a[p][M]/a[p][q] ||
      (a[i][M]/a[i][q] == a[p][M]/a[p][q] &&
       s[i] < s[q])) p = i;
        pivot(p, q);
      }
      if (q >= M) return OPTIMAL;
      else        return UNBOUNDED;
    }
    void pivot(int p, int q) {
      for (int j = 0; j <= N; ++j)
        for (int k = M; k >= 0; --k)
          if (j != p && k != q)
            a[j][k] -= a[p][k]*a[j][q]/a[p][q];
      for (int j = 0; j <= N; ++j)
        if (j != p) a[j][q] = 0;
      for (int k = 0; k <= M; ++k)
        if (k != q) a[p][k] = a[p][k]/a[p][q];
      a[p][q] = 1.0;
      s[p] = q;
    }
};

int main() {
  for (int n, m; cin >> n >> m; ) {
    array c(n+m), b(m);
    for (int i = 0; i < n; ++i)
      cin >> c[i], c[i] *= -1;
    matrix A(m, array(n+m));
    for (int i = 0; i < m; ++i) {
      for (int j = 0; j < n; ++j)
        cin >> A[i][j];
      A[i][n+i] = 1;
      cin >> b[i];
    }
    two_stage_simplex tss(A, b, c);
    double ans = -tss.solution() * m;
    printf("Nasa can spend %.0f taka.\n", ans + 0.5 - EPS);
  }
}
```

4.8 Teorema Chines do Resto

```
Autor: NU 2/Marcelo Galvão Póvoa
Complexidade: ?
Tempo de implementacao: 4 min
Testes: Testes proprios
Dependencias: Algoritmo de Euclides Extendido
Descricao: Resolve o conj de equacoes a[i]*x === b[i] (mod m[i])
para 0<=i<n com a restricao m[i]>1
#include <algorithm>
#include <vector>
#define MAXN 1000
using namespace std;

int n,a[MAXN],b[MAXN],m[MAXN];
typedef pair<int,int> pii;

int chines() {
   int x = 0, M = 1;
   for (int i=0; i<n; i++) {
     int b2 = b[i] - a[i]*x;
     pii bizu = mdc(a[i]*M, m[i]);
     int g = a[i]*M * bizu.first + m[i] * bizu.second;

     if (b2 % g) return -1;
     x += M *(bizu.first * (b2/g) % (m[i]/g));
     M *= (m[i]/g);
   }
   return (x%M+M)%M;
}

/**** Exemplo simples de uso ****/
int main(){
   int i;

   scanf(" %d",&n);
   for (i=0;i<n;i++) scanf(" %d %d %d",&a[i],&b[i],&m[i]);

   printf("%d\n",chines());
   return 0;
}
```

4.9 Teste de primalidade

```
Autor: PUC
Complexidade: O(sqrt(n))
Descrição: retorna true se n e primo,
false caso contrario. Considera primos negativos.
```

```
#include <iostream>
#include <cmath>

using namespace std;

bool is_prime(int n) {
  if (n < 0) return is_prime(-n);
  if (n < 5 || n % 2 == 0 || n % 3 == 0)
    return (n == 2 || n == 3);
  int maxP = sqrt(n) + 2;
  for (int p = 5; p < maxP; p += 6)
    if (n % p == 0 || n % (p+2) == 0) return false;
    return true;
}

int main()
{
  int x;
  while (1)
  {
    scanf("%d",&x);
    printf("%d e ",x);
    if (is_prime(x)) printf("primo.\n");
    else printf("composto.\n");
  }
  return 0;
}
```

5 Strings

5.1 Aho-Corasick

```
Autor: Felipe Sodre Silva
```

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <set>
#include <map>
#include <queue>

using namespace std;

/* Tamanho do alfabeto */
#define QT_ALPHA 53

/* Quantidade maxima de padroes analisados */
#define MAX_N 1234

/* Tamanho maximo de um padrao */
#define MAX_SZ 1234

/* Quantidade maxima de estados no automato */
#define QT_STATE (QT_ALPHA*MAX_SZ+2)

/* Helper para pegar uma aresta no automato */
#define GT(sta, c) go_to[sta][hash(c)]

struct aho{
  int go_to[QT_STATE][QT_ALPHA];
  vector<int> out[QT_STATE];
  vector<string> m_vs;
  string alpha;
  int fail[QT_STATE];
  int qts;
  aho(): qts(1){ }

  /* Reseta a estrutura */
  void clear(){
    for(int i = 0; i < QT_STATE; i++){
      out[i].clear();
    }
    memset(go_to,-1,sizeof(go_to));
    m_vs.clear();
    qts=1;

  }
```

```
  /* Funcao de hash que retorna a posicao de um caractere
     dentro de alpha. Deve ser customizada para cada diferente
     alpha */
  int hash(char c){
    if(c <= 'Z') return c-'A';
    return ('Z'-'A'+1) + c - 'a';
  }

  /* Configura o alfabeto a ser considerado */
  void set_alpha(string & s){ alpha = s; }


  /* configura o vetor com os padrões a serem considerados */
  void set_pads(vector<string> & vs){
    m_vs = vs;
    for(int j = 0; j < vs.size(); j++){
      string s = vs[j];
      int n = s.size();
      int sta = 0;
      for(int i = 0; i < n; i++){
if(GT(sta, s[i]) > -1){
  sta = GT(sta, s[i]);
}
else{
  GT(sta, s[i]) = qts;
  sta = qts++;
}
      }
      out[sta].push_back(j);
    }
  }

  /* Gera as arestas de falha no automato */
  void make_fail(){
    queue<int> q;

    for(int i = 0; i < alpha.size(); i++){
      if(GT(0,alpha[i]) == -1) GT(0, alpha[i]) = 0;
    }

    for(int i = 0; i < alpha.size(); i++){
      int an = GT(0, alpha[i]);
      if(an <= 0) continue;
      fail[an] = 0;
      q.push(an);
    }
    while(!q.empty()){
      int k = q.front();
      q.pop();
      for(int i = 0; i < alpha.size(); i++){
int an = GT(k, alpha[i]);
if(an == -1) continue;
q.push(an);
int v = fail[k];
while(GT(v,alpha[i]) == -1) v=fail[v];
fail[an] = GT(v,alpha[i]);
//printf("an=%d fail[an]=%d\n",an,fail[an]);
int nn = out[fail[an]].size();
for(int i = 0; i < nn; i++) out[an].push_back(out[fail[an]][i]);
      }
    }
  }

  /* Identifica os padroes configurados na string s.
     res[i] contem o indice do padrao achado, e end_at[i]
     contem o índice final do padrão na string procurada. */

  void query(string & s, vector<int> &res, vector<int> & end_at){
    int sta=0;
    int n = s.size();
    for(int i = 0; i < n; i++){
      while(GT(sta, s[i]) == -1){
sta = fail[sta];
      }
      sta = GT(sta, s[i]);
      for(int j = 0; j < out[sta].size(); j++){
res.push_back(out[sta][j]);
end_at.push_back(i);
      }
    }
  }

  /* Identifica os padroes configurados na string s.
     res[i] contem o indice do padrao achado, e end_at[i]
     contem o índice final do padrão na string procurada. */

  void query_has(string & s, vector<bool> & vb){
    int sta=0;
```

```
    int n = s.size();
    for(int i = 0; i < n; i++){
      while(GT(sta, s[i]) == -1){
sta = fail[sta];
      }
      sta = GT(sta, s[i]);
      for(int j = 0; j < out[sta].size(); j++){
vb[out[sta][j]] = true;
      }
    }
  }
};


/*********************************
Exemplo
*********************************/

aho a;

int main(void){


  /* O alfabeto considerado é o das letras minusculas
     e maíusculas */

  string alpha;
  for(char c = 'a'; c <= 'z'; c++) alpha.push_back(c);
  for(char c = 'A'; c <= 'Z'; c++) alpha.push_back(c);
  a.set_alpha(alpha);


  int qtcases;
  scanf("%d", &qtcases);

  while(qtcases--){
    int qtpad;
    string pad, search;

    /* Reseta a estrutura */
    a.clear();


    /* Le a string de busca */
    cin >> search;

    scanf("%d",&qtpad);
    vector<string> padroes;

    /* Lê os padrões */
    for(int i = 0; i < qtpad; i++){
      char buff[1000];
      scanf("%s",buff);
      string s = string(buff);
      padroes.push_back(s);
    }

    /* configura os padrões */
    a.set_pads(padroes);

    /* Configura o automato */
    a.make_fail();

    /* Vector que vai abrigar os indices dos padroes encontrados*/
    vector<int> res;

    /* vector que vai abrigar os ultimos indices */
    vector<int> ends_at;

    vector<bool> vb(qtpad);

    /* Verifica a existencia dos padroes na string search */
    // a.query(search, res, ends_at);
    a.query_has(search, vb);

    for(int i = 0; i < qtpad; i++){
      if(vb[i]) printf("y\n");
      else printf("n\n");
    }
  }

  return 0;

}
```

5.2 Knuth Morris Pratt (KMP)

```
Autor: NU 2/Marcelo Galvão Póvoa
Complexidade: O(n+m)
Tempo de implementacao: 3 min
Testes: SPOJ.NHAY
Dependencias: Nenhuma
Descricao: Acha todas as ocorrencias de um padrao p num texto t
#include <string.h>
#include <stdio.h>
#define MAXNP 1000

int fail[MAXNP];

void buildFail(char *p) {
   int m = strlen(p);
   int j = fail[0] = -1;
   for ( int i = 1; i <= m; i++) {
      while (j >= 0 && p[j] != p[i-1]) j = fail[j];
      fail[i] = ++j;
   }
}

void kmp(char *p, char *t) {
   int m = strlen(p), n = strlen(t);
   buildFail(p);
   for ( int i = 0, k = 0; i < n; i++) {
      while (k >=0 && p[k] != t[i]) k = fail[k];
      if ( ++k >= m ){
         /*achou em t[i-m+1 .. i]*/
         k = fail[k];
      }
   }
}

/**** Exemplo simples de uso ****/
int main(){
   char pad[10000],tex[100000];

   while (scanf(" %s",pad)==1) {
      scanf(" %s",tex);
      kmp(pad,tex);
   }
   return 0;
}
```

6 Miscelânea

6.1 Binary Indexed Tree/Fenwick Tree

```
Autor: Marcelo Galvao Povoa
Complexidade: O(lg MAX) - atualizacao e consulta
Tempo de implementacao: 2 min
Testes: SPOJ.ORDERS, SPOJ.INCDSEQ
Dependencias: Nenhuma
Descricao: Dada um vetor inicialmente vazio, eh possivel fazer
incremento no conteudo v[x] e consultar a soma do
subvetor v[1..x] de forma eficiente. Para o calculo
da soma de um subvetor qualquer usa-se a relacao
sum(a..b) =  sum(1..b) - sum(1..a-1)

Observacao: Zerar o vetor tree[] antes de utilizar

#include <stdio.h>
#include <string.h>
#define MAX 1000

int tree[MAX+1]; /*nao usar posicao em 0*/

int query(int x) {
  int sum=0;

  while (x>0) {
    sum+=tree[x];
    x-=x & (-x);
  }
  return sum;
}

/*v representa um inc/decremento em x!*/
void update(int x, int v) {
  if (x==0) return;

  while (x<=MAX) {
    tree[x]+=v;
    x+=x & (-x);
  }
}
```

```
/**** Exemplo simples de uso ****/
int main(){
  memset(tree,0,sizeof(tree));

  update(3,2);
  update(2,-1);

  printf("%d\n",query(10));
  return 0;
}
```

6.2 Binary Indexed Tree/Fenwick Tree 2D

```
Autor: Marcelo Galvao Povoa
Complexidade: O(lg*lg MAX) - atualizacao e consulta
Tempo de implementacao: 3 min
Testes: SPOJ.MATSUM
Dependencias: Nenhuma
Descricao: Dada uma matriz inicialmente vazia, eh possivel fazer
incremento no conteudo m[x,y] e consultar a soma da
submatriz m[1..x,1..y] de forma eficiente. Para o cal
culo da soma de uma submatriz qualquer usa-se a relacao
sum(a..b,c..d) =  sum(1..b,1..d) + sum(1..a-1,1..c-1)
- sum(1..b,1..c-1) - sum(1..a-1,1..d)

Observacao: Zerar a matriz tree[][] antes de utilizar
```

```
#include <stdio.h>
#include <string.h>
#define MAX 1000

int tree[MAX+1][MAX+1]; /*nao usar posicao em 0*/

int query(int x, int y) {
  int sum=0,yy=y;
  if (x==0 || y==0) return 0;

  while (x) {
    while (y) {
      sum+=tree[x][y];
      y-=y & (-y);
    }
    x-=x & (-x);
    y=yy;
  }
  return sum;
}

/*v representa um inc/decremento em x,y!*/
void update(int x, int y, int v) {
  int yy=y;
  if (x==0 || y==0) return;

  while (x<=MAX) {
    while (y<=MAX) {
      tree[x][y]+=v;
      y+=y & (-y);
    }
    x+=x & (-x);
    y=yy;
  }
}


/**** Exemplo simples de uso ****/
int main(){
  memset(tree,0,sizeof(tree));

  update(3,5,2);
  update(2,3,-1);

  printf("%d\n",query(10,10));
  return 0;
}
```

6.3 Operações com bits

```
Autor: Guilherme Kunigami
Complexidade:
Tempo de implementacao: 7 min
Testes:
- TCCC 2006, Round 1B Medium
- TCO  2006, Round 1  Easy
- SRM 320, Division 1 Hard
Dependencias: Nenhuma
```

Descricao:

```
#include <string>
#include <stdlib.h>
using namespace std;

void print_bit(int a, int k){
  for (int i=0, j = (1 << (k-1)); i<k; i++){
    printf("%d", (j&a)?1:0);
    a <<= 1;
  }
  printf("\n");
}
int bitmask(string s){
    return strtol(s.c_str(), NULL, 2);
}

/* ----------------------------------------------------------
 * Operacoes entre as mascaras de bits a e b, de tamanho k
 * ----------------------------------------------------------
 */
void operacoes_bits(int a, int b, int k){

  print_bit(a, k);
  print_bit(b, k);

  int ALL_BITS = (1 << k) - 1;
  /* bit = {0,...,k-1} */
  int bit = 6;


  print_bit(ALL_BITS, k);

  /* Uniao */
  print_bit(a | b, k);
  /* Intersecao */   a & b;
  print_bit(a & b, k);
  /* Subtracao */    a & ~b;
  print_bit(a & ~b, k);
  /* Negacao */
  print_bit(ALL_BITS ^ a, 32);
  /* Limpar o bit setado menos significativo */
  print_bit(a & (a - 1), k);
  /* Seta o 4º bit-esimo menos significativo */
  print_bit(a |= 1 << bit, k);
  /* Limpar o bit-esimo menos significativo */
  print_bit(a &= ~(1 << bit), k);
  /* Testar o bit-esimo menos significativo */
  if (a & 1 << bit)
    printf("Bit setado\n");
  else
    printf("Bit nao setado\n");
}

/* ----------------------------------------------------------
 * Itera sobre todos os subconjuntos do conjunto
 * representado pela mascara mask.
 * PS: Gera os subconjuntos maiores (em valor de mascara)
 * primeiro
 * ----------------------------------------------------------
 */
void subset(int mask, int k){
  for (int i = mask; i > 0; i = (i - 1) & mask)
    print_bit(i, k);
  print_bit(0, k);
}
/* ----------------------------------------------------------
 * Gera todas as mascaras de tamanho n, com m bits setados
 * ----------------------------------------------------------
 */
void comb(int dep, int from, int mask, int n, int m) {

  if (dep == m){
    print_bit(mask, n);
    return;
  }
  // Seta o i-esimo bit e desce
  for (int i = from; i < n; i++)
    comb(dep+1, i+1, mask | (1<<i), n, m);
}

/* ----------------------------------------------------------
 * Funcoes de bits do gcc
 * ----------------------------------------------------------
 */
void builtin(int mask){
  printf("# Bits setados: %d\n", __builtin_popcount(mask));
```

```
  //PS.: Depende do tipo do numero!
  printf("# Zeros no comeco: %d\n", __builtin_clz(mask));
  printf("# Zeros no final: %d\n", __builtin_ctz(mask));
}

/* ----------------------------------------------------------
 * Exemplo Simples de uso
 * ----------------------------------------------------------
 */
int main (){

  printf("Principais operacoes de bits\n");
  operacoes_bits(389, 454, 10);

  printf("Gera todos os subconjuntos de 1000110101:\n");
  subset(bitmask("1000110101"), 10);

  printf("Todas as combinacoes 10 escolhe 3:\n");
  comb(0, 0, 0, 10, 3);

  printf("Teste das funcoes builtin do GCC");
  builtin(bitmask("011101011000"));


  return 0;
}
```

6.4 Calculador de Expressoes

```
Autor: Alexandre Kunieda
Complexidade: O(n)
Tempo de implementacao: 4 minutos
Testes:
- SPOJbr Calculadora (n<=?) t=0.00s
Dependencias: Nenhuma
Descricao:
Calcula expressoes que envolvam parenteses e operacoes
binarias de +, -, *, /, alem de operacao unaria de -
e variaveis de nome longo


#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <map>
#include <string>
#include <algorithm>

using namespace std;

/* tamanho maximo que as variaveis/numeros assumem */
#define MAXvar 10

char expr[1000];

map<string,double> val;
char seq[MAXvar];
char *l;

double solve();
double read() {
  int n;

  while(isspace(*l)) l++;
  if(*l=='-'){
    l++;
    return -read();
  }
  else if(*l=='('){
    l++;
    return solve();
  }
  else{
    sscanf(l," %[a-zA-Z0-9]%n",seq,&n);
    l+=n;
    if( isalpha(seq[0]) ) return val[ string(seq) ];
    else return atof(seq);
  }
}
double solve(){
  char op;
  double acc=0,ult,x;

  ult=read();

  while(*l!='\0'){
    while(isspace(*l)) l++;
```

```
    op = *(l++);
    if(op==')' || op=='\0') break;

    x = read();

    if (op=='+') { acc+=ult; ult=x; }
    if (op=='-') { acc+=ult; ult=-x; }
    if (op=='*') ult*=x;
    if (op=='/') ult/=x;
  }

  return acc+ult;
}

/**** Exemplo de uso ****/
int main(){
  int i;

  while(gets(expr)!=NULL){
    /* busca por um caracter '=' na string expr[] */
    for(i=0 ; expr[i]!='\0' && expr[i]!='=' ; i++);

    /* caso tenha achado caracter '=', atribui valor a variavel */
    if(expr[i]=='='){
      l=expr+i+1; expr[i]='\0';
      val[ string(expr) ]=solve();
    }
    /* senao, apenas calcula a expressao */
    else{
      l = expr;
      printf("%.2lf\n", solve());
    }
  }

  return 0;
}
```

6.5 BIT Hasheada

```
Autor: Marcelo Galvao Povoa
Complexidade: O(n*lg n) preprocessamento
O(lg n) por query
Tempo de implementacao: ?
Testes: SPOJ.Matrix
Descricao: Implementa uma BIT que usa memória proporcional a
cardinalidade do universo s de números através de um hash 1..qid.
Permite querys em numeros fora do universo
OBS: necessario ter vetor s preenchido (pode haver repeticoes)
antes de operar a BIT

#include <cstdio>
#include <cstring>
#include <algorithm>
#define MAXN 1000
using namespace std;

int s[MAXN],n;
int qid, id[MAXN+1], bit[MAXN+1];

void init() {
  sort(s,s+n);

  qid=1; //qtde de elementos distintos
  id[1]=s[0];
  for (int i=1; i<n; i++)
    if (s[i]!=s[i-1]) id[++qid]=s[i];

  memset(bit,0,sizeof(bit));
}

int getid(int x) {
  int es=1, di=qid, me;

  if (x<id[1]) return 0;

  while (es<di) {
    me=(es+di+1)/2;

    if (id[me]==x) return me;
    if (id[me]<x) es=me;
    else di=me-1;
  }
  return es;
}

int query(int x) {
  int sum=0;
```

```
  x=getid(x);
  while (x>0) {
    sum+=bit[x];
    x-=x & (-x);
  }
  return sum;
}

/*v representa um inc/decremento em x!*/
void update(int x, int v) {
  x=getid(x);
  if (x==0) return;

  while (x<=qid) {
    bit[x]+=v;
    x+=x & (-x);
  }
}

int main() {
  s[0]=7; s[1]=3; s[2]=199; s[3]=23;
  n=4;

  init();
  update(3,4);
  update(23,-2);
  printf("%d %d %d\n",query(1), query(7), query(23));
  return 0;
}
```

6.6 Range Minimum Query (RMQ)

```
Autor: NU 2/Marcelo Galvão Póvoa
Complexidade: O(n)-preprocessamento e O(1)-consulta
Tempo de implementacao: 7 min
Testes: Testes proprios aleatorios
Dependencias: Nenhuma
Descricao: Apos o preprocessamento de um vetor v, o algoritmo
responde de forma eficiente o indice do elemento
minimo em v[a..b]. Em caso de empate, nota-se que eh
devolvido o maior indice. Caso queira o menor, desco-
mente o = em pairmin()
#include <vector>
#include <algorithm>

using namespace std;

vector<int> a;
vector< vector<int> > p;
vector<int> Log2;

int pairmin(int i1, int i2) {
   return a[i1]</*=*/a[i2] ? i1:i2;
}

void init(vector <int> &_a) {
   a=_a;
   int n=a.size();
   Log2.resize(n+1);
   for (int i=1,k=0; i<=n; i++) {
      Log2[i]=k;
      if (1 << (k+1) == i) k++;
   }

   int ln=Log2[n];
   p.assign(ln+1,vector<int>(n));
   for (int i=0; i<n; i++) p[0][i]=i;
   for (int i=1; i<=ln; i++)
      for (int j=0; j+(1 << i) -1 < n; j++) {
         int i1=p[i-1][j];
         int i2=p[i-1][j+ (1 << i-1)];
         p[i][j] = pairmin(i1, i2);
      }
}

int query(int b, int e) {
   int ln = Log2[e-b+1];
   int i1 = p[ln][b];
   int i2 = p[ln][e - (1 << ln)+1];

   return pairmin(i1,i2);
}

/**** Exemplo simples de uso ****/
int main(){
   int i,n,x;
   vector<int> vet;
```

```
   vet.clear();
   scanf(" %d",&n);
   for (i=0;i<n;i++) {
      scanf(" %d",&x);
      vet.push_back(x);
   }

   init(vet);
   printf("%d\n",query(0,n-1));
   /*imprime o maior indice do menor elemento*/

   return 0;
}
```

6.7 Range Minimum Query 2D

```
Autor: André Linhares
Complexidade:
inicialização: O(nm)
atualização e consulta: log(nm)

Teste: UVA 11297

Descrição: encontra a posição do menor
elemento de uma submatriz. A função best
pode ser adaptada para Range Maximum
Query ou para definir um critério de desempate.
```

```
#include <iostream>
#include <vector>
#include <utility>
#include <cstdlib>
#include <algorithm>

#define pii pair<int,int>

using namespace std;

#define for_to(i,j,k) for(i=j; i<=k; ++i)

#define xm (x1+x2)/2
#define ym (y1+y2)/2

#define r0 x1,xm,y1,ym
#define r1 xm+1,x2,y1,ym
#define r2 x1,xm,ym+1,y2
#define r3 xm+1,x2,ym+1,y2

#define GO go(0,r0); go(1,r1); go(2,r2); go(3,r3);

template <class T>
struct RMQ
{
  public:
  void init(vector<vector<T> > w)
  {
    R=(int)w.size()-1, C=(int)w[0].size()-1;
    v=w;
    son.assign(1,vector<int> (4,0));
    ans.resize(1,pii(0,0));
    init(0,0,R,0,C);
  }

  void update(T t,int i,int j=0)
  {
    x=i, y=j;
    v[x][y]=t;
    update(0,0,R,0,C);
  }

  pii query(int a,int b,int c=0,int d=0)
  {
    xi=a, xf=b, yi=c, yf=d;
    ret=pii(xi,yi);
    query(0,0,R,0,C);
    return ret;
  }


  int R,C,x,y,xi,xf,yi,yf;
  vector<vector<int> > son;
  vector<vector<T> > v;
  vector<pii> ans;
  pii ret;
```

```
  pii best(pii a,pii b)
  {
    if (v[a.first][a.second] < v[b.first][b.second]) return a;
    else if (v[a.first][a.second] > v[b.first][b.second])
      return b;
    return min(a,b);
  }

#define _go(i,a,b,c,d) if (a<=b && c<=d) \
{ son[node][i]=son.size(); son.push_back(vector<int> (4,0));\
ans.push_back(pii(a,c)); init(son[node][i],a,b,c,d); \
ans[node]=best(ans[node],ans[son[node][i]]); }

#define go(a,b) _go(a,b)

  void init(int node,int x1,int x2,int y1,int y2)
  { if (x1!=x2 || y1!=y2) GO; }

#undef _go
#define _go(i,a,b,c,d) if (a<=x && x<=b && c<=y && y<=d) \
update(son[node][i],a,b,c,d);  if (son[node][i]) \
ans[node]=best(ans[node],ans[son[node][i]]);

  void update(int node,int x1,int x2,int y1,int y2)
  { if (x1!=x2 || y1!=y2) GO; }

#undef _go
#define _go(i,a,b,c,d) if (son[node][i] && !(a>xf || b<xi \
|| c>yf || d<yi)) { query(son[node][i],a,b,c,d); }

  void query(int node,int x1,int x2,int y1,int y2)
  {
    if (x1==x2 && y1==y2 || (xi<=x1 && x2<=xf && yi<=y1
      && y2<=yf))
    {
      ret=best(ret,ans[node]);
      return;
    }
    GO;
  }
};

RMQ<int> T1,T2;
vector<vector<int> > v1,v2;
int i,j,k,n;
int m,t,x,y,a,x1,x2,y1,y2;
char c;
int a1,a2,lixo;
pii P;

int main()
{
  scanf("%d %d",&n,&lixo);
  v1=v2=vector<vector<int> > (n,vector<int>(n));
  for_to(i,0,n-1)
    for_to(j,0,n-1)
    {
      scanf("%d",&v1[i][j]);
      v2[i][j]=-v1[i][j];
    }
  T1.init(v1);
  T2.init(v2);
  scanf("%d",&m);
  for_to(t,1,m)
  {
    scanf(" %c",&c);
    if (c=='c')
    {
      scanf("%d %d %d",&x,&y,&a);
      --x, --y;
      T1.update(a,x,y);
      T2.update(-a,x,y);
    }
    else
    {
      scanf("%d %d %d %d",&x1,&y1,&x2,&y2);
      --x1, --x2, --y1, --y2;
      P=T1.query(x1,x2,y1,y2);
      a1=T1.v[P.first][P.second];
      P=T2.query(x1,x2,y1,y2);
      a2=-T2.v[P.first][P.second];
      printf("%d %d\n",a2,a1);
    }
  }
  return 0;
}
```