Resumen de algoritmos para competencias de programación

Kruskal

7 de abril de 2011

Índice

1. Plantillas

1.1. C++

```
#include <algorithm>
#include <iostream>
#include <iterator>
#include <sstream>
#include <fstream>
#include <cassert>
#include <climits>
#include <cstdlib>
#include <cstring>
#include <string>
#include <cstdio>
#include <vector>
#include <cmath>
```

```
#include <queue>
#include <deque>
#include <stack>
#include <list>
#include <map>
#include <set>
#include <iomanip>

using namespace std;
typedef long long ll;
template <class T> string toStr(const T &x){
stringstream s; s << x; return s.str();
}

template <class T> int toInt(const T &x){
stringstream s; s << x; int r; s >> r; return r;
}

#define For(i, a, b) for (int i=(a); i<(b); ++i)
#define D(x) cout << #x " es " << (x) << endl

const double EPS = 1e-9;
int cmp(double x, double y, double tol = EPS){
  return (x <= y + tol) ? (x + tol < y) ? -1 : 0 : 1;
}

//cuando se va a trabajar con enteros
//la suma es con 5 y no con 0.5
double round(double x){ return floor(x*100+0.5)/100; }

bool myfunction (int i,int j) { return (i>j); }

long long reverse_num( ll sourcenum ){
  ll temp = sourcenum;
  ll sum = 0;
  while ( temp ){
    sum *= 10;
    sum += temp%10;
    temp /= 10;
  }
  return sum;
}

int main()
{

  //------------------------------------|
  //para leer desde archivo             |
  freopen("filename.in","r",stdin);   //|
  //para escribir archivo               |
  freopen("filename.out","w",stdout); //|
  //------------------------------------|

  return 0;
}
```

....

1.2. Entradas especiales con C++

```
if (sscanf(s.c_str(), "%d %d", &r1, &r2) == 2)
// lee 2 numeros en caso de que sea 1 va al else
//hay que tener en cuenta q "s" es un string

if (cin.peek()!='\n') cin >> r2;
// si hay algo en esa linea que no sea
//\n lo toma y lo guarda en r2

//para leer de un buffer o algo asi
char line[200010];
gets(line);
sscanf(line, "%d", &n);

getline(cin, s);
// lee la linea y la inserta en el string s
getline(cin, s);
// tener siempre en cuenta que se usa doble getline para
// omitir el salto y solo tomar la informacion que necesitamos
```

....

1.3. Java

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.math.BigInteger;
import java.util.Scanner;


class Main{

  public static void main(String [] args){
    Scanner sc = new Scanner(System.in);
    BufferedReader r = new BufferedReader(new
          InputStreamReader (System.in));


    //Lector Scanner --> sc.nextLine();
    //Lector BufferedReader --> r.readLine();
    //--------------------------------------------|
    //Para leer o escribir archivos si se requiere  |
    System.setIn(new FileInputStream("tree.in"));   |
    System.setOut(new PrintStream("tree.out"));     |
    //--------------------------------------------|

  }
}
```

2. Matemática Aplicada

2.1. Suma de enteros, suma de cuadrados y suma de cubos

$$\sum_{i=1}^{n} i = 1 + 2 + ... + n = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + ... + n^2 = \frac{(2n+1)n(n+1)}{6}$$

$$\sum_{i=1}^{n} i^3 = 1^3 + 2^3 + ... + n^3 = \frac{n^2(n+1)^2}{4}$$

2.2. Conjunto de partes

De un conjunto S, el **conjunto de partes** $\mathrm{P}(S)$ son todos los posibles subconjuntos que se pueden formar con los elementos de S. Si el cardinal de S es de n, el cardinal de su conjunto de partes $\mathrm{P}(S)$ será de 2^n e incluye el conjunto vacío ($\emptyset$).

Field-testing:

- *UVa Live Archive* - 4794 - Sharing Chocolate
- *SPOJ* - 6073 - Chocolate

```
//tener en cuenta que el tamaño de
//valores debe ser >= que 1 << n
//y cortes es simplemente para llevar
//cuenta del numero de cortes por bitmask
for (int i = 0; i < (1<<n); ++i){
   valor[i] = cortes[i] = 0;
   for (int k = 0; k < n; ++k){
      if (i & (1 << k)){
         valor[i] += parts[k];
         cortes[i]++;
      }
   }
   suma[valor[i]].push_back(i);
   for (int g = 0; g < 105; ++g) memo[g][i] = -1;
}
```

2.3. Fibonacci $O(log\ n)$

```
int  F[2][2];
int A[2][2];
typedef  long long ll;
const int modulus = 1000000007;
int a,b,c,d;
```

```
int main()
{
        int tc;
        long long p;
        ll ans;
        scanf("%d", &tc);
        while (tc--)
        {
                F[0][0] = 1;
                F[0][1] = 1;
                F[1][0] = 1;
                F[1][1] = 0;
                A[0][0] = 1;
                A[0][1] = 0;
                A[1][0] = 0;
                A[1][1] = 1;
                scanf("%lld", &p);
                p <<= 1;
                while (p)
        {
                if (p & 1)
                {
                        a = ((ll)A[0][0] * F[0][0]) % modulus
+ ((ll)A[0][1] * F[1][0]) % modulus;
                        b = ((ll)A[0][0] * F[0][1]) % modulus
+ ((ll)A[0][1] * F[1][1]) % modulus;
                        c = ((ll)A[1][0] * F[0][0]) % modulus
+ ((ll)A[1][1] * F[1][0]) % modulus;
                        d = ((ll)A[1][0] * F[0][1]) % modulus
+ ((ll)A[1][1] * F[1][1]) % modulus;
                        A[0][0] = a % modulus;
                        A[0][1] = b % modulus;
                        A[1][0] = c % modulus;
                        A[1][1] = d % modulus;
                }
                a = ((ll)F[0][0] * F[0][0]) % modulus
 + ((ll)F[0][1] * F[1][0]) % modulus;
                        b = (ll)F[0][1] *
(F[0][0]  + F[1][1]) % modulus;
                        c = (ll)F[1][0] *
(F[0][0] + F[1][1]) % modulus;
                        d = ((ll)F[1][0] * F[0][1]) % modulus
 + ((ll)F[1][1] * F[1][1]) % modulus;
                        F[0][0] = a % modulus;
                        F[0][1] = b % modulus;
                        F[1][0] = c % modulus;
                        F[1][1] = d % modulus;
                        p >>= 1;
        }
                ans = (ll)A[0][0] * A[0][1];
                printf("%d\n", (int)(ans % modulus));

        }
        return 0;
}
```

....

2.4. Geometría

Área de un polígono. (Si no se intersecta a sí mismo, es simple):

$$A(P) = \frac{1}{2}\sum_{i=0}^{n-1}(x_i \cdot y_{i+1} - x_{i+1} \cdot y_i)$$

Centro de masa de un polígono simple con área M:

$$\bar{C}_x = \frac{\iint_R x dA}{M} = \frac{1}{6M}\sum_{i=1}^{n}(y_{i+1} \cdot y_i)(x_{i+1}^2 + x_{i+1} \cdot x_i + x_i^2)$$

$$\bar{C}_y = \frac{\iint_R y dA}{M} = \frac{1}{6M}\sum_{i=1}^{n}(x_{i+1} \cdot x_i)(y_{i+1}^2 + y_{i+1} \cdot y_i + y_i^2)$$

2.5. Áreas y volúmenes

Para $\mathbb{R}^2$ con tres puntos $P(x_1, y_1), R(x_2, y_2), Q(x_3, y_3)$ NO colineales se tiene que el área del triángulo es:

$$\acute{A}rea\ \triangle = \frac{1}{2}\left| det \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \right|$$

$$\acute{A}rea\ \triangle = \frac{(x_2y_1 - x_1y_2 + x_3y_2 - x_2y_3 - x_3y_1 + x_1y_3)}{2}$$

Teniendo tres puntos P, Q, R NO colineales en $\mathbb{R}^3$ el área del triángulo que forman es:

$$\acute{A}rea\ \triangle = \frac{\|\overrightarrow{PR} \times \overrightarrow{PQ}\|}{2}$$

Como $\acute{A}rea_{paralelogramo} = 2 \cdot \acute{A}rea\ \triangle \longrightarrow \acute{A}rea_{paralelogramo} = \|\overrightarrow{PR} \times \overrightarrow{PQ}\|$

Para encontrar el volumen de un paralelepípedo de manera vectorial, se deben conocer $\vec{u}, \vec{v}, \vec{w}$ y aplicar la fórmula: $V_p = (\vec{u} \times \vec{v}) \cdot \vec{w}$

2.6. Algoritmo de Josefo

Se trata de encontrar el lugar en el círculo inicial para que se pueda ser el último y seguir viviendo. (Basado en el problema que enfrentó Josefo).

Field-testing:

- *SPOJ* - 4557 - Musical Chairs

```
int main(){
   int n,k;
   while(cin >> n >> k && (n+k)){
       int r = 0;
       for(int i=1; i<=n;++i) r = (r+k) % i;
       cout << n << " " << k <<" " << (r+1) << endl;
   }
   return 0;
}
```

. . . .

2.7. Particiones de A condicionadas

$$\forall X\{X \in P | X \neq \emptyset\} \qquad (1)$$

$$\forall (X,Y)\{X \in P, Y \in P | X \neq Y \longrightarrow X \cap Y = \emptyset\} \qquad (2)$$

$$\{\bigcup_{i=1}^{n} X_i \mid X_i \in P\} = A \qquad (3)$$

entonces:

$$S(n,k) = S(n-1,k-1) + k \times S(n-1,k) \qquad (4)$$

. . . .

3. Combinatoria

3.1. Cuadro resumen

Fórmulas para combinaciones y permutaciones:

Tipo	*¿Repetición permitida?*	*Fórmula*
r-permutaciones	No	$\frac{n!}{(n-r)!}$
r-combinaciones	No	$\frac{n!}{r!(n-r)!}$
r-permutaciones	Sí	n^r
r-combinaciones	Sí	$\frac{(n+r-1)!}{r!(n-1)!}$

Tomado de *Matemática discreta y sus aplicaciones*, Kenneth Rosen, 5^{ta} edición, McGraw-Hill, página 315.

3.2. Criba de Eratóstenes

Field-testing:

- *SPOJ* - 2 - Prime Generator

```
typedef long long ll;

int primesGenerator( ll m, ll n ){

   vector<int> primes(100001);
```

```
    //memset( primes, false, sizeof( primes ) );

    ll cm = m;
    if( cm % 2 != 0 ) cm++;
    for( ll i = cm; i<=n; i += 2 ){
        primes[i-m] = true;
    }

    for( ll it = 3; it*it <=n; it+=2 ){
        if( m >= it*it && m%it == 0 ) primes[ 0 ] = true;
        int pMult = (m-(m%it)) + it;
        for( ll j = pMult ; j<=n; j+=it ){
            if( j >= it*it ) primes[ j-m ] = true;
        }
    }

    for( ll i = 0; i<=n-m; ++i ){
        if( i+m == 1 || i+m == 0 ) primes[ i ] = true;
        if( i+m == 2 ) primes[i] = false;
        if( !primes[i] ) cout << i+m << endl;
    }
}
```

3.3. Combinaciones, coeficientes binomiales, triángulo de Pascal

Complejidad: $O(n^2)$

$$\binom{n}{k} = \begin{cases} 1 & k = 0 \\ 1 & n = k \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{en otro caso} \end{cases}$$

```
const int N = 30;
long long choose[N+1][N+1];
/* Binomial coefficients */
  for (int i=0; i<=N; ++i) choose[i][0] = choose[i][i] = 1;

  for (int i=1; i<=N; ++i)
   for (int j=1; j<i; ++j)
    choose[i][j] = choose[i-1][j-1] + choose[i-1][j];
```

3.4. Permutaciones con elementos indistinguibles

El número de permutaciones diferentes de n objetos, donde hay n_1 objetos indistinguibles de tipo 1, n_2 objetos indistinguibles de tipo 2, ..., y n_k objetos indistinguibles de tipo k, es

$$\frac{n!}{n_1! n_2! \cdots n_k!}$$

Ejemplo: Con las letras de la palabra `PROGRAMAR` se pueden formar $\frac{9!}{2! \cdot 3!} = 30240$ permutaciones diferentes.

3.5. Desordenes, desarreglos o permutaciones completas

Un desarreglo es una permutación donde ningún elemento i está en la posición i-ésima. Por ejemplo, *4213* es un desarreglo de 4 elementos pero *3241* no lo es porque el 2 aparece en la posición 2.

Sea D_n el número de desarreglos de n elementos, entonces:

$$D_n = \begin{cases} 1 & n = 0 \\ 0 & n = 1 \\ (n-1)(D_{n-1} + D_{n-2}) & n \geq 2 \end{cases}$$

Usando el principio de inclusión-exclusión, también se puede encontrar la fórmula

$$D_n = n! \left[1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \cdots + (-1)^n \frac{1}{n!} \right] = n! \sum_{i=0}^{n} \frac{(-1)^i}{i!}$$

4. Librerías de Java

4.1. BigInteger

Field-testing:

- *SPOJ* - 24 - Small Factorials

```
static BigInteger [] fact = new BigInteger[104];
public static void init(){
  fact[0] = BigInteger.ONE;
    for (int i = 1; i<=100;++i){
      fact[i] = new BigInteger(String.valueOf(i)).multiply(fact[i-1]);
    }
}
```

5. Comparación de Strings

5.1. KMP

Field-testing:

- *SPOJ* - 32 - A Needle in the Haystack

```
string p, text;
bool tales;

void Kmp2( long *kmpNext ){
  ll i,j;
  i = j = 0;
   while (j < text.size()){
      while (i > -1 && p[i] != text[j]) i = kmpNext[i];
      i++; j++;
      if (i >= p.size()){
         printf("%d\n",j-i);
         i = kmpNext[i];
      }
   }
}

/* Calculate the table of jumps */
void preKmp( long *b ){
    ll i = 0, j = -1;
    b[ i ] = j;
    while( i<p.size() ){
        while( j>=0 && p[i] != p[j] ) j = b[j];
        i++; j++;
        b[i] = j;
    }
}

int main()
{
    ll len, *y;
    while( scanf("%lli",&len) != EOF ){
        cin >> p >> text;
        if( p.size() <= text.size() ){
            long b[ len+1 ];
            preKmp( b );
            Kmp2( b );
        }
    }
    return 0;
}
```

. . . .

5.2. Distancia de Levenshtein

Field-testing:

- *SPOJ* - 6219 - Edit distance

Mínimo número de transformaciones para convertir el String A en el String B.

```
void Levenshtein(string a, string b)
{
  D[0][0] = 0;
  int m = a.length(), n = b.length(),cost = 0;
  for (int i = 1; i<=m;++i){
    D[i][0] = i;
  }
  for (int j = 1; j<=n;++j){
    D[0][j] = j;
  }
  for (int i = 1; i <= m; ++i){
    for (int j = 1; j<= n; ++j){
        cost = (a[i-1] == b[j-1]) ? 0 : 1;
        D[i][j] = min((D[i-1][j-1]+cost),min(D[i-1][j]+1,D[i][j-1]+1));
    }
  }
  printf("%d\n",D[m][n]);
}
```

. . . .

```
int array[2100][2100], t;
string c, b;


int main(){
  cin >> t;
  For(h,0,2100) array[h][0] = array[0][h] = h;
  while( t-- ){
    cin >> c >> b;
    for ( int a = 1; a <= c.size() ; ++a ) {
        for ( int i = 1; i<= b.size(); ++i ) {
           int tales = array[a-1][i-1];
           if (c[a-1] != b[i-1]) tales++;
           array[a][i]=min(tales, min(array[a][i-1]+1,array[a-1][i]+1));
```

```
      }
    }
    cout << array[c.size()][b.size()] << endl;
  }
  return 0;
}
```

6. Grafos y árboles

6.1. Dijkstra

Field-testing:

- *SPOJ* - 15 - The Shortest Path

```
struct edge{
    int to, weight;
    edge(){}
    edge(int t,int w) :to(t),weight(w){}
    bool operator < (const edge &that) const{
       return weight > that.weight;
    }
};

const int MAXNODES = 10002;
vector <edge> g[MAXNODES];
int d[MAXNODES], p[MAXNODES];

void Dijkstra(int s,int t,int n)
{
   for (int i = 0; i<=n;++i){
     d[i] = INT_MAX;
     p[i] = -1;
   }
   d[s] = 0;
   priority_queue <edge> q;
   q.push(edge(s,0));
   while(q.size()){
       int node = q.top().to;
       int dist = q.top().weight;
       q.pop();
       if (dist > d[node]) continue;
       if (node == t) break;
       for (int i = 0; i<g[node].size();++i){
          int to = g[node][i].to;
          int w_extra = g[node][i].weight;
          if (dist + w_extra < d[to]){
            d[to] = dist + w_extra;
            p[to] = node;
            q.push(edge(to,d[to]));
          }
       }
   }
}
```

6.2. DFS

Field-testing:

- *SPOJ* - 1437 - Longest path in a tree

```
vector<int> nums[10003];
int visited[10003];
int path[10003];

int dfs( int t ){

    stack<int> p;
    p.push( t );
    int mayor = t;

    while( p.size() ){
        int h = p.top();
        visited[h] = 1;
        p.pop();

        if( path[h] > path[mayor] ) mayor = h;

        for( int i = 0; i < nums[h].size(); ++i ){
            if( !visited[nums[h][i]] ){
                p.push(nums[h][i]);
                path[nums[h][i]] = path[h]+1;
            }
        }
    }
    return mayor;
}
```

```
int main(){
    int t, u, v, last;
    while( scanf("%d",&t) != EOF ){

        int longest = t; last = -1;
        for(int i = 0; i < t+2; ++i ) nums[i].clear();

        for( int j = 1; j<t; ++j ){
            scanf("%d %d",&u,&v);
            if( u == v ) continue;
            nums[ u ].push_back( v );
            nums[ v ].push_back( u );
            last = v;
        }

        if( last == -1 ) printf("%d\n",0);
        else{
            // 2 Dfs, The first for search the longest node
            //and the second for search the longest path in the tree
            for( int i = 0; i<longest+1; ++i ) visited[i] = path[i] = 0;
            int start = dfs( 1 );
            for( int i = 0; i<longest+1; ++i ) visited[i] = path[i] = 0;
            int retry = dfs( start );
            printf("%d\n",path[retry]);
        }
    }
    return 0;
}
```

6.3. BFS

Field-testing:

- *SPOJ* - 38 - Labyrinth

```
int G[MAXN][MAXN], V[MAXN][MAXN], D[MAXN][MAXN];
int r,c;
int di[] = {-1, 1, 0, 0};
int dj[] = {0, 0, -1, 1};

int vX, vY;

void BFS(int i, int j)
{
queue <int> q;
q.push(i);
q.push(j);
int bi = i, bj = j;
  while(q.size()){
    int ci = q.front(); q.pop();
    int cj = q.front(); q.pop();
    if (D[bi][bj] < D[ci][cj]){
        bi = ci;
        bj = cj;
    }
    V[ci][cj] = 1;
    for (int k = 0; k<4;++k){
        int vi = ci + di[k], vj = cj + dj[k];
        if (G[vi][vj] > 0 && !V[vi][vj] &&
        (vi >= 0 && vi <= r) && (vj <= c && vj >= 0)){
               q.push(vi);
               q.push(vj);
               D[vi][vj] = D[ci][cj] + 1;
        }
    }
  }
  vX = bi, vY = bj;
}
```

6.4. Lowest Common Ancestor

```
LL LCA(int a, int b) {
    LL cont = 0;
    int pa = prof[a];
    int pb = prof[b];
    if (pa < pb) {
        swap(a, b);
        swap(pa, pb);
    }
    while (pa != pb) {
        cont += LL(cost[a]);
        a = p[a];
        pb++;
    }
    while (a != b) {
        cont += LL(cost[b]);
        cont += LL(cost[a]);
```

```
        a = p[a];
        b = p[b];
    }
    return cont;
}

int main() {
    int n, q, a, b;
    p[0] = cost[0] = prof[0] = 0;
    scanf("%d", &n);
    while (n) {
        for (int i=1; i<n ;i++) {
            scanf("%d %d", &a, &b);
            p[i] = a;
            cost[i] = b;
            prof[i] = prof[a]+1;
        }
        scanf("%d", &q);
        for(int i=0; i<q ;i++) {
            if (i) printf(" ");
            scanf("%d %d", &a, &b);
            printf("%lld", LCA(a, b));
        }
        printf("\n");
        scanf("%d", &n);
    }
}
```

6.5. Kruskal + Union-Find

```
struct edge{
    int start, end, weight;
    edge( ){}
    edge( int a, int b, int c ) : start(a), end(b), weight(c){}
    bool operator < ( const edge &that ) const{
        return weight < that.weight;
    }
};


vector<edge> e;
int n, m, y, j ,k;

//////////////// UNION-FIND by Andres Mejia////////////////////

int p[100003], rank[100003];
void make_set( int x ){ p[x] = x, rank[x] = 0; }
void link( int x, int y ){
    if( rank[x] > rank[y] ) p[y] = x;
    else{ p[x] = y; if( rank[x] == rank[y] ) rank[y]++; }
}
int find_set( int x ){
    return x != p[x] ? p[x] = find_set(p[x]) : p[x] ;
}
void merge( int x, int y ){ link(find_set(x), find_set(y)); }

///////////////////////////////////////////////////////////////


ll kruskal( int n ){
    ll total = 0;
    sort( e.begin(), e.end() );
    for ( int i = 0; i<=n ; ++i ) make_set(i);
    for ( int i = 0; i< e.size(); ++i ) {
        int u = e[i].start, v = e[i].end, w = e[i].weight;
        if(find_set(u) != find_set(v)){
            total += w;
            merge(u, v);
        }
    }
    return total;
}

// n is the number of nodes kruskal(n) is the mst !!
//can be neccesary e.push_back( edge(y,j,k) );
```

6.6. Minimum Spanning Tree - Prim

```
struct edge{
  double to, weight;
  edge() { }
  edge ( int t ,double w ) : to(t), weight(w){}
  bool operator < ( const edge &that ) const {
    return weight > that.weight;
  }
};
```

```
struct pos{
  double x, y;
  pos() { }
  pos( double pt ,double pw ) : x(pt), y(pw){}
};

int n, d;
double x, y, result, r, r1, r2;
double prim( vector<edge> graph[], int maxNodes );

int main(){
  while( scanf("%d",&d) != EOF ){

    vector<pos> nodes; // pnts "x" y "y" de cada nodo leido
    for( int it = 0; it<d; ++it ){
      cin >> x >> y;
      nodes.push_back( pos( x, y ) );
    }

    vector<edge> graph[ d ];
    for( int i = 0; i<d; ++i ){
      for( int a = 0; a<d; ++a ){
  if( i==a ) continue;
  r1 = pow ( (nodes[ i ].x - nodes[ a ].x ) , 2) ;
  r2 = pow ( (nodes[ i ].y - nodes[ a ].y ) , 2) ;
  r = r1 + r2;
  graph[ i ].push_back( edge( a, sqrt(r) )  );
      }
    }

    printf( "%.2f\n",  prim( graph, d ) );
    //tales( graph, d );
  }
  return 0;
}

double prim( vector<edge> graph[], int maxNodes ){

  double total = 0.0;
  priority_queue<edge> q;
  q.push( edge( 0, 0.0 ) );
  set<int> visited;

  while( q.size() ){
    int node = q.top().to;
    double weight1 = q.top().weight;
    q.pop();
    if( visited.count(node) ) continue;
    visited.insert(node);
    total += weight1;
    for( int i = 0; i<graph[node].size(); ++i ){
      if( visited.count( graph[node][i].to == 0 ) ){
  q.push( graph[node][i] );
      }
    }
  }
  return total;
}
```

....

6.7. Máximo Flujo: Ford-Fukerson

Field-testing:

- *SPOJ* - 3868 - Total Flow

```
int cap[MAXN+1][MAXN+1], prev[MAXN+1];

vector<int> g[MAXN+1]; //Vecinos de cada nodo.

void link(int u, int v, int c)
{ cap[u][v] = c; g[u].push_back(v), g[v].push_back(u); }
/*
  Crear aristas mediante link(a,b,c)
  Esta implementacion es de Andres Mejia, fue usada para
  pasar el problema
  Total Flow (MTOTALF) de Spoj
*/

int residual[MAXN+1][MAXN+1];
int fordFulkerson(int n, int s, int t){
  memcpy(residual, cap, sizeof cap);

  int ans = 0;
  while (true){
    fill(prev, prev+n, -1);
    queue<int> q;
    q.push(s);
    while (q.size() && prev[t] == -1){
      int u = q.front();
```

```
      q.pop();
      vector<int> &out = g[u];
      for (int k = 0, m = out.size(); k<m; ++k){
        int v = out[k];
        if (v != s && prev[v] == -1 && residual[u][v] > 0)
          prev[v] = u, q.push(v);
      }
    }

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

    int bottleneck = INT_MAX;
    for (int v = t, u = prev[v]; u != -1; v = u, u = prev[v]){
      bottleneck = min(bottleneck, residual[u][v]);
    }
    for (int v = t, u = prev[v]; u != -1; v = u, u = prev[v]){
      residual[u][v] -= bottleneck;
      residual[v][u] += bottleneck;
    }
    ans += bottleneck;
  }
  return ans;
}
```

6.8. Floyd - Warshall

```
void Floyd()
{
//llenar primero la matriz dp!!!!
  for (int k = 0; k<maxNodes;++k){
    for (int i = 0; i<maxNodes;++i){
      for (int j = 0; j<maxNodes;++j){
       dp[i][j] = min(dp[i][j], dp[i][k]+dp[k][j]);
      }
    }
  }
}
```

6.9. Sparse Table

Range Minimum Query (RMQ). Para encontrar la posición con el mínimo valor de un elemento entre dos índices específicos.

```
void sparseTable(int M[MAXN][LOGMAXN], int A[MAXN], int N)
  {
      int i, j;

  //initialize M for the intervals with length 1
      for (i = 0; i < N; i++)
          M[i][0] = i;
  //compute values from smaller to bigger intervals
      for (j = 1; 1 << j <= N; j++)
          for (i = 0; i + (1 << j) - 1 < N; i++)
              if (A[M[i][j - 1]] < A[M[i + (1 << (j - 1))][j - 1]])
                  M[i][j] = M[i][j - 1];
              else
                  M[i][j] = M[i + (1 << (j - 1))][j - 1];
  }
```

6.10. Segment Tree

```
int A[10]; // aqui se guarda el arbol sobre el q se va a operar
int M[10]; // aqui queda el RMQ dependiendo de la
//busqueda mm tmb queda el segment tree

void initialize( int node, int b, int e ){
     if (b == e) M[node] = b;
     else {
        initialize(2 * node, b, (b + e) / 2 );
        initialize(2 * node + 1, (b + e) / 2 + 1, e );

        if (A[M[2 * node]] <= A[M[2 * node + 1]]) M[node] = M[2 * node];
        else M[node] = M[2 * node + 1];
     }
}

int query(int node, int b, int e, int i, int j){
      int p1, p2;
      if (i > e || j < b) return -1;
      if (b >= i && e <= j) return M[node];

      p1 = query(2 * node, b, (b + e) / 2, i, j);
      p2 = query(2 * node + 1, (b + e) / 2 + 1, e, i, j);

      if (p1 == -1) return M[node] = p2;
      if (p2 == -1) return M[node] = p1;
      if (A[p1] <= A[p2]) return M[node] = p1;
```

```
    return M[node] = p2;
}

// Esta entrada es un ejemplo con nueve valores en
//la tabla de entrada y haciendo query de 5 a 7

int main(){
  memset( M, -1, sizeof(M) );
  A[0] = 2;
  A[1] = 4;
  A[2] = 3;
  A[3] = 1;
  A[4] = 6;
  A[5] = 7;
  A[6] = 8;
  A[7] = 9;
  A[8] = 1;
  A[9] = 7;
  initialize( 1, 0, 9 );
    for ( int a = 0; a < 26; ++a ) {
        cout << M[a] << " ";
    }
    cout << endl;
    cout << query( 1, 0, 9, 5 , 7 ) << endl;
  return 0;
}
```

7. Checklist para WA, TLE y RE

- Overflow.
- Requiere BigInteger.
- El programa termina anticipadamente por la condición en el ciclo de lectura. Por ejemplo, se tiene `while (cin >> n >> k && n && k)` y un caso válido de entrada es n = 1 y k = 0.
- No hay más chocolatina para partir.
- El grafo no es conexo.
- Puede haber varias aristas entre el mismo par de nodos.
- Las aristas pueden tener costos negativos.
- El grafo tiene un sólo nodo.
- La cadena puede ser vacía.
- Las líneas pueden tener espacios en blanco al principio o al final (Cuidado al usar `getline` o `fgets`).
- El arreglo no se limpia entre caso y caso.
- Se está imprimiendo una línea en blanco con un espacio (`printf(" \n")` en vez de `printf("\n")` ó `puts(" ")` en vez de `puts("")`).