# Indice

[Indice 1](#_Toc434998528)

[Estructura de Datos 2](#_Toc434998529)

[Arbol Binario Indexado 2](#_Toc434998530)

[Segment Tree 3](#_Toc434998531)

[Range Min-Max Quering 4](#_Toc434998532)

[Lowest Comon Antecesor 5](#_Toc434998533)

[Heavy Ligth Descomposition+Segmente Tree+Lowest Common Antecesor 6](#_Toc434998534)

[Centroid Descomposition+Lowest Common Antecesor 7](#_Toc434998535)

[Trie 9](#_Toc434998536)

[Grafos & Flow 10](#_Toc434998537)

[Articulations Po**int**s 10](#_Toc434998538)

[Brigdes 11](#_Toc434998539)

[Strong Connect Component 12](#_Toc434998540)

[Kruskal 13](#_Toc434998541)

[Prim 14](#_Toc434998542)

[K-th Camino Minimo 15](#_Toc434998543)

[Floyd Warshall 16](#_Toc434998544)

[Camino Circuito Eureliano 17](#_Toc434998545)

[**For**d Fulkerson 18](#_Toc434998546)

[Flujo Maximo Costo-Costo Minimo 19](#_Toc434998547)

[Hungarian Algorithm 21](#_Toc434998548)

[Geometry & Math 22](#_Toc434998549)

[Estructuras Geometricas 22](#_Toc434998550)

[Convex Hull 25](#_Toc434998551)

[Closest Pair of Po**int**s 26](#_Toc434998552)

[GCD – LCM – Extended GCD 27](#_Toc434998553)

[Area de Union+Multi Set 28](#_Toc434998554)

[Area de Union+Segment Tree 29](#_Toc434998555)

[Factorizacion 31](#_Toc434998556)

[Metodo de Gauss 33](#_Toc434998557)

[Fibonacci Logaritmico 34](#_Toc434998558)

[Binary Exponetation 35](#_Toc434998559)

[Factorial Compactado 36](#_Toc434998560)

[String Algorithms 37](#_Toc434998561)

[Palindromes 1 37](#_Toc434998562)

[Knutt Morris Pratt (Prefix Function) 38](#_Toc434998563)

[Z Function 39](#_Toc434998564)

[Cyclic Sh**if**t 39](#_Toc434998565)

[Longest Common Subsequence 40](#_Toc434998566)

[Longest Increasing or Decreasing Secuence 41](#_Toc434998567)

[Counting Change 41](#_Toc434998568)

[Edit Distance 41](#_Toc434998569)

[Espec**if**icaciones Lenguajes C++ 43](#_Toc434998570)

[Funciones Entrada C++ 43](#_Toc434998571)

[Funciones Salidas C++ 44](#_Toc434998572)

[Sincronizar Entrada/Salida ***iostream*** 45](#_Toc434998573)

# Estructura de Datos

## Arbol Binario Indexado

**# include <cstdio>**

**using namespace std;**

**int** tm, op, p;

**typedef long long** ll;

**struct** date{

**int** save[10005];

**void** update(**int** p, ll v){

**for**(**int** i = p; i <= tm; i += i& -i)

save[i] += v;

}

**ll** query(**int** p){

**int** sum=0;

**for**(**int** i = p; i > 0; i -= (i & -i))

sum += save[i];

**return** sum;

}

}bit;

**int** main(){

**scanf**("%d", &tm);

**while**(1){

**scanf**("%d %d", &op, &p);

**if**(op == -1)

**return 0;**

**if**(op)

bit.update(p);

**else**

bit.pr**int**(p);

}

}

## Segment Tree

**# include** <iostream>

**# include** <algorithm>

**# define** oo 1 << 29

**# define** RANG 30000000

**using namespace std;**

**char** c;

**int** r1, r2, r3, i, Q;

struct S\_Tree{

**int** n;

**int** elements[5005];

**int** T[RANG], Mk[RANG];

**int** Build(**int** x, **int** xend, **int** P = 1){

**if**(x == xend)

**return** T[P] = elements[x];

**int** pv = (x+xend)/2;

**return** T[P] = Build(x, pv, P\*2) +

Build(pv+1, xend, P\*2+1);

}

**void** Lazy\_propagation(**int** x, **int** xend, **int** P){

**if**(x == xend)

**return**;

**int** pv = (x+xend)/2;

T[P\*2] += (pv - x + 1)\* Mk[P];

T[P\*2+1] += (xend - pv )\* Mk[P];

Mk[P\*2] += Mk[P];

Mk[P\*2+1] += Mk[P];

Mk[P] = 0;

}

**int** Query(**int** x, **int** xend, **int** P = 1){

**if**(r2 < x || xend < r1)

**return 0;**

**if**(Mk[P])

Lazy\_propagation(x, xend, P);

**if**(r1 <= x && xend <= r2)

**return** T[P];

**int** pv = (x+xend)/2;

**return** Query(x, pv, P\*2) +

Query(pv+1, xend, P\*2+1);

}

**int** Update(**int** x, **int** xend, **int** P = 1){

**if**(Mk[P])

Lazy\_propagation(x, xend, P);

**if**(r2 < x || xend < r1)

**return** T[P];

**if**(r1 <= x && xend <= r2){

Mk[P] += r3;

T[P] += ((xend-x)+1)\*r3;

**return** T[P];

}

**int** pv = (x+xend)/2;

**return** T[P] = Update(x, pv, P\*2) +

Update(pv+1, xend, P\*2+1);

}

}St;

**int** main(){

**cin** >> St.n;

**for**(i = 1; i <= St.n; i++)

**cin** >> St.elements[i];

St.Build(1, St.n);

**cin** >> Q;

**while**(Q--){

**cin** >> c >> r1 >> r2;

**if**(c == 'Q')

**cout** << St.Query(1, St.n) << endl;

**else**{

**cin** >> r3;

St.Update(1, St.n);

}

}

**return 0;** }

## Range Min-Max Quering

**# include** <cstdio>

**# include** <cmath>

**# include** <algorithm>

**using namespace std;**

**int** mat[5005][20];

**int** n, p2, p1, q;

**void** Build\_RMQ(){

**int** cc = (**int**) log2(n);

**int** p = n, a, i, j;

**for**(i = 1; i <= cc; i++){

a = 1 << (i-1);

p -= a;

**for**(j = 1; j <= p; j++)

mat[j][i] = min(mat[j][i-1], mat[j+a][i-1]);

}

}

**void** find\_RMQ(){

**int** c = (**int**)log2(p2-p1);

**printf**("%d\n", min(mat[p1][c], mat[p2-(1<<c)+1][c]));

}

**int** main(){

**scanf**("%d %d", &n, &q);

**for**(**int** i = 1; i <= n; i++)

**scanf**("%d", &mat[i][0]);

Build\_RMQ();

**while**(q--){

**scanf**("%d %d", &p1, &p2);

find\_RMQ();

}

**return 0;**

}

## Lowest Comon Antecesor

**# include <bits/stdc++.h>**

**# define RANG 1000005**

**using namespace std;**

**int i, cn, q, x, y;**

**vector <int> v[RANG];**

**struct LCA {**

**int T[100005][20], L[100005];**

**void DFS(int np, int prev){**

**L[np] = L[prev]+1;**

**int l = v[np].size();**

**for(int i = 0; i < l; i++){**

**int nh = v[np][i];**

**if(nh != prev)**

**DFS(nh, np);**

**}**

**}**

**void BFS(int np){**

**queue <int> Q;**

**Q.push(np);**

**L[np] = 1;**

**int l, nh;**

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

**np = Q.front();**

**Q.pop();**

**l = v[np].size();**

**for(int i = 0; i < l; i++){**

**nh = v[np][i];**

**if(L[nh] == 0){**

**L[nh] = L[np]+1;**

**Q.push(nh);**

**}**

**}**

**}**

**}**

**void Build(int n){**

**BFS(1);**

**int lg = log2(n);**

**for(int j = 1; j <= lg; j++)**

**for(int i = 1; i <= n; i++)**

**if(T[i][j-1] != -1)**

**T[i][j] = T[T[i][j-1]][j-1];**

**}**

**int Query(int x, int y){**

**int sol = 0;**

**if(L[x] < L[y])swap(x, y);**

**int lg = (int)log2(L[x]);**

**for(int i = lg; i >= 0; i--)**

**if(L[x] - (1 << i) >= L[y] && T[x][i])**

**x = T[x][i], sol += (1 << i);**

**if(x == y)return sol;**

**for(int i = lg; i >= 0; i--)**

**if(T[x][i] != T[y][i] && T[x][i])**

**x = T[x][i], y = T[y][i], sol += (1 << i);**

**return sol+2;**

**return T[x][0];**

**}**

**}Lc;**

**int main(){**

**scanf("%d", &cn);**

**for(i = 2; i <= cn; i++){//Leyendo padre**

**scanf("%d", &Lc.T[i][0]);**

**v[Lc.T[i][0]].push\_back(i);**

**}**

**Lc.Build(cn);**

**scanf("%d", &q);**

**while(q--){**

**scanf("%d %d", &x, &y);**

**printf("%d\n", Lc.Query(x, y));**

**}**

## Heavy Ligth Descomposition+Segmente Tree+Lowest Common Antecesor

**# include** <bits/stdc++.h>

**using namespace std;**

typedef pair<**int**, **int**> par;

vector <par> v[10005];

vector <**int**> indx[10005];

**int** subsize[10005], chainHead[10005], chainIndx[10005], posInBase[10005], otherEnd[10005], chainNo, cont;

*St -> estructura segment tree. Build-Query-Update+Lazy Propagation*

*LC -> Lowest Common Antecesor. Level[n], T[n][log n]. Build-Query*

//Inicializar Level y subsize

**void** DFS(**int** np, **int** prev, **int** depth = 0){

Lc.Level[np] = depth;

Lc.T[np][0] = prev;

subsize[np] = 1;

**int** l = v[np].size();

**for**(**int** i = 0; i < l; i++){

**int** nh = v[np][i].first;

**if**(nh != prev){

otherEnd[indx[np][i]] = nh;

DFS(nh, np, depth+1);

subsize[np] += subsize[nh];

}

}

}

//Descomposition Hevy Ligth

**void** HDL(**int** np, **int** nc, **int** prev){

**if**(chainHead[chainNo] == -1)

chainHead[chainNo] = np;

chainIndx[np] = chainNo;

posInBase[np] = cont;

Posicion que sera usada en el Segment Tree

St.elements[cont++] = nc;

**int** nh = -1, newc, l = v[np].size();

**for**(**int** i = 0; i < l; i++){

**if**(v[np][i].first == prev)continue;

**if**(nh == -1 || subsize[nh] < subsize[v[np][i].first]){

nh = v[np][i].first;

newc = v[np][i].second;

}

}

**if**(nh != -1)

HDL(nh, newc, np);

**for**(**int** i = 0; i < l; i++)

**if**(nh != v[np][i].first && v[np][i].first != prev){

chainNo++;

HDL(v[np][i].first, v[np][i].second, np);

}

}

**int** query\_up(**int** u, **int** v){

**int** uchain = chainIndx[u], vchain = chainIndx[v], ans = -1;

**while**(uchain != vchain){

ans = max(ans, St.query(0, cont-1, 1, posInBase[chainHead[uchain]], posInBase[u]));

u = Lc.T[chainHead[uchain]][0];

uchain = chainIndx[u];

}

ans = max(ans, St.query(0, cont-1, 1, posInBase[v]+1, posInBase[u]));

**return** ans;

}

**int** query(**int** x, **int** y){

**int** lca = Lc.Query(x, y);

**return** max(query\_up(x, lca), query\_up(y, lca));

}

**void** update(**int** i, **int** val){

**int** x = otherEnd[i];

x = posInBase[x];

St.elements[x] = val;

St.update(0, cont-1, 1, x);

}

**int** n, i, a, b, c, tc;

**char** arr[50];

**int** main(){

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

cont = 0;

**for**(i = 1; i < n; i++){

**scanf**("%d %d %d", &a, &b, &c);

v[a].push\_back((par){b, c});

v[b].push\_back((par){a, c});

indx[a].push\_back(i);

indx[b].push\_back(i);

}

fill(chainHead, chainHead+10002, -1);

chainNo = 0;

DFS(1, -1);

HDL(1, -1, -1);

St.Build(0, cont-1);

Lc.Build(n);

## 

## Centroid Descomposition+Lowest Common Antecesor

**# include** <bits/stdc++.h>

**using namespace std;**

const **int** oo = 1 << 30;

**int** subsize[100005], Ant[100005], sol[100005], ref\_pos, n, x, y;

vector <**int**> v[100005];

**bool** mk[100005];

**void** DFS1(**int** np, **int** prev){

subsize[np] = 1;

**int** l = v[np].size();

**for**(**int** i = 0; i < l; i++){

**int** nh = v[np][i];

**if**(nh != prev && !mk[nh]){

DFS1(nh, np);

subsize[np] += subsize[nh];

}

}

}

**int** DFS2(**int** np, **int** prev){

**int** l = v[np].size();

**for**(**int** i = 0; i < l; i++){

**int** nh = v[np][i];

**if**(nh != prev && !mk[nh] && subsize[nh] > subsize[ref\_pos]/2)

**return** DFS2(nh, np);

}

**return** np;

}

**void** Descomposition(**int** root, **int** prev){

ref\_pos = root;

DFS1(root, root);

**int** centroid = DFS2(root, root);

Ant[centroid] = prev;

mk[centroid] = true;

**int** l = v[centroid].size();

**for**(**int** i = 0; i < l; i++){

**int** nh = v[centroid][i];

**if**(!mk[nh])

Descomposition(nh, centroid);

}

}

// LC -> tipo LCA, buscar implementacion arriba.

**void** Update(**int** x){

**int** y = x;

**while**(y > 0){

sol[y] = min(sol[y], Lc.Query(x, y));

y = Ant[y];

}

}

**int** Query(**int** x){

**int** y = x, ans = oo;

**while**(y > 0){

ans = min(ans, Lc.Query(y, x)+sol[y]);

y = Ant[y];

}

**return** ans;

}

**int** Q;

**int** main(){

**scanf**("%d %d", &n, &Q);

**for**(**int** i = 1; i < n; i++){

**scanf**("%d %d", &x, &y);

v[x].push\_back(y);

v[y].push\_back(x);

}

fill(sol, sol+n+1, oo);

Lc.Build(n);

Descomposition(1, -1);

Update(1);

**while**(Q--){

**scanf**("%d %d", &x, &y);

**if**(x == 1)

Update(y);

else

**printf**("%d\n", Query(y));

}

**return 0;**

}

## Trie

**# include** <cstdio>

**# include** <cstring>

**using namespace std;**

**int** n, q, i, j, ls, sol;

**char** s[505];

struct Trie{

Trie \*son[255];

**int** end;

}T, \*p = &T;

**int** main(){

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

**for**(i = 1; i <= n; i++){

**scanf**("%s", &s);

ls = strlen(s);

p = &T;

**for**(j = 0; j < ls; j++){

**if**(p -> son[s[j]] == NULL)

p -> son[s[j]] = new Trie();

p = p -> son[s[j]];

}

}

**scanf**("%d", &q);

**for**(i = 1; i <= q; i++){

**scanf**("%s", &s);

ls = strlen(s);

p = &T;

**for**(j = 0; j < ls; j++){

**if**(p -> son[s[j]] == NULL)

break;

p = p -> son[s[j]];

**if**(j == ls-1)

sol++;

}

}

**printf**("%d", sol);

**return 0;**

}

# Grafos & Flow

## Articulations Po**int**s

**# include** <cstdio>

**# include** <vector>

**# include** <algorithm>

**using namespace std;**

vector <**int**> v[505];

**int** low[505], D[505], x, y, cn, cc, l;

**bool** mk[505];

**void** Apoint(**int** node){

low[node] = D[node] = ++l;

**int** ls = v[node].size();

**for**(**int** i = 0; i < ls; i++){

**int** next = v[node][i];

**if**(!low[next]){

Apo**int**(next);

low[node] = min(low[node], low[next]);

**if**( (D[node] == 1 && D[next] > 2) ||

(low[next] >= D[node] && D[node] != 1) )

mk[node] = true;

}

else

low[node] = min(low[node], D[next]);

}

}

**int** main(){

**scanf**("%d %d", &cn, &cc);

**for**(**int** i = 1; i <= cc; i++){

**scanf**("%d %d", &x, &y);

v[x].push\_back(y);

v[y].push\_back(x);

}

Apoint(1);

**for**(**int** i = 1; i <= cn; i++)

**if**(mk[i])

**printf**("%d\n", i);

}

## Brigdes

**# include** <vector>

**# include** <cstdio>

**# define** RANG 5005

**using namespace std;**

struct par {

**int** np, nh;

**bool** mk;

**int** next(**int** x){

**if**(x == np)

**return** nh;

**return** np;

}

}A[RANG];

**int** cc, i, L, x, y;

**int** Low[RANG], T[RANG];

vector <**int**> v[RANG];

**void** Brigdes(**int** np){

T[np] = Low[np] = ++L;

**int** l = v[np].size();

**for**(**int** i = 0; i < l; i++){

**int** nh = A[ v[np][i] ].next(np);

**if**(!T[nh]){

A[ v[np][i] ].mk = true;

Brigdes(nh);

Low[np] = min(Low[nh], Low[np]);

**if**(Low[nh] > T[np])

**printf**("%d %d\n", np, nh);

}

else

**if**(!A[v[np][i]].mk)

Low[np] = min(Low[np], T[nh]);

}

}

**int** main(){

**scanf**("%d", &cc);

**for**(i = 1; i <= cc; i++){

**scanf**("%d %d", &x, &y);

A[i] = (par){x, y};

v[x].push\_back(i);

v[y].push\_back(i);

}

Brigdes(1);

**return 0;**

}

## Strong Connect Component

**# include** <stack>

**# include** <vector>

**# include** <cstdio>

**# include** <algorithm>

**using namespace std;**

**int** T[5005], low[5005], L;

**int** x, y, cn, cc;

vector <**int**> v[5005];

stack <**int**> S;

**bool** mk[5005];

**void** SCC(**int** np){

T[np] = low[np] = ++L;

**int** l = v[np].size();

S.push(np);

**for**(**int** i = 0; i < l; i++){

**int** nh = v[np][i];

**if**(!T[nh]){

SCC(nh);

low[np] = min(low[nh], low[np]);

}

else

**if**(!mk[nh])

low[np] = min(T[nh], low[np]);

}

**if**(low[np] == T[np]){

**while**(S.top() != np){

**printf**("%d ", S.top());

mk[S.top()] = true;

S.pop();

}

**printf**("%d\n", S.top());

mk[S.top()] = true;

S.pop();

}

}

**int** main(){

**scanf**("%d %d", &cn, &cc);

**for**(**int** i = 1; i <= cc; i++){

**scanf**("%d %d", &x, &y);

v[x].push\_back(y);

}

**for**(**int** i = 1; i <= cn; i++)

**if**(!mk[i])

SCC(i);

**return 0;**

}

## Kruskal

**# include** <queue>

**# include** <cstdio>

**using namespace std;**

**int** R[5005], Set[5005];

**int** i, x, y, z, n1, n2, sol, cn, cc;

struct par{

**int** x, y, z;

**bool** operator < (const par &a)

const {

**return** z > a.z;

}

};

priority\_queue <par> Q;

**void** make\_set(){

**for**(**int** i = 1; i <= cn; i++)

R[i] = 1, Set[i] = i;

}

**int** find\_set(**int** x){

**if**(x != Set[x])

**return** Set[x] = find\_set(Set[x]);

**return** x;

}

**void** join\_set(){

**if**(R[n1] > R[n2])

Set[n2] = n1, R[n1] += R[n2];

else

Set[n1] = n2, R[n2] += R[n1];

}

**int** main(){

freopen("kruskal.in", "r", stdin);

freopen("kruskal.out", "w", stdout);

**scanf**("%d %d", &cn, &cc);

**for**(i = 1; i <= cc; i++){

**scanf**("%d %d %d", &x, &y, &z);

Q.push((par){x, y, z});

}

make\_set();

**for**(; !Q.empty(); Q.pop()){

n1 = find\_set(Q.top().x);

n2 = find\_set(Q.top().y);

**if**(n1 != n2)

sol += Q.top().z,

join\_set();

}

**printf**("%d", sol);

**return 0;**

}

## Prim

**# include** <queue>

**# include** <vector>

**# include** <cstdio>

**using namespace std;**

struct par {

**int** n1, n2;

**bool** operator < (const par &a)

const {

**return** n2 > a.n2;

}

};

**bool** mk[5005];

**int** np, nh, nc, ch, i, l, x, y, z, sol, cn, cc;

vector <par> v[5005];

priority\_queue <par> Q;

**int** main(){

**scanf**("%d %d", &cn, &cc);

**for**(i = 1; i <= cc; i++){

**scanf**("%d %d %d", &x, &y, &z);

v[x].push\_back((par){y, z});

v[y].push\_back((par){x, z});

}

**for**(Q.push((par){1, 0});

!Q.empty();

Q.pop()){

np = Q.top().n1;

nc = Q.top().n2;

l = v[np].size();

**if**(mk[np])continue;

mk[np] = true;

sol += nc;

**for**(i = 0; i < l; i++){

nh = v[np][i].n1;

ch = v[np][i].n2;

**if**(!mk[nh])

Q.push((par){nh, ch});

}

}

**printf**("%d", sol);

**return 0;**

}

## K-th Camino Minimo

**# include** <queue>

**# include** <vector>

**# include** <cstdio>

**# define** RANG 5005

**using namespace std;**

struct par {

**int** x, y;

**bool** operator > (const par &a)

const {

**return** y > a.y;

}

};

vector <par> v[RANG];

priority\_queue <par, vector<par>, greater<par> > Q;

**int** End, cc, i, x, y, z, np, nh, nc, hc, l, k;

**int** V[RANG];

**int** k\_th(){

**for**(Q.push((par){1, 0}); !Q.empty(); ){

np = Q.top().x;

nc = Q.top().y;

Q.pop();

l = v[np].size();

V[np]++;

**if**(np == End){

**if**(V[np] == k)**return** nc;

}

**for**(i = 0 ; i < l; i++){

nh = v[np][i].x;

hc = v[np][i].y;

**if**(V[nh] < k)

Q.push((par){nh, nc+hc});

}

}

}

**int** main(){

**scanf**("%d %d %d", &cc, &End, &k);

**for**(i = 1; i <= cc; i++){

**scanf**("%d %d %d", &x, &y, &z);

v[x].push\_back((par){y, z});

v[y].push\_back((par){x, z});

}

**printf**("%d", k\_th());

**return 0;**

}

## Floyd Warshall

**# include** <cstdio>

**using namespace std;**

**int** cn, cc, i, j, k, x, y, z;

**int** map[305][305];

**int** main(){

**scanf**("%d %d", &cn, &cc);

**for**(i = 1; i <= cc; i++){

**scanf**("%d %d %d", &x, &y, &z);

**if**(map[x][y] == 0 || map[x][y] > z)

map[x][y] = z;

**if**(map[y][x] == 0 || map[y][x] > z)

map[y][x] = z;

}

**for**(k = 1; k <= cn; k++)

**for**(i = 1; i <= cn; i++)

**if**(map[i][k] > 0){

**for**(j = 1; j <= cn; j++){

**if**(map[k][j] == 0)

continue;

**if**(map[i][j] == 0 ||

map[i][j] > map[i][k]+map[k][j])

map[i][j] = map[i][k]+map[k][j];

}

}

**for**(i = 1; i <= cn; i++){

**for**(j = 1; j <= cn; j++)

**if**(map[i][j] == 0)

**printf**("? ");

else

**printf**("%d ", map[i][j]);

**printf**("\n");

}

}

## Camino Circuito Eureliano

**# include** <queue>

**# include** <vector>

**# include** <cstdio>

**using namespace std;**

struct tri{

**int** np, nh;

**bool** mk;

**int** next(**int** x){

**if**(x == np)

**return** nh;

**return** np;

}

}A[5005];

**int** ini = 1, i, j, x, y, c, cn, cc, C[5005];

vector <**int**> v[5005];

queue <**int**> Q;

**void** Euler(**int** np){

**int** ls = v[np].size();

**for**(**int** i = 0; i < ls; i++){

**int** p = v[np][i];

**if**(!A[p].mk){

A[p].mk = true;

Euler(A[p].next(np));

}

}

Q.push(np);

}

**int** main(){

**scanf**("%d %d", &cn, &cc);

**for**(i = 1; i <= cc; i++){

**scanf**("%d %d", &x, &y);

A[i] = (tri){x, y, false};

v[x].push\_back(i);

v[y].push\_back(i);

C[x]++;

C[y]++;

}

**for**(i = 1; i <= cn; i++)

**if**(C[i] % 2 == 1)

c++,

ini = i;

**if**(c > 2){

**printf**("No es camino, ni circuito");

**return 0;**

}

**if**(c == 2)

**printf**("Es camino\n");

**if**(c == 0)

**printf**("Es circuito\n");

Euler(ini);

**for**(;!Q.empty(); Q.pop())

**printf**("%d\n", Q.front());

}

## Ford Fulkerson

**# include** <queue>

**# include** <cstdio>

**# include** <vector>

**# include** <algorithm>

**# define** oo 1 << 29

**using namespace std;**

**int** sr, sk, n, m, x, y, z, np, nh, cp, p, l, i, max\_flow, b;

**int** Flow[105][105], Fr[105];

**bool** mk[105];

vector <**int**> v[105];

**int** aug\_path(){

priority\_queue <pair<**int**, **int**> > Q;

fill(Fr, Fr+n+1, -1);

fill(mk, mk+n+1, false);

mk[sr] = true;

Q.push(make\_pair(oo, sr));

b = 0;

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

cp = Q.top().first;

np = Q.top().second;

Q.pop();

**if**(np == sk){

b = max(b, cp);

break;

}

l = v[np].size();

**for**(i = 0; i < l; i++){

nh = v[np][i];

**if**(!mk[nh] && Flow[np][nh]){

mk[nh] = true;

Fr[nh] = np;

Q.push(make\_pair(min(cp, Flow[np][nh]), nh));

}

}

}

nh = sk;

**while**(Fr[nh] != -1){

np = Fr[nh];

Flow[np][nh] -= b;

Flow[nh][np] += b;

v[nh].push\_back(np);

nh = np;

}

**return** b;

}

**int** main(){

**scanf**("%d %d %d %d", &n, &m, &sr, &sk);

**for**(i = 1; i <= m; i++){

**scanf**("%d %d %d", &x, &y, &z);

v[x].push\_back(y);

Flow[x][y] = z;

}

//**while**(p = aug\_path()) max\_flow += p;

max\_flow = aug\_path();

**printf**("%d\n", max\_flow);

**return 0;**

}

## Flujo Maximo Costo-Costo Minimo

**# include** <bits/stdc++.h>

typedef **long long** ll;

**using namespace std;**

**int** n;

struct nod{

ll x,y,h;

**int** id;

}N[505];

vector<**int**> v[505];

ll dist(nod a,nod b){

**if**(a.id == 0 || b.id == 0 || a.id == n+1 || b.id == n+1)**return 0;**

**return** (b.x - a.x)\*(b.x - a.x) + (b.y - a.y)\*(b.y - a.y) + (b.h - a.h)\*(b.h - a.h);

}

**int** cap[505][505],tipo[505];

double costo[505][505],res;

vector<**int**> ady[505];

**int** from[505];

double d[505];

struct nodo{

**int** id,parent;

double costo;

**bool** operator<(const nodo& a)const{

**return** costo > a.costo;

}

};

**bool** town[505];

double cost[505];

**bool** visited[505];

**bool** spring[505];

**int** s,t,cn;

double valor[505][505];

**int** augment1(**int** source, **int** sink){

fill(from,from+sink+1,-1);

fill(d,d+sink+1,99999999.0);

fill(mk,mk+sink+1,0);

d[source] = 0;

**bool** x = 0;

**bool** y = 0;

**for**(**int** i = 1; i <= cn; i++){

**for**(**int** h = 0; h < cn ; h++){

**int** no = tipo[h];

**int** len = v[no].size();

**for**(**int** k = 0; k < len; k++){

**int** m = v[no][k];

**if**(cap[no][m] && d[m] > d[no] + costo[no][m]){

d[m] = d[no] + costo[no][m];

from[m] = no;

y = 1;

**if**(m == sink)x = 1;

}

}

}

**if**(!y)break;

}

**if**(!x)**return 0;**

**int** actual = sink;

res+=d[sink];

**while**(from[actual]!=-1){

cap[actual][from[actual]]++;

cap[from[actual]][actual]--;

actual = from[actual];

}

**return** 1;

}

**int** max\_flow(**int** sink,**int** source){

**int** r = 0;

**while**(1){

**if**(augment1(sink,source))r++;

else **return** r;

}

}

**int** main()

{

**int** a;

ll q;

**scanf**("%d %d %d %I64d",&n,&s,&t,&q);

N[0].id = 0;

N[n+1].id = n+1;

**for**(**int** i = 1; i <= n; i++){

**scanf**("%I64d %I64d %I64d",&N[i].x,&N[i].y,&N[i].h);

N[i].id = i;

**for**(**int** h = 1; h < i; h++){

// **cout**<<"d "<<dist(N[i],N[h])<<endl;

ll g = dist(N[i],N[h]);

valor[i][h] = valor[h][i] = sqrt((double)g);

**if**(g <= q\*q && N[i].h > N[h].h){

ady[i].push\_back(h);

}

**if**(g <= q\*q && N[i].h < N[h].h){

ady[h].push\_back(i);

}

}

}

**for**(**int** i = 0; i < s; i++){

**scanf**("%d",&a);

tipo[++cn] = a;

cap[0][a] = 1;

v[0].push\_back(a);

spring[a] = 1;

}

**for**(**int** i = 0; i < t; i++){

**scanf**("%d",&a);

cap[a][n+1] = 1;

v[a].push\_back(n+1);

town[a] = 1;

tipo[++cn] = a;

}

cn++;

tipo[cn] = n+1;

**for**(**int** i = 1; i <= n; i++){

**if**(spring[i]){

dijkstra(i);

}

}

**int** k = max\_flow(0,n+1);

**if**(k < t)**printf**("IMPOSSIBLE\n");

else{

**printf**("%lf\n",res);

}

**return 0;**

}

## Hungarian Algorithm

**# include**<bits/stdc++.h>

**using namespace std;**

const **int** MAXN = 105;

const **int** INF = 1000 \* 1000 \* 1000;

**int** n;

**int** a[MAXN][MAXN];

**int** u[MAXN], v[MAXN], link[MAXN], par[MAXN], used[MAXN], minval[MAXN];

**int** main() {

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

**for** (**int** i = 1; i <= n; i++)

**for** (**int** j = 1; j <= n; j++)

**scanf**("%d", &a[i][j]);

**for** (**int** i = 1; i <= n; i++) {

**for** (**int** j = 0; j < MAXN; j++) {

used[j] = false;

minval[j] = INF;

}

**int** j\_cur = 0;

par[j\_cur] = i;

do {

used[j\_cur] = true;

**int** j\_next, delta = INF, i\_cur = par[j\_cur];

**for** (**int** j = 0; j <= n; j++)

**if** (!used[j]) {

**int** cur = a[i\_cur][j] - u[i\_cur] - v[j];

**if** (cur < minval[j]) {

minval[j] = cur; link[j] = j\_cur;

}

**if** (minval[j] < delta) {

delta = minval[j]; j\_next = j;

}

}

**for** (**int** j = 0; j <= n; j++)

**if** (used[j]) {

u[par[j]] += delta; v[j] -= delta; }

else {

minval[j] -= delta;

}

j\_cur = j\_next;

} **while** (par[j\_cur]);

do {

**int** j\_prev = link[j\_cur];

par[j\_cur] = par[j\_prev];

j\_cur = j\_prev;

} **while** (j\_cur > 0);

}

**printf**("%d", -v[0]);

**return 0;**

}

# Geometry & Math

## Estructuras Geometricas

**# include** <bits/stdc++.h>

**using namespace std;**

const double EPS = 0.000000001;

struct Po**int** {

double x, y;

Po**int**(double a = 0, double b=0){

x = a; y = b;

}

double Dist(Po**int** p1){

**return** pow((pow(x-p1.x,2)+pow(y-p1.y, 2)), 1.0/2.0);

}

Po**int** operator - (const Po**int** &p)const{

**return** Po**int**(x-p.x, y-p.y);

}

Po**int** operator + (const Po**int** &p)const{

**return** Po**int**(x+p.x, y+p.y);

}

};

struct Vector{

double a, b;

Vector (Po**int** p1=Po**int**(0, 0), Po**int** p2=Po**int**(0, 0)){

a = p2.x-p1.x;

b = p2.y-p1.y;

}

private: Vector Normal(){

**return** Vector(a, -b);

};

};

struct Recta{

double A, B, C;

Recta(Po**int** p1, Po**int** p2){

Vector v = Vector(p1, p2);

A = v.b;

B = -v.a;

C = v.a\*p1.y - v.b\*p1.x;

Normalizar();

}

Recta(double a = 0, double b = 0, double c = 0){

A = a;

B = b;

C = c;

Normalizar();

}

///Vector v, vetor normal a la recta

///que se quiere obtener

**void** Recta1(Vector v, Po**int** p){

A = v.a;

B = v.b;

C = -A\*p.x-B\*p.y;

Normalizar();

}

//Rectas Paralelas

**bool** operator == (const Recta &P)const{

**return** A==P.A && B == P.B;

}

private:

**void** Normalizar(){

**if**(A < 0)

A\*=-1, B\*=-1, C\*=-1;

**if**(A == 0 && B < 0)

B \*= -1,C\*= -1;

}

double Dist\_Po**int**(Po**int** p){

**return** abs(A\*p.x+B\*p.y+C)/pow(A\*A+B\*B, 1.0/2.0);

}

Po**int** **Int**ersection\_Recta(Recta R2){

Po**int** p;

Recta R1 = Recta(A, B, C);

**if**(R1.A == 0)swap(R1, R2);

p.y = (-R2.C\*R1.A+R1.C\*R2.A)/(R1.A\*R2.B-R2.A\*R1.B);

p.x = (-R1.B\*p.y-R1.C)/R1.A;

**return** p;

}

;

};

struct Circulo{

double h, k, r;

Circulo (Po**int** p = Po**int**(0, 0), double q = 0){

h = p.x;

k = p.y;

r = q;

}

**bool** operator < (const Circulo &Q)const{

**if**(h != Q.h)**return** h < Q.h;

**if**(k != Q.k)**return** k < Q.k;

**return** r < Q.r;

}

/// op-> d**if**erenciar que punto devolver

Po**int** **Int**erseccion\_Recta(Recta R, **int** op){

double x0 = -R.A\*R.C/(R.A\*R.A+R.B\*R.B),

y0 = -R.B\*R.C/(R.A\*R.A+R.B\*R.B);

**if** (R.C\*R.C > r\*r\*(R.A\*R.A + R.B\*R.B)+EPS)

**return** Po**int**(-100000.0, -100000.0);

else **if** (abs (R.C\*R.C - r\*r\*(R.A\*R.A+R.B\*R.B)) < EPS) {

//puts ("1 po**int**");

//**cout** << x0+h << ' ' << y0+k << '\n';

**return** Po**int**(x0+h, y0+k);

}

else {

double d = r\*r - R.C\*R.C/(R.A\*R.A+R.B\*R.B);

double mult = sqrt (d / (R.A\*R.A+R.B\*R.B));

double ax,ay,bx,by;

ax = x0 + R.B \* mult + h;

bx = x0 - R.B \* mult + h;

ay = y0 - R.A \* mult + k;

by = y0 + R.A \* mult + k;

**if**(op == 1)

**return** Po**int**(ax, ay);

else

**return** Po**int**(bx, by);

}

}

///op >

Po**int** **Int**erseccion\_Circle(Circulo C, **int** op){

**return** **Int**erseccion\_Recta(Recta(2.0\*(C.h-h), 2.0\*(C.k-k), -(C.h-h)\*(C.h-h)-(C.k-k)\*(C.k-k)-r\*r+C.r\*C.r), op);

}

**bool** is\_**Int**erseccion\_Circle(Circulo C){

**if**((h-C.h)\*(h-C.h)+(k-C.k)\*(k-C.k) <= (r+C.r)\*(r+C.r))

**return** true;

**return** false;

}

**bool** is\_Inside\_Circle(Po**int** p){

**if**((p.x-h)\*(p.x-h)+(p.y-k)\*(p.y-k) <= r\*r+EPS)

**return** true;

**return** false;

}

};

**int** main(){

}

## Convex Hull

**# include** <cstdio>

**# include** <algorithm>

**using namespace std;**

typedef **long long** ll;

const **long long** RAN = 1000;

struct par{

ll x, y;

par (ll a = 0, ll b = 0){

x = a;

y = b;

}

**bool** operator <(const par &R)

const{

**if** (R.x != x)

**return** R.x > x;

else

**return** R.y > y;

}

};

**int** n, can, con;

**int** P[RAN];

par A[RAN];

ll sol(**int** a, **int** b, **int** c){

**return** (A[b].x - A[a].x) \* (A[c].y - A[a].y)-

(A[b].y - A[a].y) \* (A[c].x - A[a].x);

}

main (){

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

**for** (**int** i = 1; i <= n; i++)

**scanf** ("%lld %lld", &A[i].x, &A[i].y);

sort (A + 1, A + n + 1);

can++;

**for** (**int** i = 1; i <= n; i++){

**while** (can < con && sol (P[con-1], P[con], i) < 0)

con--;

con++;

P[con] = i;

}

can = con;

**for** (**int** i = n - 1; i >= 1; i--){

**while** (can < con && sol (P[con-1], P[con], i) < 0)

con--;

con++;

P[con] = i;

}

**printf** ("%d\n", --con);

**for** (**int** i = 1; i <= con; i++)**printf** ("%lld %lld\n", A[P[i]].x, A[P[i]].y);

}

## Closest Pair of Po**int**s

**# include** <set>

**# include** <cstdio>

**# include** <cmath>

**# include** <algorithm>

**using namespace std;**

struct par {

double x, y;

}a[5005], \*l = &a[0];

struct cmp\_x{

**bool** operator () (const par &a, const par &b)

const {

**return** a.x < b.x;

}

};

struct cmp\_y{

**bool** operator () (const par &a, const par &b)

const {

**return** a.y < b.y;

}

};

double dist(par a, par b){

**return** (double) sqrt( (a.x-b.x)\*(a.x-b.x)+(a.y-b.y)\*(a.y-b.y) );

}

**int** n;

double sol = 1 << 29;

multiset <par, cmp\_y> Q;

multiset <par, cmp\_y>::iterator lo, hi;

**int** main(){

f**scanf**(fe, "%d", &n);

**for**(**int** i = 0; i < n; i++)

f**scanf**(fe, "%lf %lf", &a[i].x, &a[i].y);

sort(a, a+n, cmp\_x());

**for**(par \*i = &a[0]; i != &a[n]; i++){

**while**(i -> x - l -> x >= sol)

Q.erase( Q.find(\*l++) );

lo = Q.lower\_bound(

(par) {i->x, i->y-sol} );

hi = Q.upper\_bound(

(par) {i->x, i->y+sol} );

**for**(; lo != hi; lo++)

sol = min(sol, dist(\*lo, \*i));

Q.insert(\*i);

}

f**printf**(fs, "%.2lf", sol);

**return 0;**

}

## GCD – LCM – Extended GCD

**# include** <cstdio>

**# include** <iostream>

**# include** <algorithm>

**using namespace std;**

**int** GCD(**int** a, **int** b){

**while**(a){

b %= a;

swap(a, b);

}

**return** b;

}

**int** GCD\_extended(**int** a, **int** b, **int** &x, **int** &y){

**if**(a == 0 ){

x = 0; y = 1;

**return** b;

}

**int** x1, y1;

**int** d = GCD\_extended(b%a, a, x1, y1);

x = y1 - (b/a) \* x1;

y = x1;

**return** d;

}

**int** a, b, g, x, y;

**int** main(){

**cin** >> a >> b;

g = GCD\_extended(a, b, x, y);

**cout** << x << " " << y << " "<< g << " " << endl;

}

ll MD(ll A,ll B,ll C){//**return** (A/B)%C

**if**(A%B == 0)

**return** A/B;

**return** (A+(C\*MD(B-(A%B),C%B,B)))/B; }

## Area de Union+Multi Set

**# include** <set>

**# include** <cmath>

**# include** <cstdio>

**# include** <algorithm>

**# define** oo 1 << 29

**using namespace std;**

struct par{

**int** x1, y1, x2, y2;

par (**int** a=0, **int** b=0, **int** c=0, **int** d=0){

x1=a, y1=b, x2=c, y2=d;

}

}A[1005];

struct tri {

**int** x, e, p;

tri (**int** a = 0, **int** b = 0, **int** c = 0){x = a, e = b, p = c;}

**bool** operator < (const tri &a)const{

**return** x < a.x;

}

}S[2005];

struct par1{

**int** y, e;

par1(**int** a = 0, **int** b = 0){y = a, e = b;}

};

struct cmp\_y{

**bool** operator () (const par1 &a, const par1 &b)const{

**return** a.y < b.y;

}

};

multiset <par1, cmp\_y> M;

multiset <par1, cmp\_y>::iterator lo;

**int** n, x, y, z, w, L, l, i, s;

**long long** amount, sol;

**int** main(){

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

**for**(i = 0; i < n; i++){

**scanf**("%d %d %d %d", &x, &y,

&z, &w);

A[i] = par(x, y, z, w);

S[2\*i] = tri(x, 1, i);

S[2\*i+1] = tri(z, -1, i);

}

sort(S, S+2\*n);

amount = 0;

**for**(i = 0; i <= 2\*n; i++){

**if**(S[i].e == -1){

M.erase(M.find((par1){A[S[i].p].y1, 1})); M.erase(M.find((par1){A[S[i].p].y2, -1}));

}

else {

M.insert((par1){A[S[i].p].y1, 1});

M.insert((par1){A[S[i].p].y2, -1});

}

sol += amount\*(**long long** )abs(S[i].x-S[i-1].x);

amount = 0;

**for**(lo = M.begin(); lo != M.end(); lo++){// amount

**if**(s == 0)

l = lo->y;

s += lo->e;

**if**(s == 0){

amount += (lo->y-l);

}

}

}

**printf**("%lld", sol);

**return 0;**

}

## Area de Union+Segment Tree

**# include** <cstdio>

**# include** <algorithm>

**# define** RANG 55000

**# define** oo 20000

**using namespace std;**

struct ct{

**int** x1, y1, x2, y2;

ct(**int** a=0, **int** b=0, **int** c=0, **int** d=0){

x1 = a; y1 = b; x2 = c; y2 = d;

}

}A[1005];

struct par{

**int** x, e, p;

par(**int** a=1, **int** b=1, **int** c=1){

x = a, e = b, p = c;

}

}event[2005];

struct cmp\_x{

**bool** operator () (const par &a, const par &b)const{

**return** a.x < b.x;

}

};

**int** a, b, n, i, x, y, z, w, sol;

**int** T[RANG\*3+5], mk[RANG\*3+5];

**int** update(**int** V, **int** x=1, **int** xend=RANG, **int** P=1){

**if**(b < x || xend < a)

**return** T[P];

**if**(a <= x && xend <= b){

mk[P] += V;

**if**(!mk[P]){

**if**(x == xend)T[P] = 0;

else T[P] = T[P\*2]+T[P\*2+1];

}

else T[P] = xend-x+1;

}

**if**(x == xend)**return** T[P];

**int** pv = (x+xend)/2;

**return** T[P] = update(V, x, pv, P\*2) + update(V, pv+1, xend, P\*2+1);

}

**int** main(){

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

**for**(i = 0; i < n; i++){

**scanf**("%d %d %d %d", &x, &y, &z, &w);

A[i] = ct(x+oo, w+oo, z+oo, y+oo);

event[i\*2] = par(x+oo, 1, i);

event[i\*2+1] = par(z+oo, -1, i);

}

sort(event, event+2\*n, cmp\_x());

**for**(i = 0; i < 2\*n; i++){

sol += T[1]\*(event[i].x-event[i-1].x);

a = A[event[i].p].y1;

b = A[event[i].p].y2-1;

update(event[i].e);

}

**printf**("%d", sol);

**return 0;**

}

## Factorizacion

**# include** <map>

**# include** <cstdio>

**# include** <algorithm>

**using namespace std;**

typedef **long long** ll;

**int** P[1000055], f[50], Div[50], D, x, F;

///Criba para descomponer en

///factores primos

**void** Criba(){

**int** N = 1000007;

**for**(**int** i = 4; i < N; i+=2)P[i] = 2;

Prim[cont\_Prim++] = 2;

**for**(**int** i = 3; i\*i < N; i += 2)

**if**(!P[i]){

Prim[cont\_Prim++] = i;

**for**(**int** j = i\*i; j < N; j += 2\*i)

P[j] = i;

}

}

///Factorizacion

**int** Fact(**int** n){

**int** F = 0;

**while**(P[n]){

f[F++] = P[n];

n /= P[n];

}

f[F++] = n;

sort(f, f+F);

**return** F;

}

///Todos los divisores de un numero

**void** div(**int** v, **int** ini, **int** fin){

**if**(ini == fin){

Div[D++] = v;

**printf**("%d\n", v);

}

else {

**int** m;

**for**(m = ini+1; m < fin && f[m] == f[ini]; m++);

**for**(**int** i = ini; i <= m; i++){

div(v, m, fin);

v \*= f[ini];

}

}

}

///Cantidad de divisores de un numero

**int** Euler(**int** n, **int** F){

**int** c = f[0];

**int** v = 0;

**for**(**int** i = 1; i <= F; i++)

**if**(f[i] != f[i-1]){

v += (c - c/f[i-1]);

c = f[i];

}

else

c \*= f[i];

**return** v;

}

///Inverso Modular

ll MD(ll A,ll B,ll C){//**return** (A/B)%C

**if**(A%B == 0)

**return** A/B;

**return** (A+(C\*MD(B-(A%B),C%B,B)))/B;

}

ll Divisor\_Sumation(){

///Productoria

(Prim[i]^(E[i]+1)-1)/(Prim[i]-1)

sol = 1ll;

**for** (**int** i = 1; i <= c && P[i] <= n; i++)

sol = (sol\*MD((MOD+pow(Prim[i],E[i]+1ll)-1)%MOD,(Prim[i]-1ll),MOD))%MOD;

}

ll Phi(ll n){  
    **if**(n == 1)**return** 2;  
    ll res = 1;  
    **for**(**int** i = 0; i < cp && primes[i] <= n; i++){  
            ll k = 1;  
            ll c = 0;  
        **while**(!(n%primes[i])){  
              n/=primes[i];  
             k\*=primes[i];  
             c = (primes[i]-1);  
        }  
        k/=primes[i];  
        **if**(c)  
        res\*=(k\*c);  
    }  
    **if**(n>1)res\*=(n-1);  
    **return** res;  
}

**int** main(){

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

Criba();

F = Fact(x);

div(1, 0, F);

**printf**("Euler %d\n", Euler(x, F));

}

## Metodo de Gauss

**# include** <cstdio>

**using namespace std;**

**int** N, i, j, k;

double M[305][305], sp;

**int** main(){

**scanf**("%d", &N);

**for**(i = 1; i <= N; i++)

**for**(j = 1; j <= N+1; j++)

**scanf**("%lf", &M[i][j]);

**printf**("---------------------\n");

**for**(i = 1; i <= N; i++)

**for**(j = i+1; j <= N; j++)

**for**(k = N+1; k >= i; k--)

M[j][k] = M[j][i]\*M[i][k] - M[j][k]\*M[i][i];

**for**(i = N; i > 0; i--){

sp = M[i][N+1];

**for**(j = i+1; j <= N+1; j++)

sp -= M[i][j]\*M[j][j];

M[i][i] = sp/M[i][i];

}

**for**(i = 1; i <= N; i++)

**printf**("%.0lf ", M[i][i]);

**return 0;**

}

## Fibonacci Logaritmico

**# include** <cstdio>

**# include** <cmath>

**using namespace std;**

**int** M[5][5][70], S[5][5], i, lg, a, b, c, d;

**long long** n;

**int** main(){

M[1][1][0] = 0;

M[1][2][0] = 1;

M[2][1][0] = 1;

M[2][2][0] = 1;

**for**(i = 1; i <= 63; i++){

M[1][1][i] = (M[1][1][i-1] \*M[1][1][i-1] +M[1][2][i-1] \* M[2][1][i-1])%10;

M[1][2][i] = (M[1][1][i-1] \* M[1][2][i-1] + M[1][2][i-1] \* M[2][2][i-1])%10;

M[2][1][i] = (M[2][1][i-1] \*M[1][1][i-1] + M[2][2][i-1] \* M[2][1][i-1])%10;

M[2][2][i] = (M[2][2][i-1] \* M[2][2][i-1] + M[2][1][i-1] \* M[1][2][i-1])%10;

}

**while**(**scanf**("%lld", &n) != EOF){

S[1][1] = S[2][2] = 1;

S[1][2] = S[2][1] = 0;

lg = (**int**) log2(n);

**for**(i = 0; i <= lg; i++){

**if**(n & (1ULL << i)){

a = (S[1][1]\*M[1][1][i]+S[1][2]\*M[2][1][i])%10;

b = (S[1][1]\*M[1][2][i]+S[1][2]\*M[2][2][i])%10;

c = (S[2][1]\*M[1][1][i]+S[2][2]\*M[2][1][i])%10;

d = (S[2][1]\*M[1][2][i]+S[2][2]\*M[2][2][i])%10;

S[1][1] = a;

S[1][2] = b;

S[2][1] = c;

S[2][2] = d;

}

}

**printf**("%d\n", S[2][1]%10);

}

}

## Binary Exponetation

**# include** <cstdio>

**using namespace std;**

**int** a, b;

cons t**int** MOD = 1000000007;

**int** binpow (**int** a, **int** n) {

**int** res = 1;

**while** (n) {

**if** (n & 1)

res = (a\*res)%MOD;

a = (a\*a)%MOD;

n = n >> 1;

/\*\*Desplaza los bits a la

derecha y desaparece el primero\*/

}

**return** res;

}

**int** main(){

**while**(1){

**scanf**("%d %d", &a, &b);

**printf**("%d\n", binpow(a, b));

}

}

## Factorial Compactado

**# include** <iostream>

**# include** <cstdio>

**# include** <algorithm>

**# define** RANG 1000000

**# define** MOD 10

**using namespace std;**

**int** n, C, tmp, S;

**int** i, j, P[1000005], M[1000005], E[1000005];

string s;

**int** main() {

//Criba, para descomponer en factores primos

**for**(i = 4; i <= RANG; i += 2)P[i] = 2;

**for**(i = 3; i\*i <= RANG; i+=2)

**if**(!P[i])

**for**(j = i\*i; j <= RANG; j += 2\*i)

P[j] = i;

/\*\*Variaciones sin repeticiones

25'000'000, hasta mas\*\*/

**scanf**("%d %d", &n, &C);

**for**(i = 1; i <= C; i++)

**scanf**("%d",&M[i]);

//Compactar factoriales

// i! ^ E[i]

E[n]++;

**for**(i = 1; i <= C; i++){

**if**(M[i] > 1)E[M[i]]--;

M[i] = 0;

}

//Descompone el factorial en la productoria de sus terminos

**for**(i = n-1; i >= 2; i--)

E[i] += E[i+1];

//Descomponer el factoria

//en potencias de factores primos

**for**(i = n; i >= 2; i--)

**if**(P[i]){

E[i/P[i]] += E[i];

E[P[i]] +=E[i];

E[i] = 0;

}

//Espec**if**icidad para eliminar los

//ultimos digitos iguales a 0

tmp = min(E[2], E[5]);

E[2] -= tmp; E[5] -= tmp;

//Calcular la Variacion expresada

//en la productoria de factores primos

S = 1;

**for**(i = 2; i <= n; i++){

S = (S \* modexp(i, E[i])) % 10;

E[i] = 0;

}

**printf**("%d\n", S);

**return 0;**

}

## Calculo de Expresiones

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Calculation of the value of the expression.

Uses recursive descent parser. Supports doubles.

Available operations:

+ - / \* Usual meaning

- Unary minus

# Root (3 # 8 = 2)

^ Power (2 ^ 3 = 8)

# Square root (unary)

TODO: find a good problem on it

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <bits/stdc++.h>

using namespace std;

string s;

int len;

double ans;

int p;

double exp();

double term();

double interm();

double factor();

double pr1Sign(char c) {

return (c == '-' || c == '+');

}

double pr2Sign(char c) {

return (c == '\*' || c == '/');

}

double pr3Sign(char c) {

return (c == '#' || c == '^');

}

double factor() {

double res = 0.0, dpow = 1.0;

bool dot = false;

if (p >= len) {

return res;

}

if (s[p] == '(') {

p++;

res = exp();

p++;

return res;

}

if (s[p] == '-') {

p++;

res = interm();

return -res;

}

if (s[p] == '#') {

p++;

res = interm();

return pow(res, 0.5);

}

while (p < len && ((s[p] >= '0' && s[p] <= '9') || s[p] == '.')) {

if (s[p] == '.')

dot = true;

else {

res = res \* 10 + s[p] - '0';

if (dot)

dpow \*= 10.0;

}

p++;

}

return res / dpow;

}

double interm() {

double res, t;

char c;

res = factor();

while (p < len && pr3Sign(s[p])) {

c = s[p];

p++;

t = factor();

if (c == '^') {

if (t < 1.0 && t > -1.0 && t != 0 && res < 0)

res = -pow(-res, t);

else

res = pow(res, t);

}

if (c == '#') {

if (t < 0)

res = -pow(-t, (1.0 / res));

else

res = pow(t, (1.0 / res));

}

}

return res;

}

double term() {

double res, t;

char c;

res = interm();

while (p < len && pr2Sign(s[p])) {

c = s[p];

p++;

t = interm();

if (c == '\*')

res \*= t;

if (c == '/')

res /= t;

}

return res;

}

double exp() {

double res, t;

char c;

res = term();

while (p < len && pr1Sign(s[p])) {

c = s[p];

p++;

t = term();

if (c == '-')

res -= t;

if (c == '+')

res += t;

}

return res;

}

void remove\_whitespaces(string &s) {

string temp = s;

s = "";

for (int i = 0; i < (int) temp.length(); i++)

if (temp[i] != ' ')

s.append(1, temp[i]);

}

int main() {

getline(cin,s);

remove\_whitespaces(s);

len = s.length();

ans = exp();

printf("%lg",ans);

return 0;

}

# String Algorithms

## Palindromes 1

O(n^2)

**# include** <cstdio>

**# include** <cstring>

**using namespace std;**

**int** i, j, ls;

**bool** m[5005][5005];

**char** s[5005];

**int** main(){

**scanf**("%s", s+1);

ls = strlen(s+1);

**for**(i = 1; i <= ls; i++)

m[1][i] = true;

**for**(i = 2; i <= ls; i++){

**for**(j = i; j <= ls; j++)

**if**(s[j] == s[j-i+1] && m[i-(i > 2? 2:1)][j-1])

m[i][j] = true;

}

**return 0;**

}

## Knutt Morris Pratt (Prefix Function)

/\*\*

Determina las ocurrencias de un

patron dentro de un texto

O(N+M)

\*/

**# include** <cstring>

**# include** <cstdio>

**using namespace std;**

**int** i, k, lp, lt;

**int** F[5005];

**char** Text[5005], Pattern[5005];

**int** main(){

**scanf**("%s\n%s", Text+1, Pattern+1);

lp = strlen(Pattern + 1);

lt = strlen(Text + 1);

**for**(i = 2; i <= lp; i++){

**while**(k > 0 && Pattern[k + 1] != Pattern[i])

k = F[k];

**if**(Pattern[k + 1] == Pattern[i])

k++;

F[i] = k;

}

k = 0;

**for**(i = 1; i <= lt; i++){

**while**(k > 0 && Pattern[k + 1] != Text[i])

k = F[k];

**if**(Pattern[k + 1] == Text[i])k++;

**if**(k == lp){

**printf**("%d ", i-lp+1);

k = F[k];

}

}

**return 0;**

}

## Z Function

/// En cada posicion de Z[i] esta guardado el tamaño del mayor sufijo

/// que es prefijo de la palabra y empieza en "i".

**# include** <cstdio>

**# include** <cstring>

**# include** <algorithm>

**using namespace std;**

**int** i, l, r, ls, Z[5005];

**char** s[5005];

**int** main(){

**scanf**("%s", &s);

ls = strlen(s);

**for** (**int** i = 1, l = 0, r = 0; i < ls; ++i) {

**if** (i <= r)

Z[i] = min (r - i + 1, Z[i - l]);

**while** (i + Z[i] < ls && s [ Z[i] ] == s [i + Z[i]])

++Z[i];

**if** (i + Z[i] - 1> r)

l = i, r = i + Z[i] - 1;

}

**for**(i = 0; i < ls; i++)

**printf**("%d ", Z[i]);

}

## Cyclic Shift

**# include** <bits/stdc++.h>

**using namespace std;**

string min\_cyclic\_sh**if**t(string s){

s += s;

**int** n = (**int**) s.length ();

**int** i = 0, ans = 0;

**while** (i < n/2){

ans = i;

**int** j = i+1, k = i;

**while** (j < n && s[k] <= s[j]){

**if** (s[k] < s[j]) k = i;

else ++k; ++j;

}

**while** (i <= k)

i += j - k;

}

**return** s.substr(ans, n / 2);

}

string S;

**int** n;

**char** A[505];

**int** main(){

**cin** >> n;

s**printf**(A, "%d", n);

S = min\_cyclic\_sh**if**t((string)A);

**cout** << S;

}

## Hashing

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Hashing in strings based problems.

This code compares substrings using two hashes (one

uses 2^64 as a modulo, another 10^9 + 7)

Based on problem C from here: http://codeforces.ru/gym/100133

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <bits/stdc++.h>

using namespace std;

const int MAXN = 105000;

const int mod = (int) 1e9 + 7;

const int p = 53;

string s;

int n, m;

long long h1[MAXN], h2[MAXN];

long long pp1[MAXN], pp2[MAXN];

long long getHash1(int l, int r) {

l--; r--;

long long h = h1[r];

if (l > 0)

h -= h1[l - 1];

h \*= pp1[n - 1 - r];

return h;

}

long long getHash2(int l, int r) {

l--; r--;

long long h = h2[r];

if (l > 0)

h = (h - h2[l - 1] + mod) % mod;

h = (h \* pp2[n - 1 - r]) % mod;

return h;

}

int main() {

getline(cin, s);

n = (int) s.length();

pp1[0] = 1; pp2[0] = 1;

for (int i = 1; i <= n; i++) {

pp1[i] = pp1[i - 1] \* p;

pp2[i] = (pp2[i - 1] \* p) % mod;

}

h1[0] = s[0] - 'a' + 1;

h2[0] = (s[0] - 'a' + 1) % mod;

for (int i = 1; i < n; i++) {

h1[i] = h1[i - 1] + pp1[i] \* (s[i] - 'a' + 1);

h2[i] = (h2[i - 1] + pp2[i] \* (s[i] - 'a' + 1)) % mod;

}

scanf("%d", &m);

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

int a, b, c, d;

scanf("%d %d %d %d", &a, &b, &c, &d);

if (getHash1(a, b) == getHash1(c, d) && getHash2(a, b) == getHash2(c, d))

puts("Yes");

else

puts("No");

}

return 0;

}

Dinamic Programing

## Longest Common Subsequence

**# include** <cstdio>

**# include** <algorithm>

**# define** RANG 100

**using namespace std;**

**int** c, la, lb, Dp[RANG][RANG];

**char** A[RANG], B[RANG];

**int** main() {

**scanf** ("%s\n", A + 1);

**scanf** ("%s", B + 1);

la = strlen (A + 1);

lb = strlen (B + 1);

**for** (**int** i = 1; i <= lb; i++)

**for** (**int** j = 1; j <= la; j++)

**if** (B[i] == A[j])

Dp[i][j] = Dp[i - 1][j - 1] + 1;

else

Dp[i][j] = max (Dp[i - 1][j], Dp[i][j - 1]);

**printf** ("%d\n", Dp[lb][la]);

**return 0;**

}

## Longest Increasing or Decreasing Secuence

**# include** <cstdio>

**# include** <algorithm>

**using namespace std;**

**int** i, j, n, s, sol[505], L[505], Id[505], a[505], up;

**int** main(){

freopen("LIS.in", "r", stdin);

freopen("LIS.out", "w", stdout);

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

**for**(i = 1; i <= n; i++)

**scanf**("%d", &a[i]);

**for**(i = 1; i <= n; i++){

**if**(a[i] > sol[s]){

sol[++s] = a[i];

Id[s] = i;

L[i] = Id[s-1];

}

else{

up = upper\_bound(sol+1, sol+s+1, a[i])-sol;

sol[up] = a[i];

Id[up] = i;

L[i] = Id[up-1];

}

}

**printf**("%d", s);

**return 0;**

}

## Counting Change

/\*\*

Sirve para ver de cuantas **for**mas

con una cantidad de de valores

se puede obtener un numero

O(M\*N+M\*log2(M)

\*/

**# include** <cstdio>

**# include** <algorithm>

**using namespace std;**

**int** M, N, i, j, c[20], S[5005];

**int** main(){

**scanf**("%d %d", &M, &N);

**for**(i = 1; i <= M; i++)

**scanf**("%d", &c[i]);

sort(c+1, c+M+1);

S[0] = 1;

**for**(i = 1; i <= M; i++)

**for**(j = c[i]; j <= N; j++)

S[j] +=S[j-c[i]];

**printf**("%d", S[N]);

**return 0;**

}

## Edit Distance

**# include** <cstdio>

**# include** <algorithm>

**# define** RANG 100

**using namespace std;**

**int** la, lb, s, Dp[RANG][RANG];

**char** A[RANG], B[RANG];

**int** main() {

**scanf** ("%s", A + 1);

**scanf** ("%s", B + 1);

la = strlen (A + 1);

lb = strlen (B + 1);

**for** (**int** i = 0; i <= max(la, lb); i++) {

Dp[0][i] = i;

Dp[i][0] = i;

}

**for** (**int** i = 1; i <= lb; i++)

**for** (**int** j = 1; j <= la; j++) {

s = 1;

**if** (B[i] == A[j])

s = 0;

Dp[i][j] = min (min ( Dp[i - 1][j - 1] + s, Dp[i - 1][j] + 1), Dp[i][j - 1] + 1);

}

**printf** ("%d\n", Dp[lb][la]);

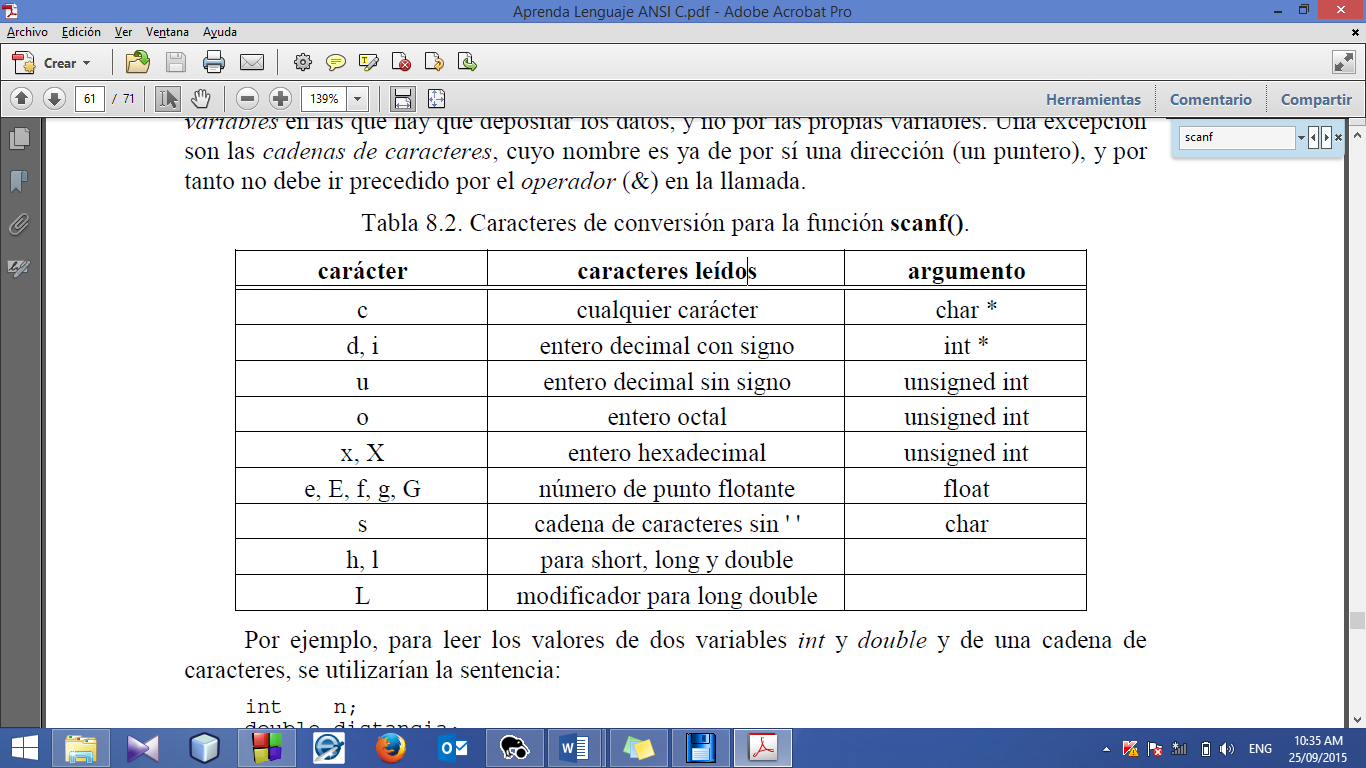
**return 0;**

}

# Especificaciones Lenguajes C++

## Funciones Entrada C++

* **scanf**(tipo, &var) lee los datos de entrada en el stdin (flujo de entrada estandar)
* f**scanf**(file, tipo, &var) lee en un flujo de entrada dado, por lo general un fichero *file*
* **scanf**(string, tipo, &var) obtiene la entrada que se va a analizar desde un string



En los **for**matos de la cadena de control de **scanf()** pueden **int**roducirse *corchetes* [...],

que se utilizan como sigue. La sentencia,

**scanf**("%[AB \n\t]", s); // se leen solo los caracteres indicados

lee caracteres hasta que encuentra uno diferente de (’A’,’B’,’ ’,’\n’,’\t’). En otras

palabras, se leen sólo los caracteres que aparecen en el corchete. Cuando se encuentra un

carácter dist**int**o de éstos se detiene la lectura y se devuelve el control al programa que llamó a

**scanf()**. Si los corchetes contienen un carácter (^), se leen todos los caracteres dist**int**os de los

caracteres que se encuentran dentro de los corchetes a continuación del (^). Por ejemplo, la

sentencia,

**scanf**(" %[^\n]", s);

lee todos los caracteres que encuentra hasta que llega al carácter ***nueva línea*** ’\n’. Esta

sentencia puede utilizarse por tanto para leer líneas completas, con blancos incluidos.

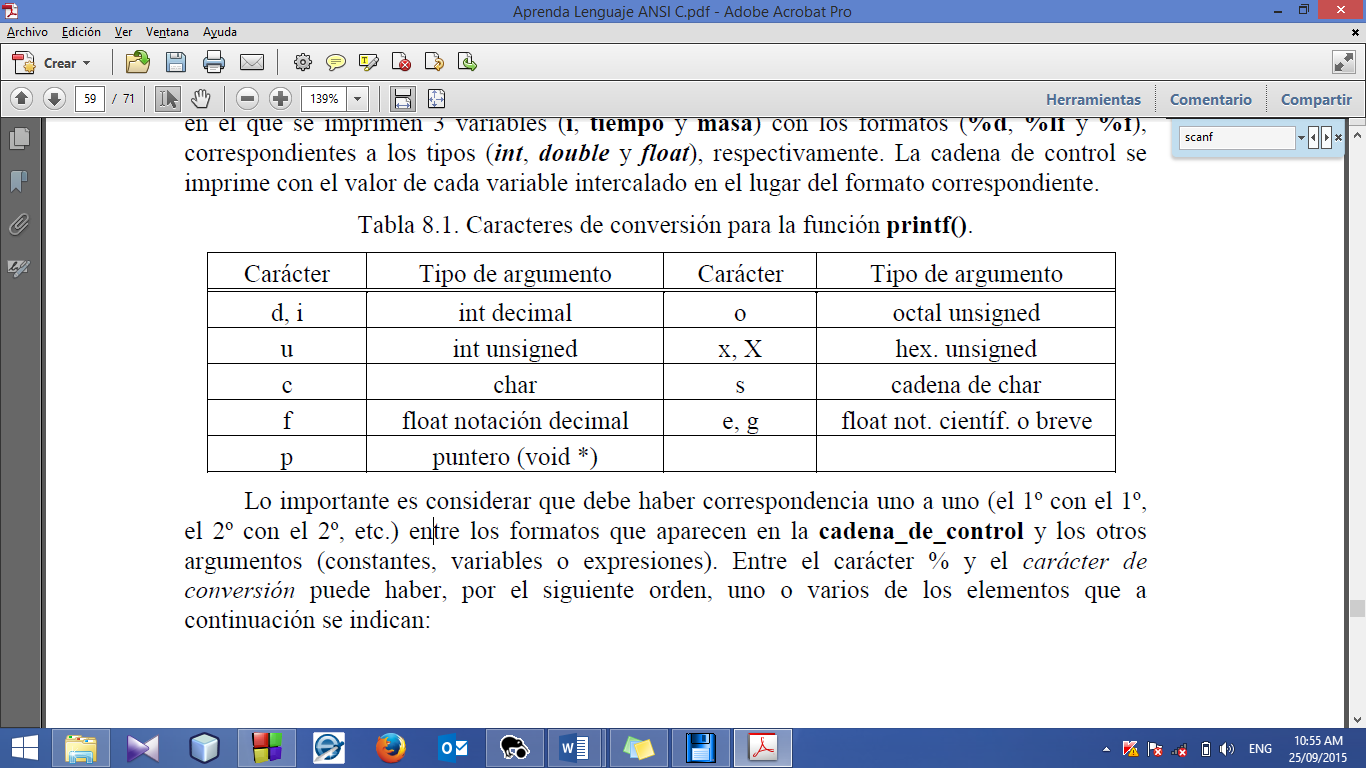
Recuérdese que con el **for**mato ***%s*** la lectura se detiene al llegar al primer delimitador

(carácter blanco, tabulador o nueva línea).

## Funciones Salidas C++

* **printf**(tipo, &var) lee los datos de entrada en el stdin (flujo de entrada estandar)
* f**printf**(file, tipo, &var) lee en un flujo de entrada dado, por lo general un fichero *file*
* spr**int**(string, tipo, &var) obtiene la entrada que se va a analizar desde un string

**Tipo (Argumentos)**



* Un número entero positivo, que indica la *anchura* mínima del campo en caracteres.
* Un signo (-), que indica *alineamiento* por la izda (el defecto es por la dcha).
* Un punto (.), que separa la anchura de la *precisión*.
* Un número entero positivo, la *precisión*, que es el nº máximo de caracteres a imprimir
* en un *string*, el nº de decimales de un *float* o *double*, o las c**if**ras mínimas de un ***int***o *long*.
* Un *cual****if****icador*: una (h) para *short* o una (l) para *long* y *double*

## Sincronizar Entrada/Salida ***iostream***

ios\_base::sync\_with\_stdio(0);

**cin**.tie(0);