**KMP**

#include<bits/stdc++.h>

**using** **namespace** std**;**

#define MAX\_N 100010 //1000010

string T**,** P**;** // T = text, P = pattern

int b**[**MAX\_N**],** n**,** m**;** // b = back table, n = length of T, m = length of P

void kmpPre**()** **{**

int i **=** 0**,** j **=** **-**1**;** b**[**0**]** **=** **-**1**;**

**while** **(**i **<** m**)** **{**

**while** **(**j **>=** 0 **&&** P**[**i**]** **!=** P**[**j**])** j **=** b**[**j**];**

i**++;** j**++;**

b**[**i**]** **=** j**;**

**}** **}**

void kmp**()** **{**

int i **=** 0**,** j **=** 0**;**

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

**while** **(**j **>=** 0 **&&** T**[**i**]** **!=** P**[**j**])** j **=** b**[**j**];**

i**++;** j**++;**

**if** **(**j **==** m**)** **{**

cout**<<**"P se encontro en la posicion "**<<**i **-** j**<<**"\n"**;**

j **=** b**[**j**];**

**}** **}** **}**

int main**()** **{**

T**=**"I DO NOT LIKE SEVENTY SEV BUT SEVENTY SEVENTY SEVEN"**;**

P**=**"SEVENTY SEVEN"**;**

n**=**T**.**length**();**

m**=**P**.**length**();**

kmpPre**();**

kmp**();**

**return** 0**;**

**}**

**LCS**

//Encontrar la subcadena mas grande o larga entre dos string

#include<bits/stdc++.h>

**using** **namespace** std**;**

int LCS**(**string a**,**string b**){**

int l1**,**l2**,**maxx**=**0**;**

l1**=**a**.**length**();**

l2**=**b**.**length**();**

int memo**[**l1**+**1**][**l2**+**1**];**

memset**(**memo**,**0**,sizeof(**memo**));**

**for(**int i**=**0**;**i**<**l1**;**i**++){**

**for(**int j**=**0**;**j**<**l2**;**j**++){**

**if(**a**[**i**]==**b**[**j**]){**

**if(**i**==**0 **||** j**==**0**){**

memo**[**i**][**j**]=**1**;**

**}else** **{**

memo**[**i**][**j**]=**memo**[**i**-**1**][**j**-**1**]+**1**;**

**}**

maxx**=**max**(**maxx**,**memo**[**i**][**j**]);**

**}**

**}**

**}**

/\* for(int i=0;i<l1;i++){

for(int j=0;j<l2;j++){

cout<<memo[i][j]<<" ";

}

cout<<"\n";

}\*/

**return** maxx**;**

**}**

int main**(){**

string a**,**b**;**

a**=**"alsdfkjfjkdsal"**;**

b**=**"fdjskalajfkdsla"**;**

cout**<<**LCS**(**a**,**b**)<<**" Es la longitud de la subcadena mas grande"**;**

**return** 0**;**

**}**

**DSU**

///Permite determinar de manera eficiente a que conjunto pertenece un elemento,

///si dos elementos se encuentran en un mismo conjunto y unir dos conjuntos en un uno.

#include<bits/stdc++.h>

**using** **namespace** std**;**

const int MAX**=**1000**;**

struct dsu**{**

int padre**[**MAX**],**len**[**MAX**];**

void iniciar**(**int n**){**

//indexar desde 0,1 depende del prgblema

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

padre**[**i**]=**i**;**

len**[**i**]=**1**;**

**}**

**}**

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

**if(**padre**[**x**]==**x**){**

**return** x**;**

**}**

**return** find\_**(**padre**[**x**]);**

**}**

//El padre de a, que apunte a el padre de b

void merge\_**(**int a**,**int b**){**

padre**[**find\_**(**a**)]=**find\_**(**b**);**

**}**

///Comprension de caminos

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

**if(**padre**[**x**]==**x**){**

**return** x**;**

**}**

**return** padre**[**x**]=**find\_**(**padre**[**x**]);**

**}**

//pasar conjunto mas pequeño al mas grande

void merge\_c**(**int a**,**int b**){** //a sera el mas grande, b el mas pequeño

int fa**=**find\_c**(**a**);**

int fb**=**find\_c**(**b**);**

**if(**fa**!=**fb**){**

**if(**len**[**fa**]<**len**[**fb**]){**

swap**(**fa**,**fb**);**

**}**

len**[**fa**]+=**len**[**fb**];**

padre**[**fb**]=**fa**;**

**}**

**}**

**};**

int main**(){**

dsu d**;**

d**.**iniciar**(**10**);**

//cout<<d.find\_(5)<<"\n";

d**.**merge\_c**(**2**,**1**);**

d**.**merge\_c**(**3**,**1**);**

d**.**merge\_c**(**4**,**2**);**

cout**<<**d**.**find\_**(**3**)<<**"\n"**;**

d**.**merge\_c**(**4**,**1**);**

cout**<<**d**.**find\_**(**4**)<<**"\n"**;**

**return** 0**;**

**}**

**FenwickTree**

///Estructura de datos que permite procesar consultas por rangos y actualizaciones individuales sobre un arreglo.

#include<bits/stdc++.h>

**using** **namespace** std**;**

**typedef** vector**<**int**>** vi**;**

struct FenwickTree**{**

vi ft**;**

//indexamos desde 1

void iniciar**(**int n**){** ft**.**assign**(**n **+** 1**,** 0**);** **}**

void iniciar**(**vi **&**v**){**

ft**.**assign**(**v**.**size**()** **+** 1**,** 0**);**

**for(**int i **=** 1**;** i **<=** v**.**size**();** i**++)**

actualizar**(**i**,** v**[**i **-** 1**]);**

**}**

//bit menos significativo en 1

int lsOne**(**int n**){** **return** n **&** **(-**n**);** **}**

int rsq**(**int i**){**//suma de 1 hasta i

int acum **=** 0**;**

**for(;** i**;** i **-=** lsOne**(**i**))** acum**+=**ft**[**i**];**

**return** acum**;**

**}**

int rsq**(**int i**,** int j**){**//suma de i hasta j

**return** rsq**(**j**)** **-** **((**i**==**1**)?** 0**:** rsq**(**i **-** 1**));**

**}**

//Suma n a la posicion pos

void actualizar**(**int pos**,** int n**){**//n = nuevo - anterior

**for(;** pos **<** ft**.**size**();** pos **+=** lsOne**(**pos**))**

ft**[**pos**]** **+=** n**;**

**}**

void r**(**int pos**,** int n**){**//n = nuevo - anterior actualizaciones con 0 y demas

**for(;** pos **<** ft**.**size**();** pos **+=** lsOne**(**pos**))**

ft**[**pos**]** **+=** n**\*-**1**;**

**}**

**};**

int main**(){**

int arr**[]** **=** **{**0**,** 1**,** 0**,** 1**,** 2**,** 3**,** 2**,** 1**,** 1**,** 0**};**

vi ar**;**

**for(**int i **=** 0**;** i **<** 10**;** i**++)** ar**.**push\_back**(**arr**[**i**]);**

FenwickTree tree**;**

tree**.**iniciar**(**ar**);**

cout **<<** tree**.**rsq**(**4**)** **<<** endl**;**

cout **<<** tree**.**rsq**(**2**,** 2**)** **<<** endl**;**

tree**.**actualizar**(**2**,**2**);**

cout **<<** tree**.**rsq**(**2**,** 2**)** **<<** endl**;**

**return** 0**;**

**}**

**LAZY PROPAGATION**

#include<bits/stdc++.h>

**using** **namespace** std**;**

struct st\_LP**{** ///info de cada nodo

st\_LP **\***left**,\***right**;**

int sum**,**l**,**r**,**m**,**lazy**;**

st\_LP**(**int l**,**int r**):**l**(**l**),**r**(**r**),**sum**(**0**),**lazy**(**0**){** //intervalos men,may

**if(**l**!=**r**){**

m**=(**l**+**r**)/**2**;**

left**=new** st\_LP**(**l**,**m**);**

right**=new** st\_LP**(**m**+**1**,**r**);**

**}**

**}**

void propagate**(){**

**if(**lazy**!=**0**){**

///actualizar nodo

sum**=(**r**-**l**+**1**)\***lazy**;**

**if(**l**!=**r**){**

left**->**lazy**=**lazy**;**

right**->**lazy**=**lazy**;**

**}**

///actualizar hijos

lazy**=**0**;**

**}**

**}**

///actualice de a hast\_LPa b con el valor v

void update**(**int a**,**int b**,**int v**){**

propagate**();**

**if(**a**>**r **||** b**<**l**)** **return** **;**

**if(**a**<=**l **&&** r**<=**b**){**

lazy**=**v**;**

propagate**();**

**return;**

**}**

left**->**update**(**a**,**b**,**v**);**

right**->**update**(**a**,**b**,**v**);**

sum**=**left**->**sum**+**right**->**sum**;**

**}**

//suma en el intervalo a,b O l,r

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

propagate**();**

**if(**a**>**r **||** b**<**l**)**

**return** 0**;**

**if(**a**<=**l **&&** r**<=**b**)**

**return** sum**;**

**return** left**->**get**(**a**,**b**)+**right**->**get**(**a**,**b**);**

**}**

**};**

int main**(){**

st\_LP tree**(**0**,**8**);**

**for(**int i**=**0**;**i**<**8**;**i**++){**

int x**;**

cin**>>**x**;**

tree**.**update**(**i**,**i**,**x**);**

**}**

//

cout**<<**tree**.**get**(**1**,**1**)<<**"\n"**;**

//intervalo i,j,nuevo valor

tree**.**update**(**1**,**1**,**0**);** //no actualiza el rango en 0

cout**<<**tree**.**get**(**1**,**1**);**

**return** 0**;**

**}**

/\*\*

1 2 7 5 9 10 2 3

\*/

**SEGMENT TREE**

#include<bits/stdc++.h>

**using** **namespace** std**;**

struct st**{** ///info de cada nodo

st **\***left**,\***right**;**

int sum**,**l**,**r**,**m**;**

st**(**int l**,**int r**):**l**(**l**),**r**(**r**),**sum**(**0**){** //intervalos men,may

**if(**l**!=**r**){**

m**=(**l**+**r**)/**2**;**

left**=new** st**(**l**,**m**);**

right**=new** st**(**m**+**1**,**r**);**

**}**

**}**

//cambie el valor de la posicion de pos por el valor v

void update**(**int pos**,**int v**){**

**if(**l**==**r**){**

sum**=**v**;**

**}else{**

**if(**pos**<=**m**){**

left**->**update**(**pos**,**v**);**

**}else{**

right**->**update**(**pos**,**v**);**

**}**

sum**=**left**->**sum**+**right**->**sum**;**

**}**

**}**

//suma en el intervalo a,b O l,r

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

**if(**a**>**r **||** b**<**l**)**

**return** 0**;**

**if(**a**<=**l **&&** r**<=**b**)**

**return** sum**;**

**return** left**->**get**(**a**,**b**)+**right**->**get**(**a**,**b**);**

**}**

**};**

int main**(){**

st tree**(**0**,**8**);**

**for(**int i**=**0**;**i**<**8**;**i**++){**

int x**;**

cin**>>**x**;**

tree**.**update**(**i**,**x**);**

**}**

cout**<<**tree**.**get**(**1**,**3**);**

**return** 0**;**

**}**

**BELLMAN FORD**

//distancia minima con pesos negativos, y ciclos negativos

//bellman ford, desde un nodo a todos los demas nodos

#include<bits/stdc++.h>

**using** **namespace** std**;**

const int oo**=**100000**;**

struct edge**{**

int u**,**v**,**w**;**

**};**

int main**(){**

int nodos**,**aristas**;**

cin**>>**nodos**>>**aristas**;**

edge edges**[**oo**];**

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

int u**,**v**,**w**;**

cin**>>**u**>>**v**>>**w**;**

edges**[**i**].**u**=**u**;**

edges**[**i**].**v**=**v**;**

edges**[**i**].**w**=**w**;**

**}**

int source**;**

//cin>>source;

source**=**0**;**

int min\_dis**[**oo**];**

**for(**int i**=**0**;**i**<**nodos**+**5**;**i**++)**

min\_dis**[**i**]=**100000**;**

min\_dis**[**source**]=**0**;**

**for(**int i**=**0**;**i**<**nodos**-**1**;**i**++){**

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

**if(**min\_dis**[**edges**[**j**].**v**]>**min\_dis**[**edges**[**j**].**u**]+**edges**[**j**].**w**){**

min\_dis**[**edges**[**j**].**v**]=**min\_dis**[**edges**[**j**].**u**]+**edges**[**j**].**w**;**

//memorizar el padre para el camino (e.u) edges[j].u

**}**

**}**

**}**

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

**if(**min\_dis**[**edges**[**j**].**v**]>**min\_dis**[**edges**[**j**].**u**]+**edges**[**j**].**w**){**

cout**<<**"Existe ciclo negativo\n"**;**

**}**

**}**

cout**<<**"Minimas distancias desde source\n"**;**

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

cout**<<**i**+**1**<<**" = "**<<**min\_dis**[**i**]<<**"\n"**;**

**}**

**return** 0**;**

**}**

**COMPONENTES FUERTEMENTE CONEXAS DE UN GRAFO DIRIGIDO**

#include<bits/stdc++.h>

**using** **namespace** std**;**

vector**<**int**>**dfs\_low**,**dfs\_num**,**s**;**

bool vis**[**100**];**

vector**<**int**>**grafo**[**100**];**

int dfsCont**=**0**,**N**;**

void tarjanSCC**(**int u**){**

dfs\_low**[**u**]=**dfs\_num**[**u**]=**dfsCont**++;**

s**.**push\_back**(**u**);**

vis**[**u**]=true;**

**for(**int i**=**0**;**i**<**grafo**[**u**].**size**();**i**++){**

int v**=**grafo**[**u**][**i**];**

**if(**dfs\_num**[**v**]==-**1**){**

tarjanSCC**(**v**);**

**}**

**if(**vis**[**v**]){**

dfs\_low**[**u**]=**min**(**dfs\_low**[**u**],**dfs\_low**[**v**]);**

**}**

**}**

**if(**dfs\_low**[**u**]==**dfs\_num**[**u**]){**

cout**<<**"Componentes:\n"**;**

**while(true){**

int v**=**s**.**back**();**

s**.**pop\_back**();**

cout**<<**v**<<**"\n"**;**

vis**[**v**]=false;**

**if(**v**==**u**){**

**break;**

**}**

**}**

**}**

**}**

void solve**(){**

dfs\_num**.**assign**(**N**+**1**,-**1**);**

dfs\_low**.**assign**(**N**+**1**,**0**);**

memset**(**vis**,**0**,sizeof(**vis**));**

dfsCont**=**0**;**

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

**if(**dfs\_num**[**i**]==-**1**){**

tarjanSCC**(**i**);**

**}**

**}**

**}**

int main**(){**

N**=**7**;**

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

grafo**[**i**].**clear**();**

**}**

grafo**[**0**].**push\_back**(**1**);**

grafo**[**1**].**push\_back**(**3**);**

grafo**[**3**].**push\_back**(**2**);**

grafo**[**2**].**push\_back**(**1**);**

grafo**[**3**].**push\_back**(**4**);**

grafo**[**4**].**push\_back**(**5**);**

grafo**[**5**].**push\_back**(**7**);**

grafo**[**7**].**push\_back**(**6**);**

grafo**[**6**].**push\_back**(**4**);**

solve**();**

**return** 0**;**

**}**

**COMPROBACION BIPARTITO**

#include<bits/stdc++.h>

**using** **namespace** std**;**

const int MAX**=**100**;**

vector**<**vector**<**int**>>**g**(**MAX**);**

int vis**[**MAX**];**

bool bipartito**(**int u**){**

queue**<**int**>**q**;**

q**.**push**(**u**);**

vis**[**u**]=**1**;**

**while(**q**.**size**()){**

u**=**q**.**front**();**

q**.**pop**();**

**for(**int i**=**0**;**i**<**g**[**u**].**size**();**i**++){**

int v**=**g**[**u**][**i**];**

**if(**vis**[**v**]==**0**){**

q**.**push**(**v**);**

**if(**vis**[**u**]==**1**){**

vis**[**v**]=**2**;**

**}else** **if(**vis**[**u**]==**2**){**

vis**[**v**]=**1**;**

**}**

**}else** **if(**vis**[**v**]==**vis**[**u**]){**

**return** **false;**

**}**

**}**

**}**

**return** **true;**

**}**

int main**()** **{**

//freopen("in.txt","r",stdin);

//freopen("out.txt","w",stdout);

ios\_base**::**sync\_with\_stdio**(**0**);**

cin**.**tie**(**0**);**

int nodos**,**aristas**,**u**,**v**;**

cin**>>**nodos**>>**aristas**;**

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

g**[**i**].**clear**();**

vis**[**i**]=**0**;**

**}**

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

cin**>>**u**>>**v**;**

g**[**u**].**push\_back**(**v**);**

**}**

**if(**bipartito**(**0**)){**

cout**<<**"Es bipartito\n"**;**

**}else{**

cout**<<**"No es bipartito\n"**;**

**}**

**return** 0**;**

**}**

**DIJKSTRA**

#include<bits/stdc++.h>

**using** **namespace** std**;**

vector**<**vector**<**pair**<**int**,**int**>>>**g**;**

vector**<**int**>**dis**;**

void dijkstra**(**int s**){**

//distancia, nodo y ordene por distancia de menor a mayor

priority\_queue**<**pair**<**int**,**int**>,** vector**<**pair**<**int**,**int**>>,** greater**<**pair**<**int**,**int**>>>**q**;**

dis**[**s**]=**0**;**

q**.**push**({**0**,**s**});**

**while(**q**.**size**()){**

//saco el meno en distancia

int u**=**q**.**top**().**second**;**

int w**=**q**.**top**().**first**;**

q**.**pop**();**

//procesa el minimo

**if(**dis**[**u**]!=**w**)continue;**

**for(**auto e**:**g**[**u**]){**

int nodo**=**e**.**first**;**

int nuevopeso**=**w**+**e**.**second**;**

//relajacion

**if(**dis**[**nodo**]>**nuevopeso**){**

dis**[**nodo**]=**nuevopeso**;**

q**.**push**({**nuevopeso**,**nodo**});** //para que siga sacando el menor

**}**

**}**

**}**

**}**

int main**(){**

int nodos**,**aristas**;**

int u**,**v**,**w**;**

cin**>>**nodos**>>**aristas**;**

g**.**assign**(**nodos**+**1**,**vector**<**pair**<**int**,**int**>>(**0**));**

dis**.**assign**(**nodos**+**1**,**INT\_MAX**);**

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

cin**>>**u**>>**v**>>**w**;** //de u a v con peso w

g**[**u**].**push\_back**({**v**,**w**});**

**}**

/\*

for(int i=0;i<g.size();i++){

for(int j=0;j<g[i].size();j++){

cout<<g[i][j].first<<" ";

}

cout<<"\n";

}\*/

int a**;**

a**=**1**;**//desde a a todos los nodos

dijkstra**(**a**);**

**for(**int i**=**1**;**i**<**dis**.**size**();**i**++){**

cout**<<**"distancia minima de "**<<**a**<<**" hasta "**<<**i**<<**" es:"**<<**dis**[**i**]<<**"\n"**;**

**}**

**return** 0**;**

**}**

/\*

5 6

1 3 2

1 2 1

2 3 4

2 4 3

3 5 5

4 5 6

\*/

**FLOYD WARSHALL**

//Dado un grafo halla la distancia mínima entre cualquier par de nodos. g[i][j] guardará la distancia mínima entre el nodo i y el j.

//NODOS MAXIMOS <=500

#include<bits/stdc++.h>

**using** **namespace** std**;**

const int oo**=**10000**;**

int main**(){**

int n**,**m**;**

cin**>>**n**>>**m**;**

vector**<**vector**<**int**>>**min\_dis**(**n**,**vector**<**int**>(**n**,**oo**));**

vector**<**vector**<**int**>>**nxt**(**n**,**vector**<**int**>(**n**,-**1**));**

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

int u**,**v**,**w**;**

cin**>>**u**>>**v**>>**w**;**

min\_dis**[**u**-**1**][**v**-**1**]=**w**;**

nxt**[**u**-**1**][**v**-**1**]=**v**-**1**;**

**}**

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

min\_dis**[**i**][**i**]=**0**;**

nxt**[**i**][**i**]=**i**;**

**}**

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

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

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

**if(**min\_dis**[**i**][**j**]>**min\_dis**[**i**][**k**]+**min\_dis**[**k**][**j**]){**

min\_dis**[**i**][**j**]=**min**(**min\_dis**[**i**][**j**],**min\_dis**[**i**][**k**]+**min\_dis**[**k**][**j**]);**

nxt**[**i**][**j**]=**nxt**[**i**][**k**];**

**}**

**}**

**}**

**}**

cout**<<**"Minima distancia\n"**;**

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

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

cout**<<**min\_dis**[**i**][**j**]<<**" "**;**

**}**

cout**<<**"\n"**;**

**}**

cout**<<**"Camino minimo de from a to\n"**;**

int from**,**to**;**

from**=**4**;**

to**=**3**;**

from**--;**to**--;**

**if(**min\_dis**[**from**][**to**]==**oo**){**

cout**<<**"No hay camino\n"**;**

**}else{**

**while(**from**!=**to**){**

cout**<<**from**+**1**<<**" "**;**

from**=**nxt**[**from**][**to**];**

**}**

cout**<<**from**+**1**<<**"\n"**;**

**}**

**return** 0**;**

**}**

/\*\*

4 5

2 1 4

1 3 -2

2 3 3

3 4 2

4 2 -1

\*/

**ORDEN TOPOLOGICO DAG BFS**

#include<bits/stdc++.h>

**using** **namespace** std**;**

int main**()** **{**

int nodos**,**aristas**,**u**,**v**;**

int entrantes**[**nodos**]={**0**};** //para cada nodo cuantas fechas entran

cin**>>**nodos**>>**aristas**;**

vector**<**vector**<**int**>>**g**(**nodos**);**

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

cin**>>**u**>>**v**;**

g**[**u**].**push\_back**(**v**);**

entrantes**[**v**]++;**

**}**

queue**<**int**>**q**;**

vector**<**int**>**salida**;**

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

**if(**entrantes**[**i**]==**0**){**

q**.**push**(**i**);**

**}**

**}**

**while(**q**.**size**()){**

int s**=**q**.**front**();**

salida**.**push\_back**(**s**);**

q**.**pop**();**

**for(**int i**=**0**;**i**<**g**[**s**].**size**();**i**++){**

int ii**=**g**[**s**][**i**];**

entrantes**[**ii**]--;**

**if(**entrantes**[**ii**]==**0**){**

q**.**push**(**ii**);** //puedo hacer la tarea

**}**

**}**

**}**

cout**<<**"\n"**;**

**for(**int i**=**0**;**i**<**salida**.**size**();**i**++){**

cout**<<**i**<<**" "**;**

**}**

**return** 0**;**

**}**

/\*\*

6 5

0 3

1 3

1 4

2 4

2 5

\*/

**ORDEN TOPOLOGICO DAG DFS**

#include<bits/stdc++.h>

**using** **namespace** std**;**

vector**<**int**>**grafo**[**100**];**

bool vis**[**100**];**

vector**<**int**>**ts**;**

void dfs**(**int u**){**

vis**[**u**]=true;**

**for(**int i**=**0**;**i**<**grafo**[**u**].**size**();**i**++){**

int v**=**grafo**[**u**][**i**];**

**if(!**vis**[**v**]){**

dfs**(**v**);**

**}**

**}**

ts**.**push\_back**(**u**);** //Importante

**}**

void DAG**(){**

**for(**int i**=(**int**)**ts**.**size**()-**1**;**i**>=**0**;**i**--){**

cout**<<**ts**[**i**]<<**" "**;**

**}**

**}**

void init**(){**

memset**(**vis**,false,sizeof(**vis**));**

**for(**int i**=**0**;**i**<**100**;**i**++){**

grafo**[**i**].**clear**();**

**}**

**}**

int main**(){**

init**();**

grafo**[**0**].**push\_back**(**1**);**

grafo**[**0**].**push\_back**(**2**);**

grafo**[**1**].**push\_back**(**3**);**

grafo**[**1**].**push\_back**(**2**);**

grafo**[**2**].**push\_back**(**5**);**

grafo**[**2**].**push\_back**(**3**);**

grafo**[**3**].**push\_back**(**4**);**

//grafo[4].push\_back(1);

grafo**[**6**].**push\_back**(**7**);**

**for(**int i**=**0**;**i**<**8**;**i**++){**

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

dfs**(**i**);**

**}**

**}**

DAG**();**

**return** 0**;**

**}**

**PUNTOS DE ARTICULACION Y PUENTES**

#include<bits/stdc++.h>

**using** **namespace** std**;**

int UNVISITED**=-**1**,**dfsRoot**,**rootChildren**,**dfsNumberCounter**;**

vector**<**int**>**dfs\_num**;**

vector**<**int**>**dfs\_low**;**

vector**<**int**>**g**[**100**];**

vector**<**int**>**dfs\_padre**,**punto\_articulacion**;**

int nodos**,**aristas**;**

void articulationPointAndBridge**(**int u**)** **{**

dfs\_low**[**u**]** **=** dfs\_num**[**u**]** **=** dfsNumberCounter**++;** // dfs\_low[u] <= dfs\_num[u]

**for** **(**int j **=** 0**;** j **<** **(**int**)**g**[**u**].**size**();** j**++)** **{**

int v **=** g**[**u**][**j**];**

**if** **(**dfs\_num**[**v**]** **==** UNVISITED**)** **{**

dfs\_padre**[**v**]** **=** u**;**

**if** **(**u **==** dfsRoot**){**// special case if u is a root

rootChildren**++;**

**}**

articulationPointAndBridge**(**v**);**

**if** **(**dfs\_low**[**v**]** **>=** dfs\_num**[**u**]){**//puntos de articulacion

punto\_articulacion**[**u**]** **=** **true;**

**}**

**if** **(**dfs\_low**[**v**]** **>** dfs\_num**[**u**]){** // para puente

printf**(**" Arista (%d, %d) es un puente\n"**,** u**,** v**);** //Si los quiero los guardo

**}**

dfs\_low**[**u**]** **=** min**(**dfs\_low**[**u**],** dfs\_low**[**v**]);**

**}else** **if** **(**v **!=** dfs\_padre**[**u**]){**

dfs\_low**[**u**]** **=** min**(**dfs\_low**[**u**],** dfs\_num**[**v**]);**

**}**

**}**

**}**

void init**(){**

dfsNumberCounter **=** 0**;**

dfs\_num**.**assign**(**nodos**,** UNVISITED**);**

dfs\_low**.**assign**(**nodos**,** 0**);**

dfs\_padre**.**assign**(**nodos**,** 0**);**

punto\_articulacion**.**assign**(**nodos**,** 0**);**

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

g**[**i**].**clear**();**

**}**

**}**

void solve**(){**

printf**(**"Puentes:\n"**);** //Puentes

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

**if** **(**dfs\_num**[**i**]** **==** UNVISITED**)** **{**

dfsRoot **=** i**;**

rootChildren **=** 0**;**

articulationPointAndBridge**(**i**);**

punto\_articulacion**[**dfsRoot**]** **=** **(**rootChildren **>** 1**);** // special case

**}**

**}**

printf**(**"Puntos de articulacion:\n"**);**

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

**if** **(**punto\_articulacion**[**i**]){**

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

**}**

**}**

**}**

int main**(){**

int u**,**v**;**

cin**>>**nodos**>>**aristas**;**

init**();**

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

cin**>>**u**>>**v**;**

g**[**u**].**push\_back**(**v**);**

g**[**v**].**push\_back**(**u**);**

**}**

solve**();**

**return** 0**;**

**}**

/\*

6 6

0 1

1 2

1 3

1 4

1 5

4 5

\*/