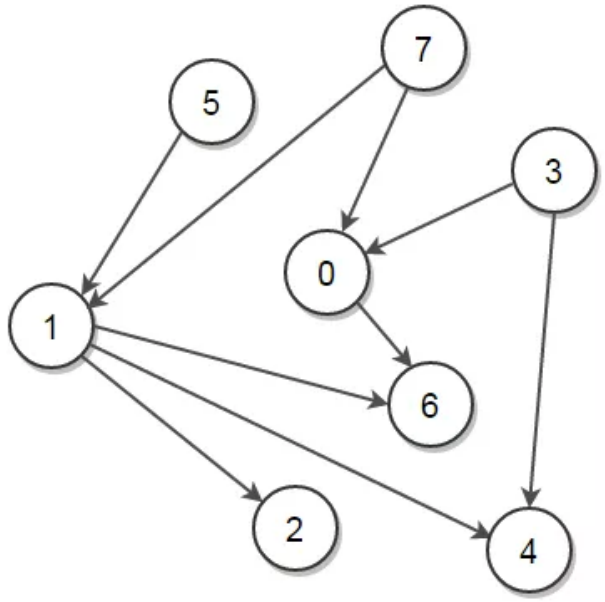
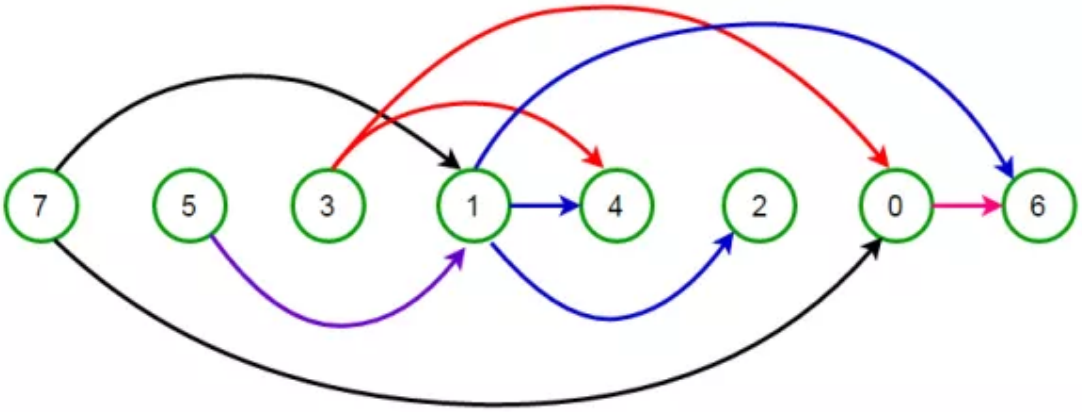
**DFS, BFS - primene**

**1. Dat je usmeren aciklički graf. Odštampati sva topološka uređenja.**




Na primer, mogući topološki redosledi su:
7, 5, 3, 1, 4, 2, 0, 6




Uočimo da za svaku usmerenu granu u -> v, čvor u se u topološkom redosledu pojavljuje pre čvora v (ima nižu poziciju tj. globalno dodeljen broj pri numeraciji čvorova)

ILI

7, 5, 1, 2, 3, 4, 0, 6 ILI

5, 7, 3, 1, 0, 2, 6, 4 ILI

3, 5, 7, 0, 1, 2, 6, 4 ILI

5, 7, 3, 0, 1, 4, 6, 2 ILI

7, 5, 1, 3, 4, 0, 6, 2 ILI

5, 7, 1, 2, 3, 0, 6, 4 ILI

3, 7, 0, 5, 1, 4, 2, 6 ILI

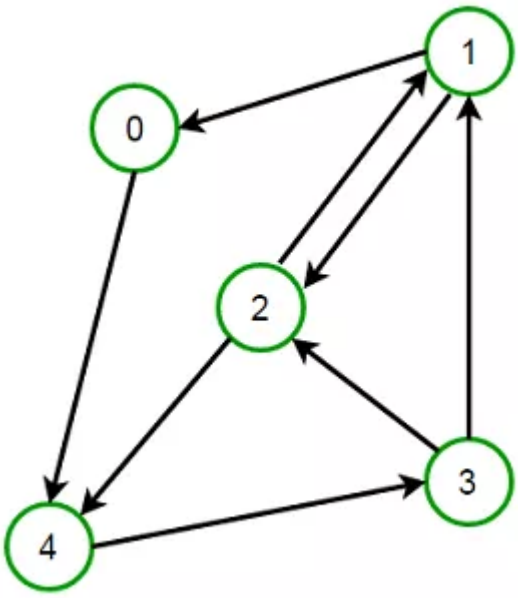
I još mnogo...

Znamo da  pri nalaženju topološkog uređenje u DAG-u možemo da koristimo Depth First Search (DFS)  algoritam, ali smo prikazali i Kahn-ov algoritam

Ideja u ovom zadatku je da koristimo Kahn-ov  Topological Sort tako da tragamo za čvorovima čiji ulazni stepen je 0 i da uklanjamo izlazne grane iz tih čvorova. Kreiramo sva moguća uređenja gde pri (topološkoj) numeraciji čvorova kandidate za naredni korak numeracije biramo među čvorovima koji u tekućem grafu imaju ulazni stepen 0.  Koristimo algoritamsku strategiju pretrage sa vraćanjem (backtracking)

**2.  Implementirati Tarjan-ov algoritam za određivanje komponenti jake povezanosti usmerenog grafa G.  Usmereni graf je jako povezan ako je svaki čvor dostižan iz svih drugih čvorova.**

Primer jako povezanog grafa:




Ideja: Prilikom DFS obilaska datog usmerenog grafa G implicitno se formira DFS drvo, odnosno šuma. Bez narušavanja opštosti možemo pretpostaviti da je graf takav da postoji čvor iz kog se on može u potpunosti obići, odnosno da ima DFS drvo. Nazovimo baznim čvorom neke jake komponente onaj čvor te komponente koji ima najmanju vrednost dolazne numeracije.

Lema 1: Neka je b bazni čvor jake komponente X. Tada za svaki čvor v iz X važi da je v potomak čvora b u odnosu na DFS drvo i svi čvorovi na putu od b do v pripadaju komponenti X.

Lema 2: Neka je b bazni čvor i neka su $b_1$, $b_2$, . . . $b_k$ bazni čvorovi koji su potomci čvora b. Tada važi da je jaka komponenta kojoj pripada čvor b skup svih potomaka čvora b koji nisu potomci nijednog drugog čvora $b_1$, $b_2$, . . . , $b_k$.

Lema 3: Čvor v je bazni čvor akko važi v.Pre = v.minPre.

Da bismo izdvojili čvorove koji pripadaju poddrvetu sa korenom u datom baznom čvoru, možemo iskoristiti stek na koji ćemo stavljati čvor prilikom prve posete tokom DFS obilaska grafa. Kada tokom obilaska naiđemo na čvor koji se već nalazi na steku, znamo da će jednoj komponenti povezanosti pripadati svi čvorovi koji se nalaze na steku počev od tog čvora. Poprečne grane neće biti razmatrane, jer kada stignemo do čvora koji je već posećen, vršimo obradu samo ako se on nalazi na steku (što neće biti slučaj sa krajnjim čvorom poprečne grane).

**3. Implementirati Kosaraju-ov algoritam za proveru da li je usmereni graf G jako povezan. Usmereni graf je jako povezan ako je svaki čvor dostižan iz svih drugih čvorova.**

4.

# Racunarska mreza

| Vremensko ograničenje | Memorijsko ograničenje | ulaz | izlaz |
|---|---|---|---|
| 1 s | 64 MB | standardni ulaz | standardni izlaz |

U računarskoj mreži nalazi se $N$ računara, numerisanih od 0 do $N$-1. Svaki računar, nakon što dobije poruku, prenosi je do drugih računara. Ako poruka od računara $X$ može stići do računara $Y$, to ne mora obavezno da znači da će poruka od računara $Y$ stići do računara X. Sistem administratori žele da izračunaju najmanji broj računara od kojih se mora početi sa slanjem poruke da bi poruka uspela da stigne do svih računara u mreži.
Za bolji prenos poruka, sistem administrator veruju da znate da morate dodati minimalni broj grana tj. veza kako bi osnažili povezanost između računara-komponenti, tako da kada pošaljete poruku od ma kog računara, ona će biti distribuirana do svih drugih računara.
Napišite program koji nalazi dva broja: prvi broj je minimalan broj računara sa kojih se može poslati poruka da bi se distribuirala do svih računara u mreži, a drugi broj je minimalni broj novih veza koje se dodaju kako bi se osnažila povezanost koja dozvoljava da poruka poslata sa bilo kog računara dospe do svakog računara u mreži.
Ulaz
Prvi red standardnog ulaza sadrži dva cela broja $N$ i $M$ (1 < N ≤ 1 600, 0 ≤ M ≤ 120 000), koji predstavljaju broj računara i broj grana među njima. U svakoj od narednih $M$ linija opisana je grana tj. veza koja postoji. Prvi broj u toj vezi predstavlja polazni čvor tj. kompjuter sa kog se šalje poruka, a drugi broj predstavlja računar koji prima poruku.
Izlaz
U jedinom redu standardnog ulaza, Vaš program mora da ispiše dva cela broja – prvi broj je minimalni broj računara koji se mogu koristiti kao polazni računari u topologiji distribuiranja jedne poruke za celu mrežu, dok drugi broj je minimalni broj grana koje se moraju dodati da se proširi kompresovana mreža na način da poruka poslata sa ma kog računara stigne do svih ostalih računara.

Primer
Ulaz
```
6 12
0 1
0 2
1 0
1 2
2 0
2 1
3 4
3 5
4 3
4 5
5 3
5 4
```
Izlaz
```
2 2
```

Rešenje:

```
#include <iostream>
#include <vector>
#include <stack>
#include <set>
using namespace std;
```

```
const int MAXN = 1600;

int n, m;
vector<int> edges[MAXN];
vector<int> rEdges[MAXN];
stack<int> st;
bool visited[MAXN];
int scc[MAXN];
bool inEdges[MAXN];
bool outEdges[MAXN];

void dfs(int u, int parent) {
    visited[u] = true;
    for (int i = 0; i < edges[u].size(); i++) {
        if (visited[edges[u][i]]) continue;
        dfs(edges[u][i], parent);
    }
    st.push(u);
}

void dfsScc(int u, int sccCount) {
    scc[u] = sccCount;
    for (int i = 0; i < rEdges[u].size(); i++) {
        if (scc[rEdges[u][i]]) continue;
        dfsScc(rEdges[u][i], sccCount);
    }
}

int main() {
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int u, v; cin >> u >> v;
        edges[u].push_back(v);
        rEdges[v].push_back(u);
    }
    for (int i = 0; i < n; i++) {
        if (visited[i]) continue;
        dfs(i, i);
    }
    int sccCount = 0;
    while (!st.empty()) {
        if (!scc[st.top()]) dfsScc(st.top(), ++sccCount);
        st.pop();
    }
    for (int u = 0; u < n; u++) {
        for (int v = 0; v < edges[u].size(); v++) {
            if (scc[u] == scc[edges[u][v]]) continue;
            outEdges[scc[u]] = true;
        }
```

```
        for (int v = 0; v < rEdges[u].size(); v++) {
            if (scc[u] == scc[rEdges[u][v]]) continue;
            inEdges[scc[u]] = true;
        }
    }

    int first = 0, last = 0;
    for (int i = 1; i <= sccCount; i++) {
        if (!inEdges[i]) first++;
        if (!outEdges[i]) last++;
    }

    cout << first << " ";
    if (sccCount == 1) first=last=0;
    cout << max(first, last) << endl;

    return 0;
}
```

| # | Status | Poeni | Vreme | Memorija |
|---|---|---|---|---|
| 1 | OK | 1 | 0,02s | 0,00MB |
| 2 | OK | 1 | 0,00s | 0,00MB |
| 3 | OK | 1 | 0,00s | 1,00MB |
| 4 | OK | 1 | 0,00s | 0,00MB |
| 5 | OK | 1 | 0,00s | 1,00MB |
| 6 | OK | 1 | 0,00s | 0,00MB |
| 7 | OK | 1 | 0,00s | 1,00MB |
| 8 | OK | 1 | 0,00s | 0,00MB |
| 9 | OK | 1 | 0,02s | 1,00MB |
| 10 | OK | 1 | 0,02s | 2,00MB |
| 11 | OK | 1 | 0,00s | 0,00MB |
| 12 | OK | 1 | 0,02s | 1,00MB |
| 13 | OK | 1 | 0,00s | 0,00MB |
| 14 | OK | 1 | 0,00s | 1,00MB |
| 15 | OK | 1 | 0,00s | 0,00MB |
| 16 | OK | 1 | 0,05s | 3,00MB |
| 17 | OK | 1 | 0,02s | 2,00MB |
| 18 | OK | 1 | 0,16s | 4,00MB |
| 19 | OK | 1 | 0,09s | 3,00MB |
| 20 | OK | 1 | 0,05s | 3,00MB |

**JAVA rešenje**

| # | Status | Poeni | Vreme | Memorija |
|---|---|---|---|---|
| 1 | OK | 1 | 0,09s | 18,00MB |
| 2 | OK | 1 | 0,09s | 18,00MB |
| 3 | OK | 1 | 0,09s | 17,00MB |
| 4 | OK | 1 | 0,06s | 18,00MB |
| 5 | OK | 1 | 0,11s | 18,00MB |
| 6 | OK | 1 | 0,11s | 17,00MB |
| 7 | OK | 1 | 0,09s | 17,00MB |
| 8 | OK | 1 | 0,08s | 18,00MB |
| 9 | OK | 1 | 0,09s | 17,00MB |
| 10 | OK | 1 | 0,20s | 26,00MB |
| 11 | OK | 1 | 0,08s | 17,00MB |
| 12 | OK | 1 | 0,11s | 17,00MB |
| 13 | OK | 1 | 0,11s | 17,00MB |
| 14 | OK | 1 | 0,09s | 17,00MB |
| 15 | OK | 1 | 0,06s | 16,00MB |
| 16 | OK | 1 | 0,50s | 57,00MB |
| 17 | OK | 1 | 0,20s | 27,00MB |
| 18 | OK | 1 | 0,59s | 95,00MB |
| 19 | OK | 1 | 0,52s | 59,00MB |
| 20 | OK | 1 | 0,55s | 56,00MB |

```
import java.util.ArrayDeque;
import java.util.ArrayList;
```

```
import java.util.Scanner;
import java.util.Stack;

public class cnet {

	static int n, m;
	static ArrayList<Integer>[] edges;
	static ArrayList<Integer>[] rEdges;
	static boolean visited[];
	static int[] scc;
	static boolean[] inEdges;
	static boolean[] outEdges;
	static Stack<Integer> st = new Stack<>();

	static void dfs(int u, int parent) {
		visited[u] = true;
		for (int i = 0; i < edges[u].size(); i++) {
			if (visited[edges[u].get(i)])
				continue;
			dfs(edges[u].get(i), parent);
		}
		st.push(u);
	}

	static void dfsScc(int u, int sccCount) {
		scc[u] = sccCount;
		for (int i = 0; i < rEdges[u].size(); i++) {
			if (scc[rEdges[u].get(i)] > 0)
				continue;
			dfsScc(rEdges[u].get(i), sccCount);
		}
	}

	public static void main(String[] args) {
		Scanner in = new Scanner(System.in);
		n = in.nextInt();
		m = in.nextInt();
		edges = new ArrayList[n];
		rEdges = new ArrayList[n];
		visited = new boolean[n];
		scc = new int[n];
		inEdges = new boolean[n+1];
		outEdges = new boolean[n+1];
		for (int i = 0; i < n; i++)
			edges[i] = new ArrayList<>();
		for (int i = 0; i < n; i++)
			rEdges[i] = new ArrayList<>();

		for (int i = 0; i < m; i++) {
			int u = in.nextInt();
```

```
                int v = in.nextInt();
                edges[u].add(v);
                rEdges[v].add(u);
            }

            for (int i = 0; i < n; i++) {
                if (visited[i])
                    continue;
                dfs(i, i);
            }
            int sccCount = 0;
            while (!st.empty()) {
                if (scc[st.peek()] == 0)
                    dfsScc(st.peek(), ++sccCount);
                st.pop();
            }
            for (int u = 0; u < n; u++) {
                for (int v = 0; v < edges[u].size(); v++) {
                    if (scc[u] == scc[edges[u].get(v)])
                        continue;
                    outEdges[scc[u]] = true;
                }
                for (int v = 0; v < rEdges[u].size(); v++) {
                    if (scc[u] == scc[rEdges[u].get(v)])
                        continue;
                    inEdges[scc[u]] = true;
                }
            }

            int first = 0, last = 0;
            for (int i = 1; i <= sccCount; i++) {
                if (!inEdges[i])
                    first++;
                if (!outEdges[i])
                    last++;
            }

            System.out.print(first + " ");
            if (sccCount == 1)
                first = last = 0;
            System.out.println(Math.max(first, last));
        }

}
```

5.

# A Sky Full of Stars

| Vremensko ograničenje | Memorijsko ograničenje | ulaz | izlaz |
|---|---|---|---|
| 0,6 s | 64 MB | standardni ulaz | standardni izlaz |

Takmičari su dobili N kartica, obeleženih brojevima od 1 do N koje moraju obojiti do kraja takmičarskog dana.
Poznato je da među njima postoji M parova karti koje moraju biti obojene istom bojom. Napisati program koji pronalazi najveći broj broja manjih od N koje se mogu upotrebiti za bojenje svih karti.

## Ulaz

U prvom redu standardnog ulaza dati su celi brojevi N i M ($1 \leq N \leq 10^9, 1 \leq M \leq 10^5$). U narednih M redova su data po dva cela broja – obeležja dve karte, koje moraju da budu iste boje. Među datim parovima, neki parovi se mogu ponoviti.

## Izlaz

Na standarni izlaz ispisati jedan ceo broj, najveći broj boja koje se mogu upotrebiti za bojenje svih karti.

## Primer

### Ulaz

```
12 5
2 1
2 3
1 2
1 3
1 4
```

### Izlaz

```
9
```

Karte 1,2,3,4 se boje istom bojom. Zato je najveći ukupan broj boja 9.

Pravimo graf (čvorovi bili karte, a grane bi postojale između dva čvora ukoliko oni treba da se oboje istom bojom).
Potrebno je da izračunamo broj komponenata povezanosti grafa čiji bi čvorovi bili karte, a grane bi postojale između dva čvora ukoliko oni treba da se oboje istom bojom.

Međutim, eksplicitno predstavljanje grafa je nemoguće zbog potencijalno velikog broja čvorova (10^9), pa se može koristiti drugi pristup. Treba da cuvamo samo cvorove sa granama, a ostale tretiramo zasebno.

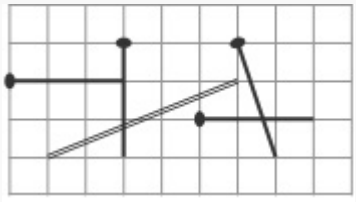
Koristicemo mapu da bi cuvali cvorove i odredicemo broj komponenti povezanosti obicnim dfs-om. Za svaki cvor (koji sadrži granu) pokrenucemo dfs, ako je on neposecen, onda on pripada jednoj komponenti povezanosti, i dfs-om posetimo sve cvorove te komponente. Ako je posecen, znaci da smo vec uracunali komponentu povezanosti kojoj taj cvor pripada, i njega cemo preskociti.

Složenost: O(mlogm) za obradu ulaza, jer koristimo mapu, iza koje je balansirano bst. O(m + |V|) za dfs, gde je |V| broj cvorova iz kojih polaze grane.

**6.**

## Despacito

| Vremensko ograničenje | Memorijsko ograničenje | ulaz | izlaz |
|---|---|---|---|
| 0,5 s | 512 MB | standardni ulaz | standardni izlaz |

 Na datoj slici, dvostrukom linijom je predstavljen provodnik pod naponom. Provodnik se ne sme dodirnuti, kao i svaki metalni element izložen provodniku. Ali, oko provodnika su rasporedjene posebne metalne iglice.

Ulaz

U prvoj liniji standardnog ulaza dat je ukupan broj iglica i provodnika. U svakoj zasebnoj narednoj liniji standardnog ulaza, opisana je iglica preko dva para brojeva: *x* i *y* koordinata krajeva. Poslednja linija standardnog ulaza sadrži koordinate tačaka koje predstavljaju krajeve provodnika. Sve date koordinate su celi brojevi iz segmenta [0, 10000]. Ne postoji više od 1000 iglica.

Izlaz

U jedinoj liniji standardnog izlaza ispisati tačno jedan broj – broj iglica koje se mogu bezbedno dotaći golim rukama (u smislu da te iglice nemaju elektro veze sa provodnikom).

Ulaz

```
5

5 2 8 2
3 4 3 1

7 1 6 4

3 3 0 3

6 3 1 1
```

Izlaz

```
2
```

Rešenje:
Potrebno je naći sve iglice povezane sa provodnikom.
Zato ćemo praviti graf čiji će čvorovi biti iglice i provodnik, a grane između dva čvora pravimo ako se iglice/provodnik koje oni predstavljaju seku.

Da bi uspostavili graf, moracemo da obavimo n^2 koraka, što bi trebalo da bude dovoljno za ovo ograničenje. Moracemo da ispitamo presek svake dve iglice, tj. iglice i provodnika, i povežemo ih ukoliko njihov presek nije prazan. Na kraju cemo u O(n) koraka ispitati presek provodnika sa iglicama jednim prolaskom dfs-a.

Presek dve iglice može predstavljati problem ukoliko su one paralelne, jer ce formula tada zahtevati da delimo sa 0, pa cemo morati da razmotrimo paralelnost pre traženja preseka.

```
#include <iostream>
using namespace std;

struct Point
{ int x, y; };

const int NMAX = 1024;

Point A[NMAX], B[NMAX];
int n;

bool adj[NMAX][NMAX];
bool visited[NMAX];

int compSize;

int direction(Point A, Point B, Point C)
{ int a1=B.x-A.x, a2=B.y-A.y;
  int b1=C.x-A.x, b2=C.y-A.y;
  int p=a1*b2, q=a2*b1;
  if(p>q) return +1;
  if(p<q) return -1;
  return 0;
}

bool onSegment(Point A,Point B,Point C)
{
  return min(A.x,B.x)<=C.x && C.x<=max(A.x,B.x) &&   min(A.y,B.y)<=C.y &&
C.y<=max(A.y,B.y);
}

bool intersect(Point A, Point B, Point C, Point D)
{ int d1 = direction(A,B,C);
  int d2 = direction(A,B,D);
  int d3 = direction(C,D,A);
  int d4 = direction(C,D,B);
  if (d1*d2<0 && d3*d4<0)return true;
  if (d1==0 && onSegment(A,B,C)) return true;
  if (d2==0 && onSegment(A,B,D)) return true;
  if (d3==0 && onSegment(C,D,A)) return true;
  if (d4==0 && onSegment(C,D,B)) return true;
```

```
  return false;
}

void dfs(int i)
{
   compSize++;
  visited[i]=true;
  for(int j=0; j<n; j++)
    if(adj[i][j] & !visited[j]) dfs(j);
}

int main()
{
   int i;
  cin >> n;
  for(i=0;i<n;i++)
    cin >> A[i].x >> A[i].y >> B[i].x >> B[i].y;


  for(int i=0; i<n; i++)
    for(int j=i+1; j<n; j++)
          adj[i][j] = adj[j][i] = intersect(A[i],B[i],A[j],B[j]);

  compSize=0;
  dfs(n-1);

  cout << n-compSize << endl;

  return 0;
}
```

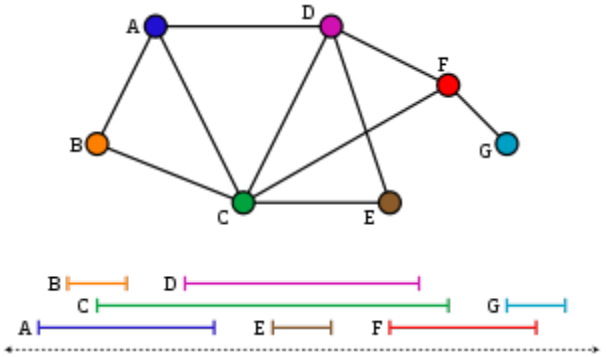
7.

## Tango to Evora

| Vremensko ograničenje | Memorijsko ograničenje | ulaz | izlaz |
|---|---|---|---|
| 1 s | 128 MB | standardni ulaz | standardni izlaz |

Problem D Interval Graph

Interval graf – uvod




Ideja 1: Booklet

Ideja 2: Prvo primetimo da listovi u našem drvetu nemaju velikog uticaja na samo drvo.
Listove možemo smatrati kao intervale koji sadrže samo jednu tacku.
Da bi došli do punog zakljucka, treba da uocimo odredjena pravila.
Da bi intervali imali zajednicke tacke, postoje cetiri moguca odnosa izmedu intervala.
Ili se nalaze jedan unutar drugog (2 slucaja), ili se delimicno preklapaju (2 slucaja).
Primetimo da za cvor koji nije list, tj. ima potomke, i nije koren, tj. ima pretka,
prva dva slucaja ne dolaze u obzir za bilo koji od odnosa sa cvorovima sa kojima je povezan, jer bi
oba slucaja implicirala povezanost potomaka i pretka, što je kontradikcija.
TROUGAO, tj. CIKLUS!!!

Moguc je specifican slucaj kada cvor obuhvata sve potomke i pretka.
Medjutim, u tom slucaju bi potomci morali da budu listovi, a predak koren,
te cemo ovaj slucaj lako pokriti uklanjanjem listova.

Dakle, za cvor koji nije ni list, ni koren, preostaju dva slucaja:
svaki od cvorova povezanih sa njim je interval koji ga delimicno preklapa ili levo ili desno.

Kako vec znamo da ce se preklapati sa jedne strane sa svojim pretkom,
ostaje samo još jedan cvor sa kojim ce on moci da ima zajednicke tacke.
Dakle, dobili smo uslov: svaki cvor koji nije koren i nije list,
mora da ima najviše jednog potomka koji nije list.

Što se samog korena tice, on ce morati da ima najviše
dva potomka koji nisu listovi,
i ukoliko ima jednog potomka, moci cemo da prenesemo svojstvo korena na njegovog potomka,
tj. njegov potomak ce moci da ima najviše dva potomka koji nisu listovi.
Ovo možemo vizuelizovati kao odnos u kome je koren interval
koji potpuno pripada intervalu koji je njegov potomak,
a potomci njegovog potomka su intervali koji delimicno preklapaju po jednu
od strana tog intervala.
Pošto su nam eksplicitno dati potomci svakog cvora,
ne moramo da radimo dfs kako bi pronašli listove,

pa cemo samo bfs-om proveriti da li svi preostali cvorovi ispunjavaju uslov

Složenost: O(n) za bfs

## 8.
## Breaking The Habit

| Vremensko ograničenje | Memorijsko ograničenje | ulaz | izlaz |
|---|---|---|---|
| 1 s | 1000 MB | standardni ulaz | standardni izlaz |

Bora je otet na jednoj raskrsnici u njemu nepoznatom gradu. Otmičaru su ga strpali u auto i voze ga na neko tajno mesto. Oni znaju da Bora ne poznaje taj grad, pa mu nisu zavezali oči, a Bora je odlučio da zapamti kojim putem ga voze na sledeći način: Grad se sastoji od pravougaone mreže oblakodera. Između svaka dva susedna oblakodera nalazi se ulica. Ulice su numerisane brojevima, počevši od broja 1, odozgo prema dole i sa leva na desno. Kada ga otmičari voze po nekoj ulici, Bora u tajnosti zapiše u mobilni telefon visinu oblakodera sa leve i visinu oblakodera sa desne strane ulice. Nakon što auto pređe preko neke raskrsnice, Bora ponovo zapamti visinu sa leve i visinu sa desne strane. Na svakoj raskrsnici auto može nastaviti pravo, skrenuti levo, skrenuti desno, ili se polukružno okrenuti (i nastaviti ulicom odakle je došao). Kada su ga otmičari doveli na cilj, Bora je uspeo da pošalje policiji poruku sa podacima o visinama koje je zapisao.

Napisati program koji će pomoći policiji da otkrije koordinate raskrsnice na koju su otmičari odveli Boru.

Ulaz

U prvom redu ulaza se nalaze dva cela broja R i K, koji predstavljaju broj redova i broj kolona gradske mreže.

U svakom od sledećih R redova se nalazi po K celih brojeva, visine oblakodera redom.

U sledećem redu nalazi se celi broj N, dužina puta kojim se vozio Bora.

U sledećem redu nalazi se N brojeva, visine oblakodera koje je Bora video sa leve strane.

U sledećem redu nalazi se N brojeva, visine oblakodera koje je Bora video sa desne strane.

Izlaz

U prvom i jedinom redu izlaza treba ispisati koordinate (prvo red pa kolonu) raskrsnice gde su otmičari odveli Boru.

Ako postoji više rešenja, ispisati bilo koje.

Ograničenja

$3 \leq R \leq 100$

$3 \leq K \leq 100$

$1 \leq N \leq 10,000$

Visina svakog oblakodera je veća ili jednaka od 1, a manja ili jednaka od 10,000.

Ulaz

```
4 4
7 3 5 4
2 8 9 7
3 5 2 8
1 3 5 7
5
5 9 8 2 9
2 2 2 8 7
```

Izlaz

```
1 3
```

**Ideja 1:**
Za svako stanje na putu BFS-om proverimo gde sve mozemo stici (levo, pravo, desno, polukruzno). Dakle, to bi bilo N*R*K provera, ali kako je vrednost 10000*100*100 na relativni na granici zbog vremenskog ogranicenja, onda koristimo Queue u kojm pamtimo gde sve mozemo biti u odnosu na tekuci poziv i na taj nacin ne proveravamo sve lokacije.

Slozenost algoritma je O(broj_pomeraja*n*m). Najgori slučaj je ako su visine svih zgrada jednake.

**Ideja 2:**
BFS uz ograničenje da dozvoljavamo posećivanje čvorova ako smo ih ranije posetili (ne u trenutnoj iteraciji).
Na početku generišemo sve moguce čvorove do kojih smo mogli da stignemo sa prvom levom i desnom visinom, a onda nastavljamo od tih čvorova na one čvorove koje možemo. Kako u jednoj iteraciji nećemo posetiti nijedan čvor više od jednom, a postoji n iteracija, složenost je O(nrk).

**9.**

| Vremensko ograničenje | Memorijsko ograničenje | ulaz | izlaz |
|---|---|---|---|
| 60 s | 1000 MB | standardni ulaz | standardni izlaz |

Dat je lavirint širine A i visine B. Odrediti da li postoji izlaz iz lavirinta počevši od datih koordinata X i Y. Koordinate 0,0 predstavljaju gornje levo polje lavirinta.

Ulaz

U prvom redu standardnog ulaza nalaze se vrednosti A i B odvojene razmakom. U drugom redu standardnog ulaza nalaze se koordinate X i Y odvojene razmakom. Topologija lavirinta je data u sledećih B redova koji sadrže po A karaktera od kojih svaki predstavlja po jedno polje lavirinta. Karakter 'X' predstavlja zid, a razmak (' ') predstavlja prolaz.

Izlaz

U prvom redu standardnog izlaza treba da se nalazi reč "DA" ako postoji izlaz počevši od polja (X, Y) ili reč "NE" ako izlaz ne postoji.

Ograničenja

1 <= A,B <= 256
0 <= X < A
0 <= Y < B

ex

Ulaz

```
10 5
1 1
 XXX  XXX
X   XX   X
XXX X  X X
X     XX
XXXXXXX XX
```

Izlaz
DA

Ideja 1: Backtracking (vremensko ograničenje 60 s)

Ideja 2:
BFS
Idemo po slobodnim poljima dokle možemo. Ako u jednom trenutku nemamo više neposećenih polja koji su sledeci po bfs udaljenosti, onda nema rešenja.
U suprotnom, ako dođemo do praznog polja na obodu lavirinta, onda možemo napustiti lavirint. možemo direktno proveravati da li je grana na obodu, ili možemo ograditi lavirint karakterima Y, i zaustaviti algoritam kad posetimo polje Y, koje je van lavirinta.
Složenost: O(a * b)

**10. Konstruisati algoritam koji za neusmereni graf G proverava da li ima Ojlerov ciklus, Ojlerov put ili nijedno od ta dva.**

**11. Konstruisati algoritam koji za dati usmereni graf G utvrdjuje da li sadrži Ojlerov ciklus.**

**12. Implementirati Fleury-ev algoritam za odredjivanje Ojlerovog ciklusa/puta u neusmerenom grafu G.**

**13. Implementirati Hierholzer-ov algoritam za pronalaženje Ojlerovog ciklusa u usmerenom grafu G.**

**14. Neka je dat n reči. Utvrditi da li je moguće ulančati sve te reči tako što se kraj jedne od reči nadovezuje na početak naredne. Cilj je vratiti se u reč iz koje smo krenuli.**
**Primer: abba, aabb, bba se mogu ulančati kao abba → aabb → bba, dok se reči abb, bcd, dce ne mogu ulančati.**

**15. Konstruisati algoritam koji u neusmerenom grafu G pronalazi sve Hamiltonove puteve.**

**16. Konstruisati algoritam koji u usmerenom acikličkom grafu G pronalazi Hamiltonov put.**