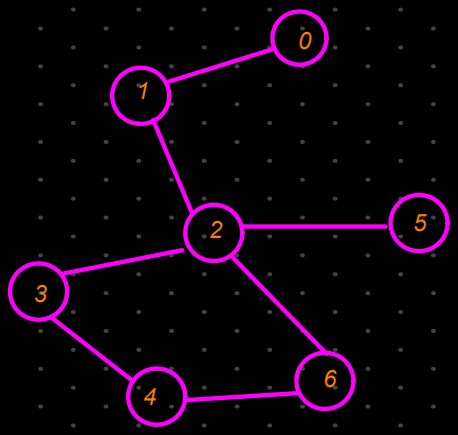
Graph Traversal

Depth First Search (DFS)



لو انا عايز اعدي علي كل الnodes هاخد كل نود واعدي علي ال neighbors بتوعها وكل نود هعدي عليها هعلم عليها

```
Traverse (u){
    if node u is visited
        return
    mark u as visited
    for each node v in neighbors of u {
        Traverse(v)
    }
}
```

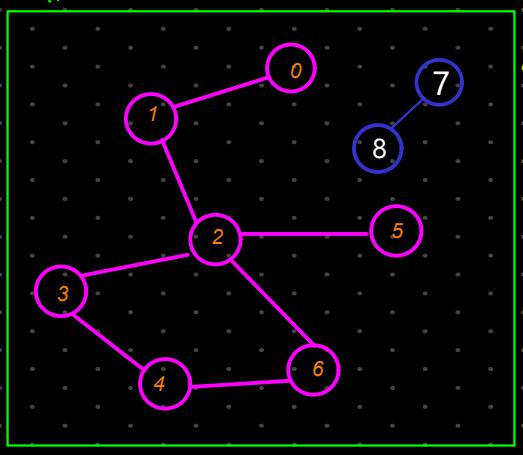
algorithm not code !!

Code

- انسب طريقه لتخزين الجراف هي ال adjlist عشان انا عايز ال neighbors

```
const int N = 1e5 +5 ;
vector<int> adjlist[N];
bool visited [N];

void DFS(int u) {
    if(visited[u])
        return;
    visited[u] = true;
    for (int v : adjlist[u])
        DFS(v);
}

int main() {
    int n,u,v ;cin>>n;
    for( int i=0 ; i < n ; ++i) {
        cin>>u>>v;
        adjlist[u].push_back(v);
        adjlist[v].push_back(u);
    }
    DFS( u: 0);
}
```

بس الكود اللي قدامنا في حته صغيره وهي ان ممكن يكون الجراف كدا



Dis Connected Graph

دلوقت انت بدأت ال DFS بتاعك عند 0 هيعدي علي البنفسجي كله وملوش علاقه بالازرق والحل هنا تعديل صغير علي الكود

```
int main() {
    int n,u,v ;cin>>n;
    for( int i=0 ; i < n ; ++i) {
        cin>>u>>v;
        adjlist[u].push_back(v);
        adjlist[v].push_back(u);
    }

    for (int i=0 ; i < n ; ++i ) {
        if(!visited[i])
            DFS(u);
    }
}
```

طيب لو انا مديك disconnected graph وبسألك اي عدد ال connected componenet

هنا هعدل بس علي اخر كود اني هعمل counter يزيد واحد كل مره هنادي ال DFS

```
for (int i=0 ; i < n ; ++i ) {
    if(!visited[i]){
        ++cnt;
        DFS(u);
    }
}
```

Q: هيديك جراف الجراف دا هيديك فيه عدد النودز وال edges وعايز يعرف هل هو tree ول لا

SOL: شروط ال tree كانوا 3 لو حققت 2 منهم التالت يتحقق لوحده انا لسا مخدتش ازاي اعرف ال graph هو acyclic ولا لا فهقدر احسب ال edges بتساوي النودز ناقص واحد ولا لا و احسب ال connected component

---

Q: هيديك جراف وعايز يعرف هل هو في cycle ول لا

SOL: هتروح تقف عند كل نود وتشوف ال neighbors وتعرف ال parent اللي هو منادي عليك اصلا وتشوف هتلاقي 4 احتمالات :-

النود اللي عليها الدور:-

visited & parent — مش هعمل حاجه

visited & not parent — cyclic كدا هي

not visited & not parent — هروحلها

not visited & parent — مستحيل تحصل

```cpp
bool isCyclic(int u, int p) {
    vis[u] = true;
    for (int v : adj[u]) {
        if (vis[v] && v != p)   return true;
        else if (!vis[v]) {
            if (isCyclic(v, u)) return true;
        }
    }
    return false;
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 0; i < m; i++) {
        scanf("%d %d", &u, &v);
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
    isCyclic(0, -1);
    return 0;
}
```

الكود دا تمام وشغال علي ال undirected لكن
لما نروح لل undirected الشغل هيختلف

الطريقه اللي همشي بيها اني هميز النودز عندي الي 3 حاجات visited & inprogress &notvisited

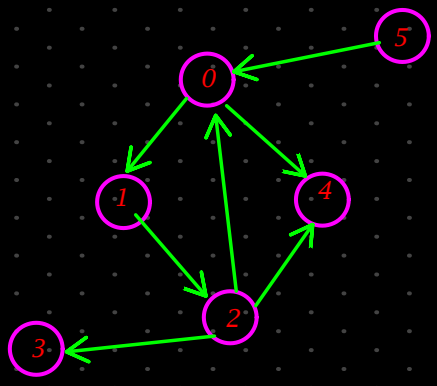
ال visited اللي انا زورتها ورادع منها
ال inprogress هي اللي زورتها خلال رحلتي ولسا مرجعتش
ال notvisited هي ال انا لسا مقربتلهاش

طب هعرف منين هي cyclic ولا لا لما ارجع لنود وتطلع inprogress يبقي cyclic

المسأله دي بيبقي اسمها Is DAG (Directed Acyclic Graph)

```cpp
const int N = 1e5 + 5, M = 2e5 + 5;

const int NOT_VISITED = 0, IN_PROGRESS = 1, VISITED = 2;

int n, m, u, v;

vector<int> adj[N];
int vis[N];
bool isCyclic(int u) {
    vis[u] = IN_PROGRESS;
    for (int v : adj[u]) {
        if (vis[v] == NOT_VISITED) {
            if (isCyclic(v))  return true;
        }
        if (vis[v] == IN_PROGRESS) {
            return true;
        }
    }
    vis[u] = VISITED;
    return false;
}
```

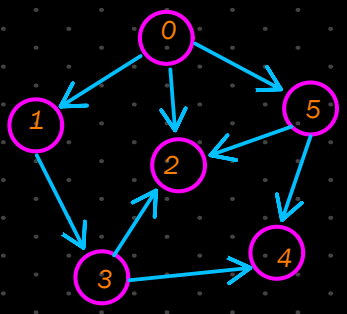
بس الفكره فالمسأله هنا هنبدأ منين عشان انا لو بدأت من
واحده برا ال cycle هيقولي ان مفيش cycle فلازم
النود دي اللي هبدأ منها تكون جوا ال cycle
عشان كدا الحل ابدأ من عند اي نود تكون not visited



هنا لو انا بدأت من ال4 او 3 هيقولي ان مفيش cycle مع ان في من 0 2 1 0

```cpp
int main() {
    scanf("%d %d", &n, &m);
    for (int i = 0; i < m; i++) {
        scanf("%d %d", &u, &v);
        adj[u].push_back(v);
    }
    for (int u = 0; u < n; u++) {
        if (vis[u] == NOT_VISITED) {
            if (isCyclic(u)) {
                puts("Cyclic");
                return 0;
            }
        }
    }
    puts("Acyclic");
    return 0;
}
```

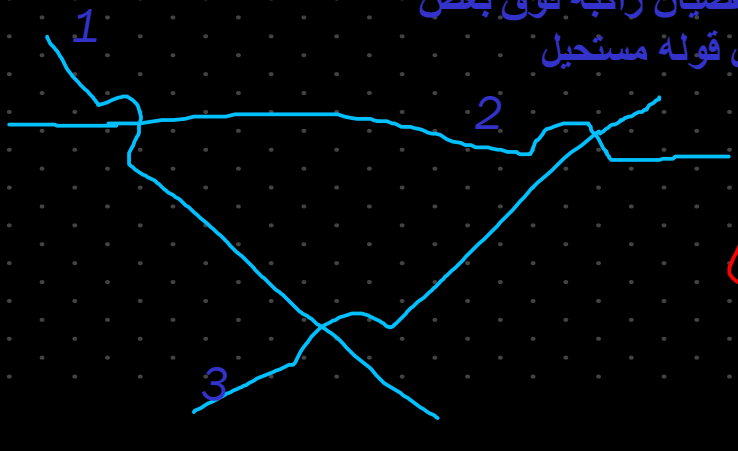
Q:each course have prerequirements I will give you each course ande its prerequirements print the good ordering of the courses



هعدي عل كل كورس اشوف الجيران بتوعه وكل كورس اخلص الجيران بتوعه اطبعه والكورس بردو اللي ملوش جيران يطبع *DFS* عادي خالص بسالفكره هنا من عند البدايه مينفعش ابدأ من عند اي نقطه لكن هبدأ من عند اي حاجه *not visited* واتاكد ان كله بقا *visited*

```
const int N = 1e5 + 5, M = 2e5 + 5;

int n, m, u, v;

vector<int> adj[N];
bool vis[N];

void DFS(int u) {
    vis[u] = true;
    for (int v : adj[u]) {
        if (!vis[v]) {
            DFS(v);
        }
    }
    printf("%d\n", u);
}

int main() {
    scanf("%d %d", &n, &m);
    for (int i = 0; i < m; i++) {
        scanf("%d %d", &u, &v);
        adj[u].push_back(v);
    }

    for (int u = 0; u < n; u++) {
        if (!vis[u]) {
            DFS(u);
        }
    }
    return 0;
}
```

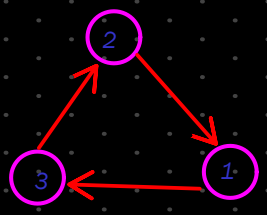
في مسأله حلوه يعتبر نفس فكرة اللي فاتت هيديك شوية عصيان راكبه فوق بعض واختار الترتيب اللي اشيل بيع العصيان صح ولو مفيش حل قوله مستحيل
هيديك عند كل نقطة تقاطع مين فوق ومين تحت



مستحيل

اولا كدا عرفت منين انها مستحيل

لما رسمت الجراف لقيته cyclic



3 تتشال قبل 2
2 يتشال قبل 1
1 تتشال قبل 3

يبقي الحل نفس الكود بتاع المسأله اللي قبله بس اشوف الاول هي cyclic ولا لا

Time Complexity of the DFS

O(n+m)

n : number of nodes
m: number of edges

n<<<<<m

O(m)

n>>>>>>m

O(n)