# DAY 5

## TOPOLOGICAL SORT &
## STRONGLY CONNECTED COMPONENTS

LeViAtHaN

# เฉลย ROYAL PARADE

คล้าย zombie land

1. หา node ที่ผ่านทาง "node ที่อยู่ระหว่างเส้นทางจาก S->E ที่สั้นที่สุด"
    a. ถ้าเรา DFS สองรอบจากทั้ง S และ E ระยะทางทั้งสองค่าจะบวกกันได้ระยะทางจาก S->E พอดี
2. DFS จาก ST -> ED โดยไม่ผ่าน node ที่ขบวนเสด็จผ่าน

# เฉลยการบ้าน น้ำยาเพิ่มความเร็ว

เก็บใน dp[80005][9] (ดื่มไม่เกิน 8 ขวด)

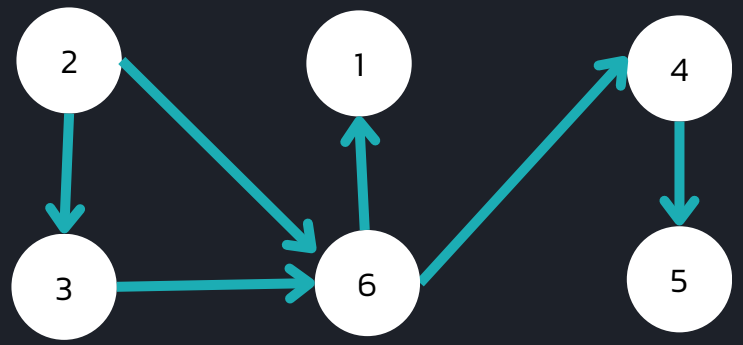
ตอนเดินวน ใช้ u,w,use,pa; เรียง w น้อยไปมาก
หากตกห้องที่มี drink และ v!=pa และ use<Q -> ดื่มได้และไปหาห้องต่อไป หรือ เลือกไม่ดื่มก็ได้

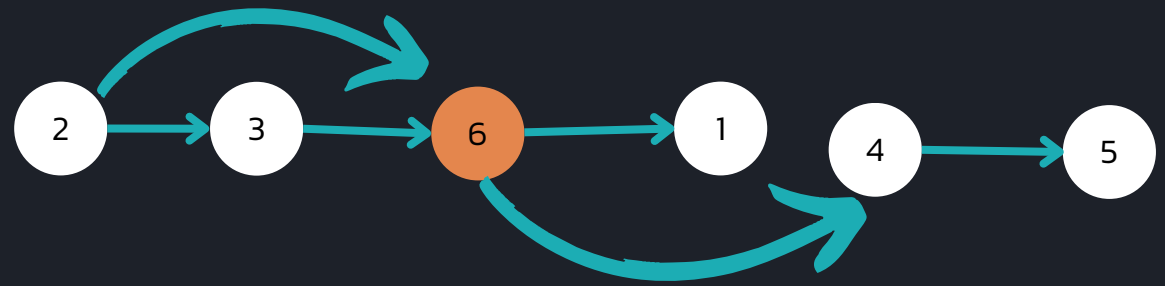
# TOPOLOGICAL SORT

LeViAtHaN

# TOPOLOGICAL SORT คืออะไร

การมอง An acyclic graph ให้อยู่ในลำดับที่ node ใดๆจะไปถึงได้ หาก node ก่อนหน้า visited หมดแล้ว




จะ visit node 6 ได้ จะต้อง visit 2 และ 3 มาก่อน

**Tips:**

- หากเจอ cycle ในกราฟ = ไม่เป็น topological เพราะไม่มีจุดเริ่ม
- ใช้การนับจำนวน node ที่ชี้เข้า (k ครั้ง) หากเรา DFS มาเจอ node u ทั้งหมด k ครั้ง
  แปลว่า node ที่ชี้เข้าทั้งหมดถูกเยี่ยมมาหมดแล้ว

LeViAtHaN

# CSES Course Schedule

## [15 min]

LeViAtHaN

# เฉลย CSES COURSE SCHEDULE

ใช้ topological sort ตรงๆเลย
เราเก็บว่า node มีตัวชี้เข้ากี่ตัว แล้วก็วน i จาก [0,N-1] ไป DFS คอร์สที่ไม่มีตัวชี้เข้าแล้ว
ในคอร์สที่ลงเรียนแล้ว ให้ไปลบตัวชี้เข้าของคอร์สที่ต่อจากคอร์สนี้ แล้ว DFS ไปเรื่อยๆ

# CSES Longest Flight Route

## [15 min]

LeViAtHaN

# เฉลย CSES LONGEST FLIGHT ROUTE

หาทางจาก 1 ไป N ที่ผ่าน node ที่มากสุด (ไม่ใช่ระยะทางยาวที่สุด)

task 1 : ใช้ DPF โดยใช้ if เป็น (dp[v]<dp[u]+1) นับเมืองที่ต้องผ่านแทน
task 2 : หาทางไปเมือง N ดังนั้น ระหว่าง DFS ต้องเก็บ pa ไปด้วย
pa ตัวสุดท้ายที่เก็บจะเป็น pa ที่ผ่านเมืองมากสุด เราก็แค่ย้อน pa พอ

# USACO Timeline

## [20 min]

LeViAtHaN

# เฉลย USACO TIMELINE

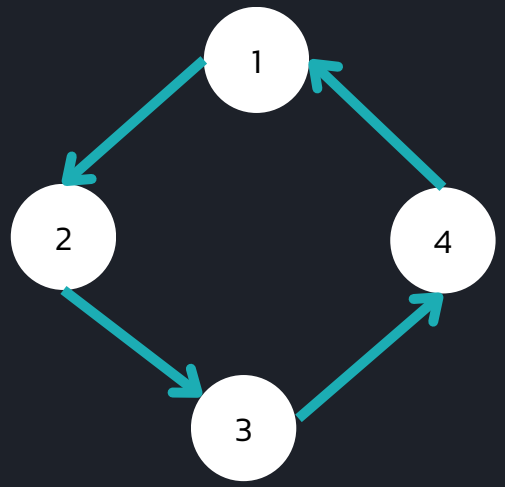
ใช้ topo ผสม DP

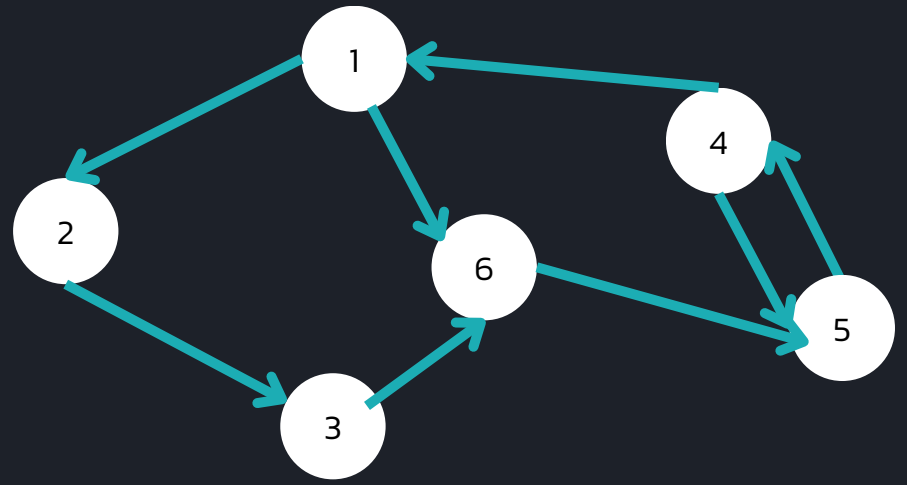
point : เราจะยังไม่เข้าไปใน node ใด หากเรายังไม่รู้วันที่แน่นอนของ node ก่อนหน้านั้น ดังนั้นเราจะเริ่มจาก node ที่ไม่มีตัวเทียบว่ามีเหตุการณ์ไหนเกิดก่อน แล้วค่อย DFS ไปเรื่อยๆ จาก node u ไป node v ให้เราหา dp[v] = max(dp[v],dp[u]+w)

# STRONGLY CONNECTED COMPONENTS

LeViAtHaN

# STRONGLY CONNECTED COMPONENTS คืออะไร



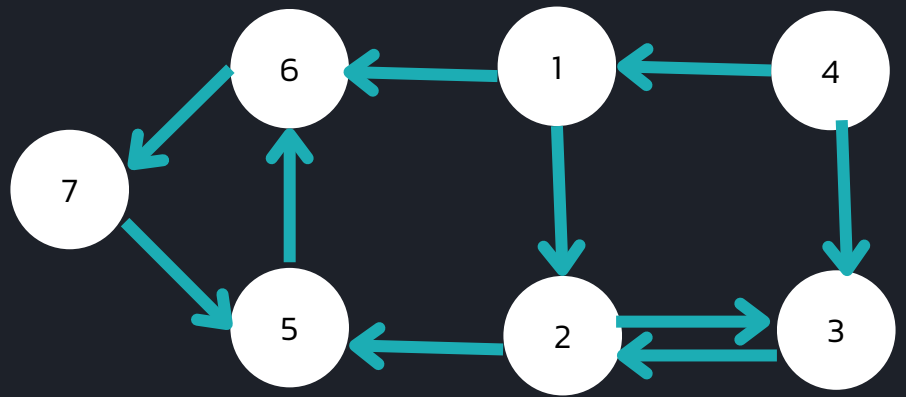




**Strongly Connected**

a directed graph ที่ node ทุก node สามารถไปหากันได้

LeViAtHaN

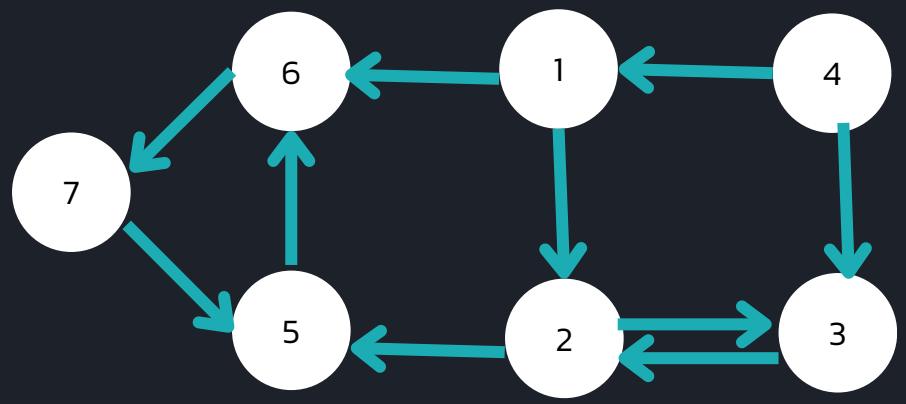
# STRONGLY CONNECTED COMPONENTS คืออะไร

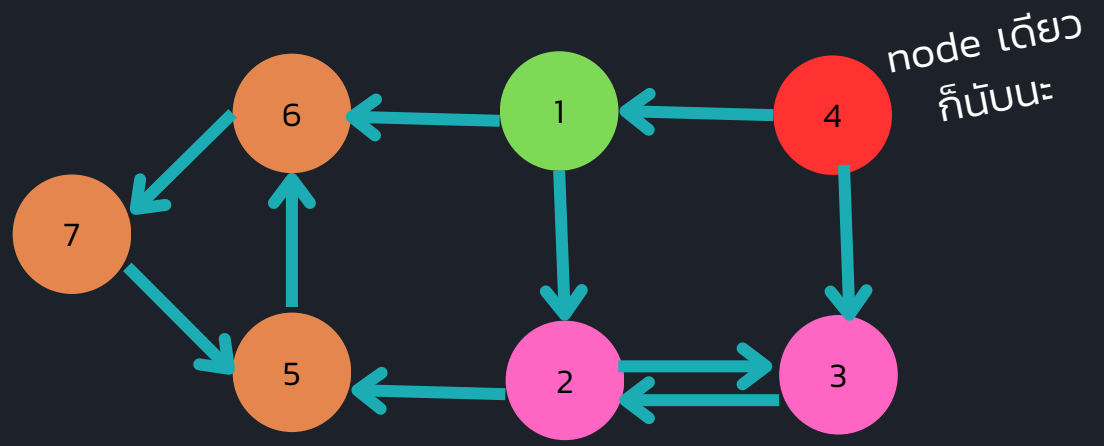



Strongly Connected Components

บางส่วนของ directed graph ที่ใหญ่ที่สุด ที่ node ทุก node สามารถไปหากันได้

# STRONGLY CONNECTED COMPONENTS คืออะไร



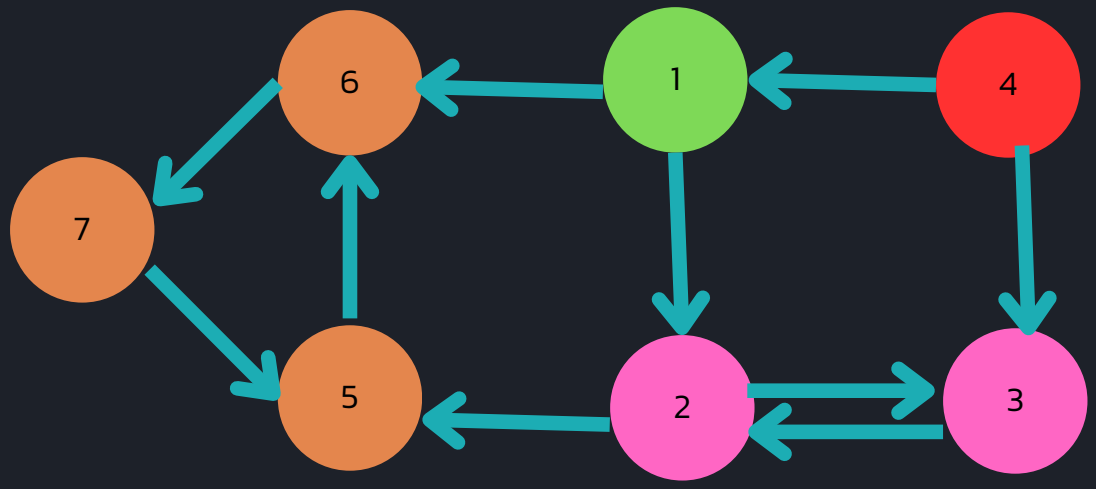

Strongly Connected Components

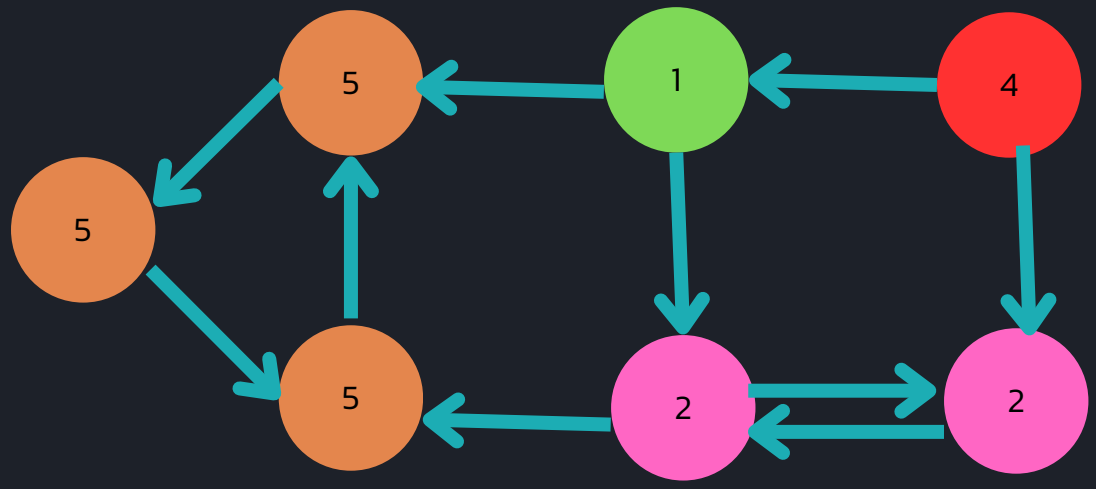
บางส่วนของ directed graph ที่ใหญ่ที่สุด ที่ node ทุก node สามารถไปหากันได้

LeViAtHaN

# LOW LINK คืออะไร

low link ของ u : node v ที่มีค่าน้อยที่สุดที่ node u จะไปถึงได้
มี low link เดียวกัน = อยู่ใน SCC เดียวกัน



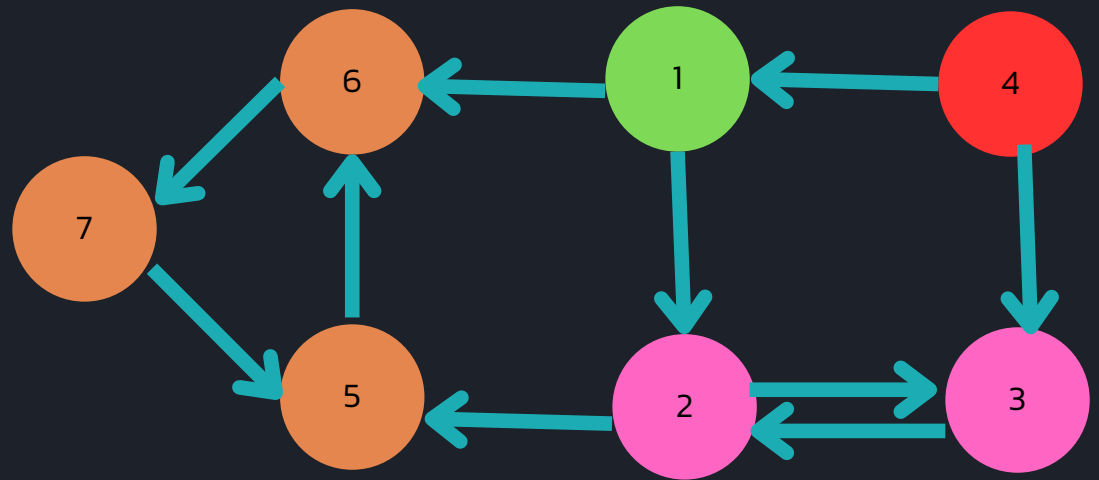




LeViAtHaN

# STRONGLY CONNECTED COMPONENTS คืออะไร

โจทย์ Strongly Connected Components ที่พบได้บ่อย คือ

- หาจำนวน SCC ในกราฟ
- หา node ตัวแทน SCC แต่ละอัน (lowlink)
- หาขนาดSCC ที่ใหญ่หรือเล็กที่สุด ในกราฟ




LeViAtHaN

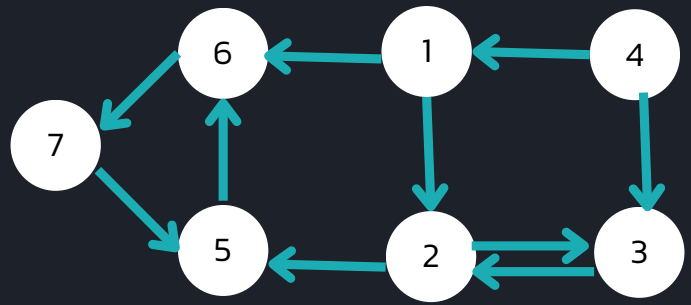
# SOL 1 : KOSARAJU'S O(N+M)

DFS สองรอบ เพื่อหา SCC ใน topological order

ไม่มี node อื่นให้ visit = ไม่มี path เชื่อมไป node อื่น / node รอบๆถูก visit ไปแล้ว

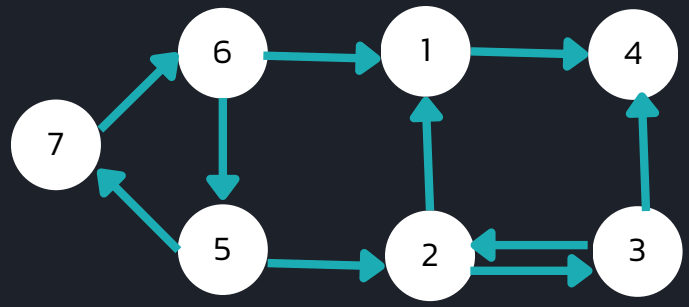
**DFS รอบแรก** : DFS จนเจอตัวที่ไม่มี node อื่นให้ visit แล้วเก็บตัวนั้นลงใน stack

**DFS รองสอง** : กลับกราฟ แล้วเอาตัวจาก stack มา DFS จนไม่มี node อื่นให้ visit




vector <int> path1[N]

ปกติ




vector <int> path2[N]

กลับกราฟ

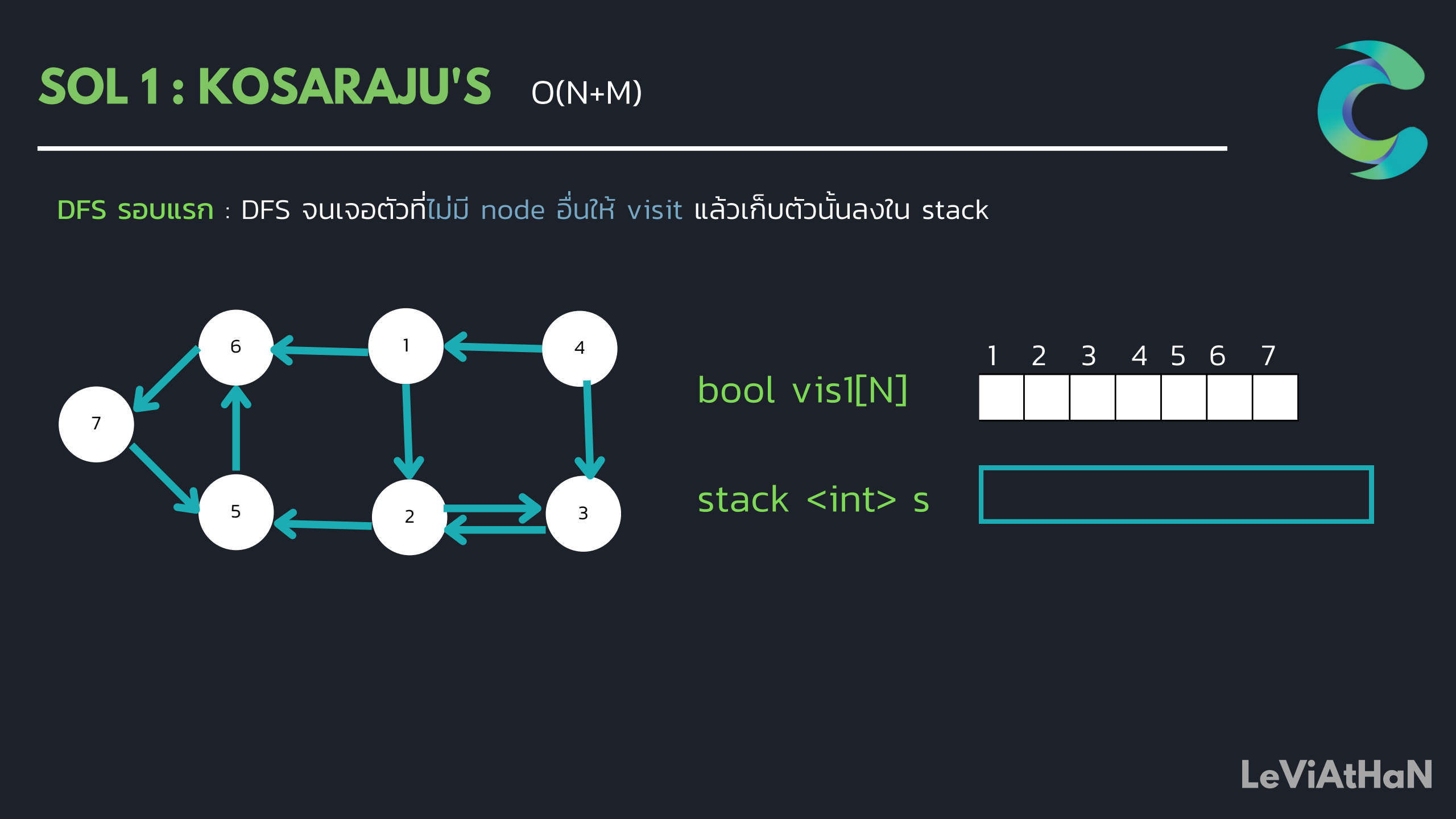
bool vis1[N]

bool vis2[N]

stack <int> s

LeViAtHaN

# SOL 1 : KOSARAJU'S O(N+M)



DFS รอบแรก : DFS จนเจอตัวที่ไม่มี node อื่นให้ visit แล้วเก็บตัวนั้นลงใน stack




bool vis1[N]

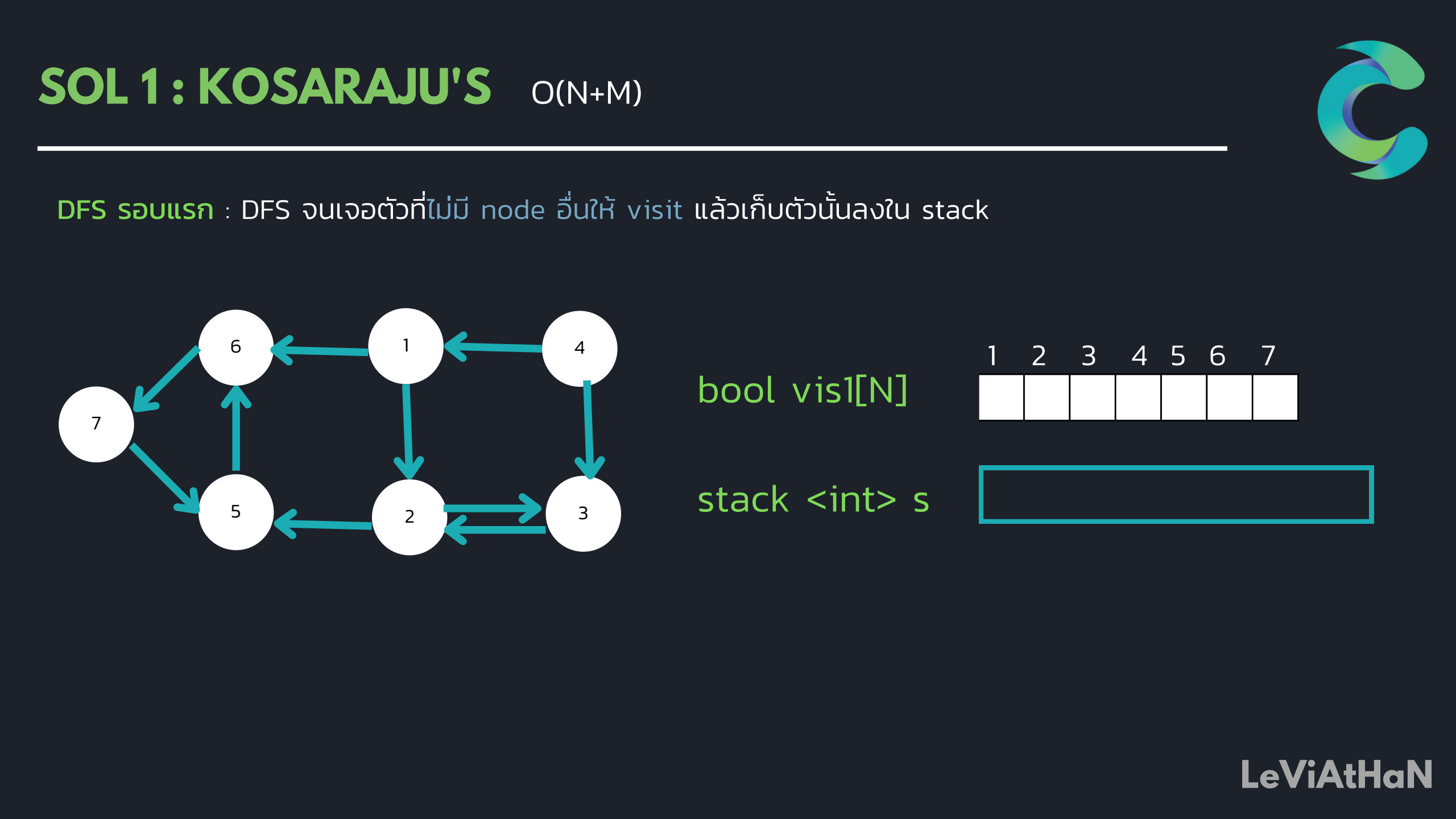



stack <int> s



LeViAtHaN

# SOL 1 : KOSARAJU'S O(N+M)



DFS รอบแรก : DFS จนเจอตัวที่ไม่มี node อื่นให้ visit แล้วเก็บตัวนั้นลงใน stack




bool vis1[N]



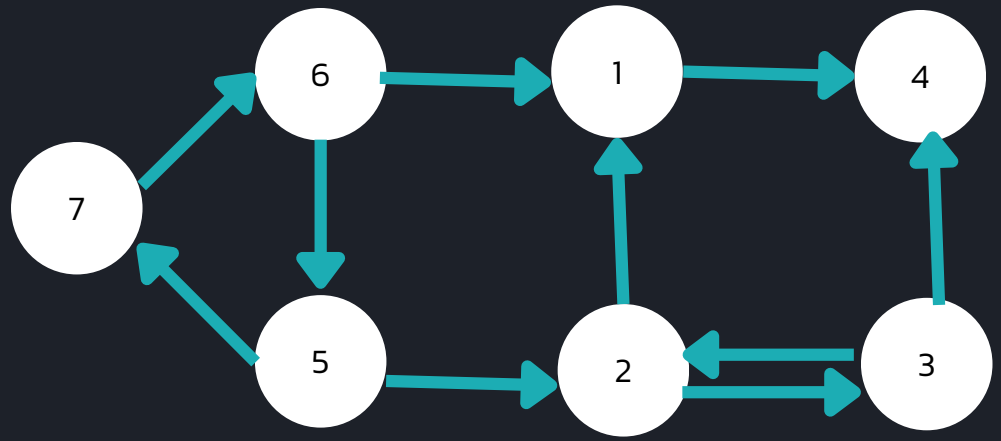

stack <int> s



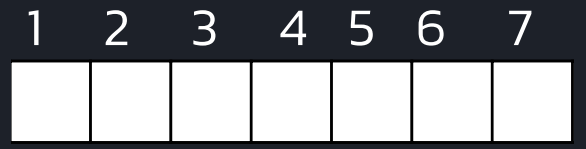
LeViAtHaN

# SOL 1 : KOSARAJU'S O(N+M)

DFS รองสอง : กลับกราฟ แล้วเอาตัวจาก stack มา DFS จนไม่มี node อื่นให้ visit




bool vis2[N]

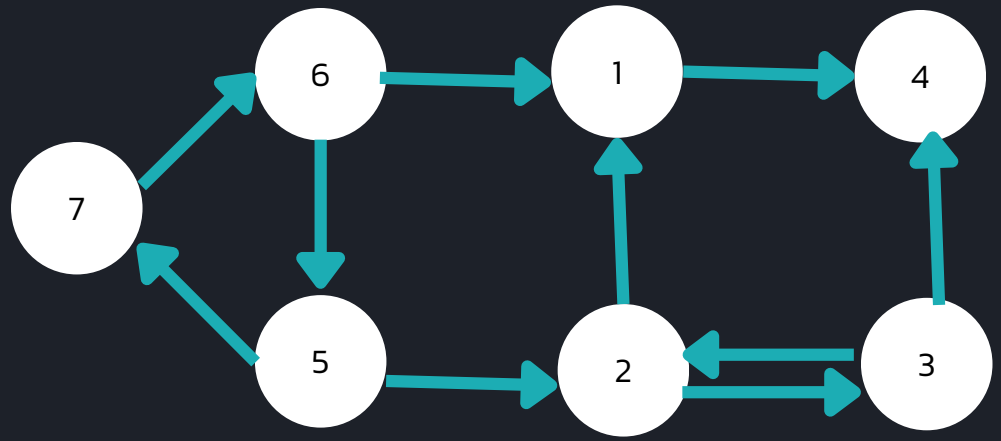



stack <int> s

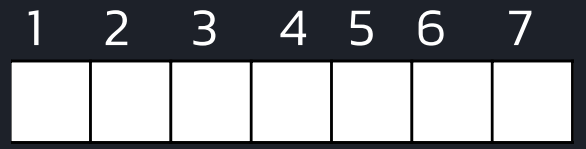

LeViAtHaN

# SOL 1 : KOSARAJU'S O(N+M)

DFS รองสอง : กลับกราฟ แล้วเอาตัวจาก stack มา DFS จนไม่มี node อื่นให้ visit




bool vis2[N]



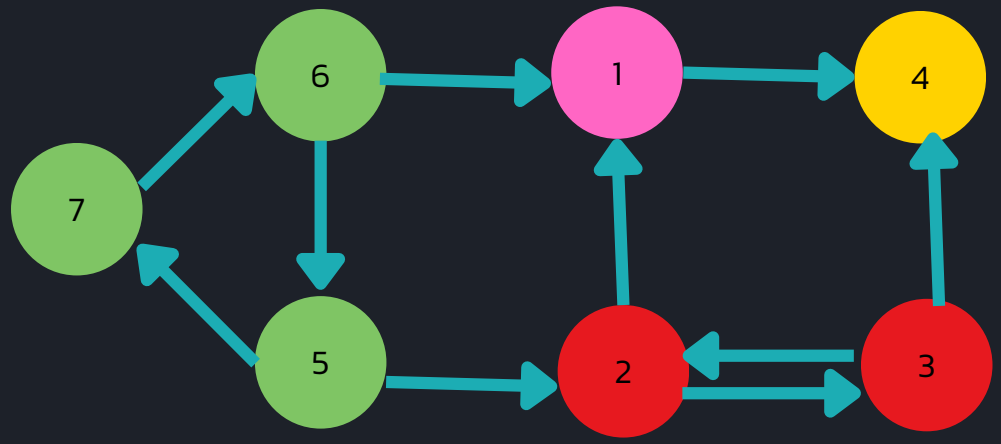

stack <int> s

**5 7 6 3 2 1 4**

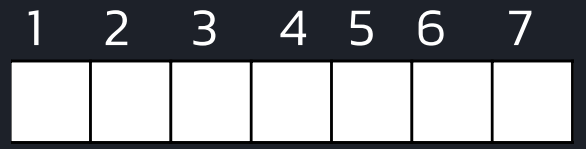
LeViAtHaN

# SOL 1 : KOSARAJU'S O(N+M)

DFS รองสอง : กลับกราฟ แล้วเอาตัวจาก stack มา DFS จนไม่มี node อื่นให้ visit




bool vis2[N]




stack <int> s

**5 7 6 3 2 1 4**

SCC : {4}, {1}, {2,3}, {5,6,7}

LeViAtHaN

# SOL 1 : KOSARAJU'S CODE

ตัวแปร

```
const int NX = 1e5 + 1;
int N,M,u,v;
vector<int> path1[NX], path2[NX];
stack <int> s;
bool vis[NX];

int ct = 0; // many of componant
int id[NX]; // node number of componant
```

ฟังก์ชั่น DFS

```
void DFS(int u, int round, int num = 0) {
    vis[u] = true;

    vector<int> &path = (round == 1) ? path1[u] : path2[u];
    for (auto &v : path) {
        if (!vis[v]) DFS(v, round, num);
    }

    if (round == 1) s.push(u);      //push node
    if (round == 2) id[u] = num;    //set id
}
```

# SOL 1 : KOSARAJU'S CODE

**DFS รอบแรก** : DFS จนเจอตัวที่ไม่มี node อื่นให้ visit แล้วเก็บตัวนั้นลงใน stack

```
//DFS1 : push node to stack
for (int i=1;i<=N;i++) {
    if(!vis[i])
        DFS(i, 1);
}
```

**DFS รองสอง** : กลับกราฟ แล้วเอาตัวจาก stack มา DFS จนไม่มี node อื่นให้ visit

```
//DFS2 : reverse graph, get node from stack
memset(vis,0,sizeof(vis));
while(!s.empty()){
    if(!vis[s.top()]) {
        ct++;
        DFS(s.top(), 2, ct);
    }
    s.pop();
}
```

```
void DFS(int u, int round, int num = 0) {
    vis[u] = true;

    vector<int> &path = (round == 1) ? path1[u] : path2[u];
    for (auto &v : path) {
        if (!vis[v]) DFS(v, round, num);
    }

    if (round == 1) s.push(u);      //push node
    if (round == 2) id[u] = num;    //set id
}
```

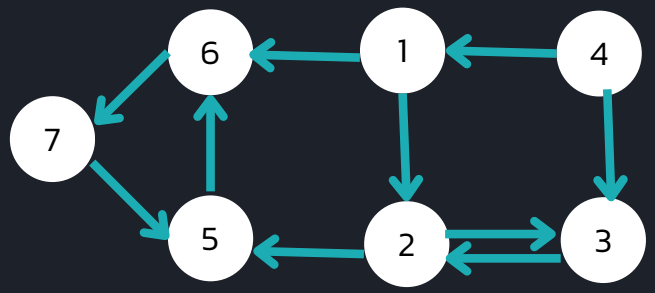
DFS function

LeViAtHaN

# SOL 2 : TARJAN'S O(N+M)

DFS ครั้งเดียว แต่มีการใช้ค่า disc และ lowlink disc[u] คือ ลำดับที่ node u ถูก visit ครั้งแรก ** disc[u] อาจจะไม่เท่ากับค่า u **
หลังจาก DFS v รอบๆแล้ว หาก disc==lowlink แปลว่า node นั้นเป็นจุดเริ่มของ SCC

การ DFS :
เก็บ node u ไว้ใน stack, ตั้ง disc[u]
เยี่ยม node v รอบๆ u
    หาก v ยังไม่ถูกเยี่ยม -> DFS(v)
    หาก v ยังอยู่ใน stack -> set lowlink[u]= min(lowlink[u],lowlink[v])
หาก หลังจาก DFS u แล้วพบว่า low-link[u] = disc[u] -> u เป็น low-link value ของ SCC
    ให้ pop node ใน stack ออก จนเจอ disc[u] ใน stack




vector <int> path[N]

int disc[N]

bool vis[N]

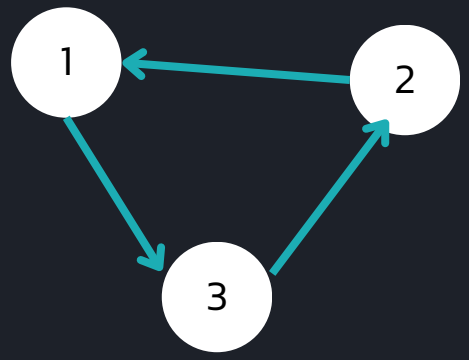
int lowlink[N]

stack <int> s

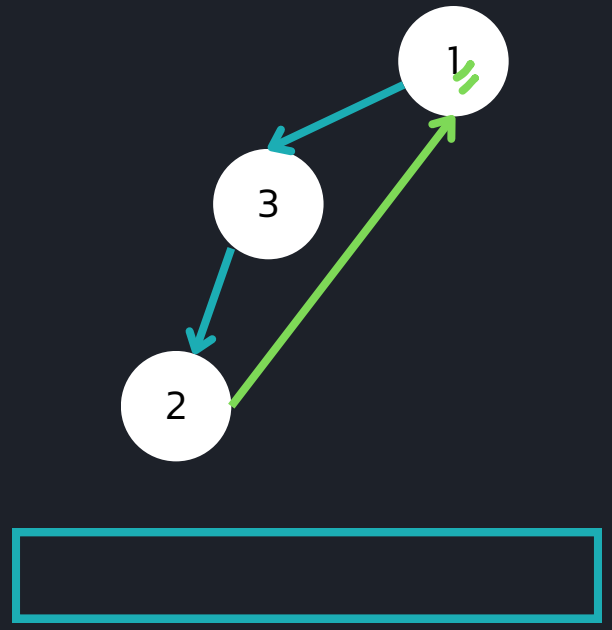
LeViAtHaN

# SOL 2 : TARJAN'S O(N+M)

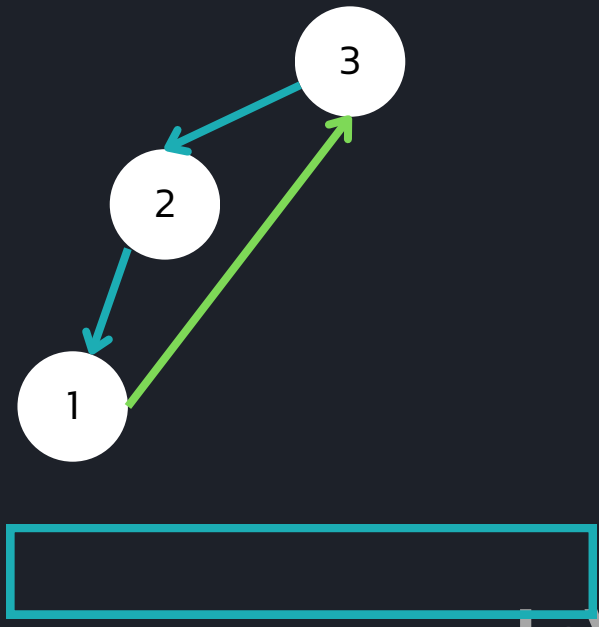
เก็บ node u ไว้ใน stack
เยี่ยม node v รอบๆ u
    หาก v ยังไม่ถูกเยี่ยม -> DFS(v)
    หาก v ยังอยู่ใน stack -> set low-link[u]= min(disc[u],disc[v])
หาก หลังจาก DFS u แล้วพบว่า low-link[u] = disc[u] แปลว่า u เป็น low-link value ของ SCC
    ให้ pop node ใน stack ออก จนเจอ disc[u] ใน stack




EX1 : เริ่มจาก 1

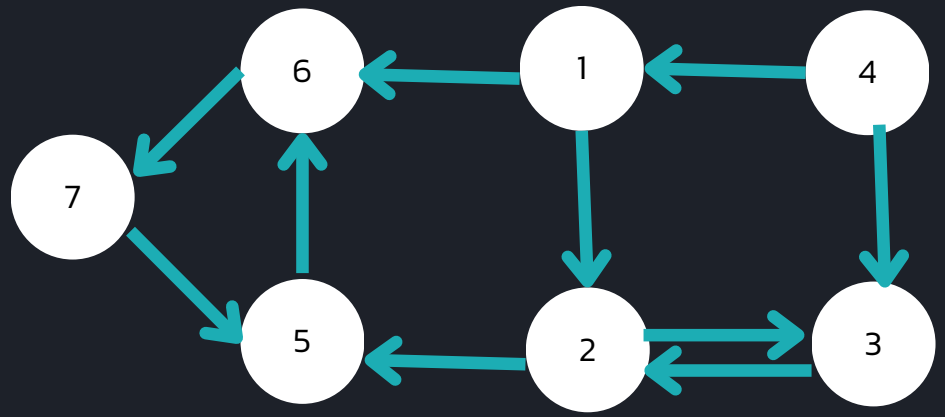



EX1 : เริ่มจาก 3




LeViAtHaN

# SOL 2 : TARJAN'S O(N+M)




bool vis[N]

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| | | | | | | |

int disc[N]

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| | | | | | | |

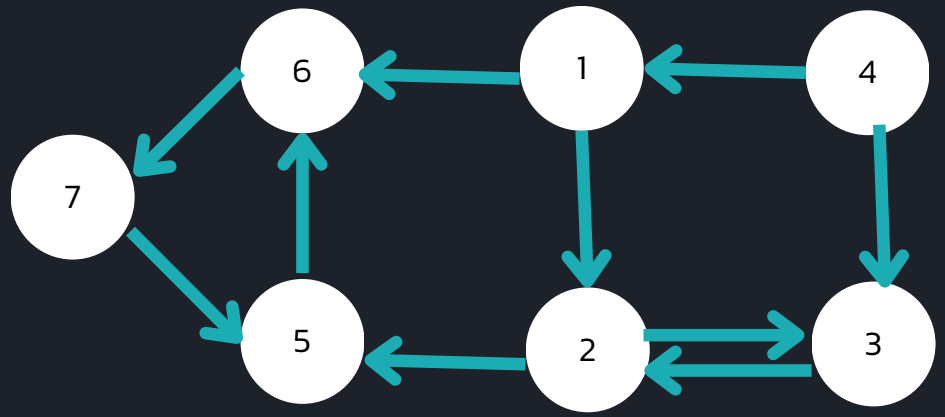
int lowlink[N]

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| | | | | | | |

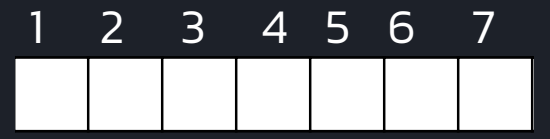
stack <int> s

LeViAtHaN

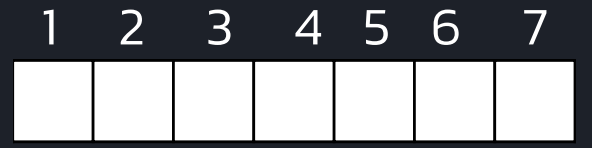
# SOL 2 : TARJAN'S O(N+M)



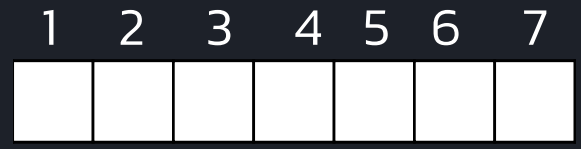

bool vis[N]




int disc[N]



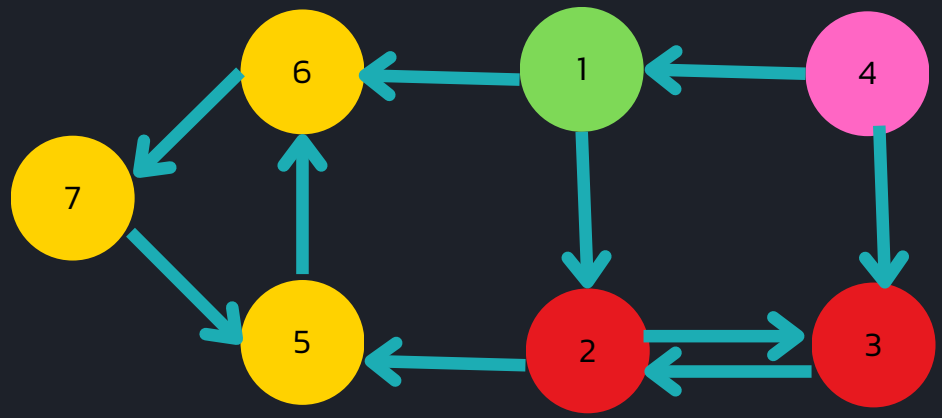

int lowlink[N]



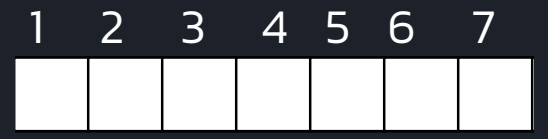

stack <int> s

LeViAtHaN

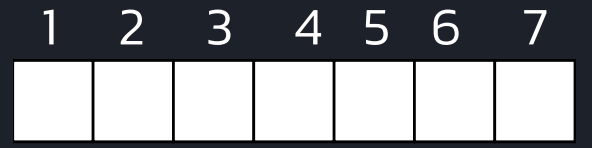
# SOL 2 : TARJAN'S O(N+M)



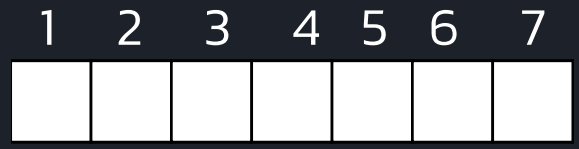

bool vis[N]




int disc[N]




int lowlink[N]




stack <int> s

LeViAtHaN

# SOL 2 : TARJAN'S CODE

## ตัวแปร

```
const int NX = 1e5 + 1;
int N,M,u,v;
vector<int> path[NX];
stack <int> s;
bool vis[NX];

int ct = 0; // many of componant
int ids = 1; // assign disc
int disc[NX], lowlink[NX]; // node number of componant
bool onStack[NX]; //check if u is in the stack
```

## ฟังก์ชั่น DFS

```
void DFS(int u) {
    vis[u] = true;
    onStack[u] = true;
    disc[u] = lowlink[u] = ids++;
    s.push(u);

    //check all neighbor
    for (auto v : path[u]) {
        if (!vis[v]) DFS(v);
        if (onStack[v]) lowlink[u]=min(lowlink[u],lowlink[v]);
    }

    //if we are low link of SCC
    if(disc[u]==lowlink[u]){
        ct++;
        while(!s.empty()){
            int temp=s.top(); s.pop();
            onStack[temp]=0;
            lowlink[temp]=u;
            if(temp==u) break;
        }
    }
}
```

LeViAtHaN

# CSES Planets and Kingdoms

## [20 min]

LeViAtHaN

# เฉลย CSES PLANETS AND KINGDOMS

เราใช้ TARJAN'S เลยเพื่อที่ไม่ต้องไปรื้อ DFS หา ID/LOWLINK อีกหลายรอบ
ข้อนี้ทำตาม ALGO ตรงๆไปได้เลย เว้นแต่ตรง LOWLINK เราต้องใช้เป็นจำนวนของ SCC ที่เจอแทน

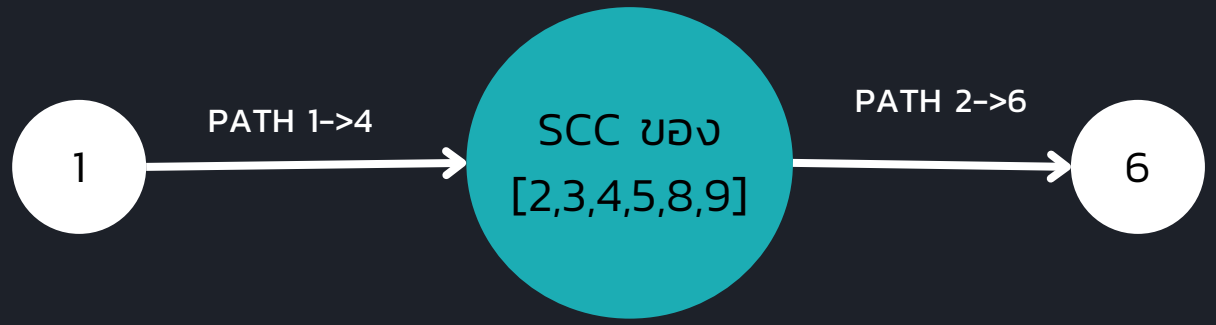
# CSES Coin Collector [40 min]

LeViAtHaN

# เฉลย CSES COIN COLLECTOR



สร้างกราฟใหม่ โดย ใช้ SCC รวม NODE หลาย NODE มาเป็น NODE เดียวที่มีเหรียญรวมกัน

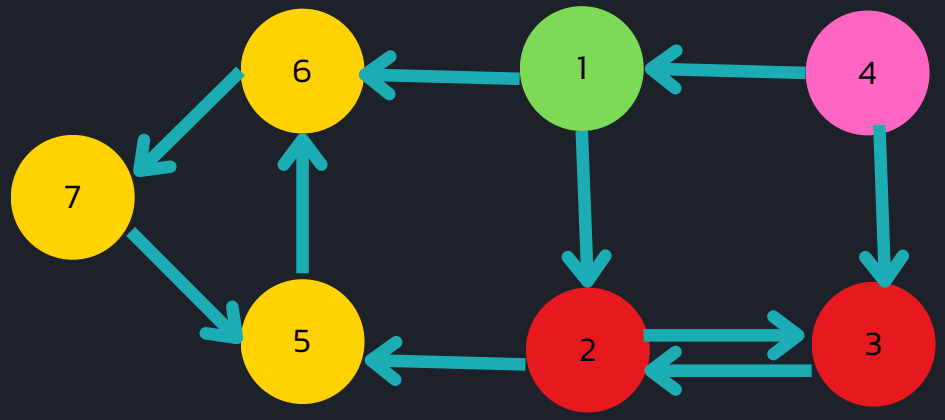



ในนี้เดินวนแล้วไปไหนก็ได้เพราะโจทย์ไม่กำหนดว่าแต่ละ NODE เดินได้แค่ครั้งเดียว

TAsk 1 : ใช้ Kasaraju's but use vector instand of stack มา gen หาแต่ละ SCC ยัดแต่ละ lowlink ใส่ใน comps
Task 2 : รวม value ของแต่ละ SCC
Task 3 : ย้าย reverse path ของ SCC ที่ต่างกันไปยัดใส่ lowlink ของแต่ละ SCC
Task 4 : ไล่ DP แต่ละตัวว่าสามารถมีค่าที่มาถึงได้มากสุดเท่าไหร่

LeViAtHaN

# KEY TAKEAWAY




**Strongly Connected Components**

บางส่วนของ directed graph ที่ node ทุก node สามารถไปหากันได้

**disc** ลำดับที่ node u ถูก visit ครั้งแรก

**lowlink** node v ที่มีค่าน้อยที่สุดที่ node u จะไปถึงได้

## KOSARAJU'S O(N+M)

DFS สองรอบและมีการกลับกราฟ

**DFS รอบแรก :**

DFS จนเจอตัวที่ไม่มี node อื่นให้ visit แล้วเก็บตัวนั้นลงใน stack

**DFS รองสอง :**

กลับกราฟ แล้วเอาตัวจาก stack มา DFS จนไม่มี node อื่นให้ visit

ตัวที่ DFS ไปถึงได้ แปลว่าอยู่ใน SCC เดียวกัน

## TARJAN'S O(N+M)

ใช้ disc และ lowlink

DFS แค่หนึ่งครั้ง โดยจาก u

หาก v ยังไม่ถูกเยี่ยม -> DFS(v)

หาก v ยังอยู่ใน stack -> set low-link[u]= min(disc[u],disc[v])

หลังจาก DFS u แล้วพบว่า low-link[u] = disc[u]

ให้ pop node ใน stack ออก จนเจอ disc[u] ใน stack

ตัวที่ถูก pop ออกมา แปลว่าอยู่ใน SCC เดียวกัน

LeViAtHaN

# HOMEWORK

- CSES Game Routes
- CESS Round Trip 2