

DAY 5



TOPOLOGICAL SORT &
STRONGLY CONNECTED COMPONENTS

ឆេស GOD OF WAR



ឆ្លេរ ROYAL PARADE



เฉลยการบ้าน น้ำยาเพิ่มความเร็ว



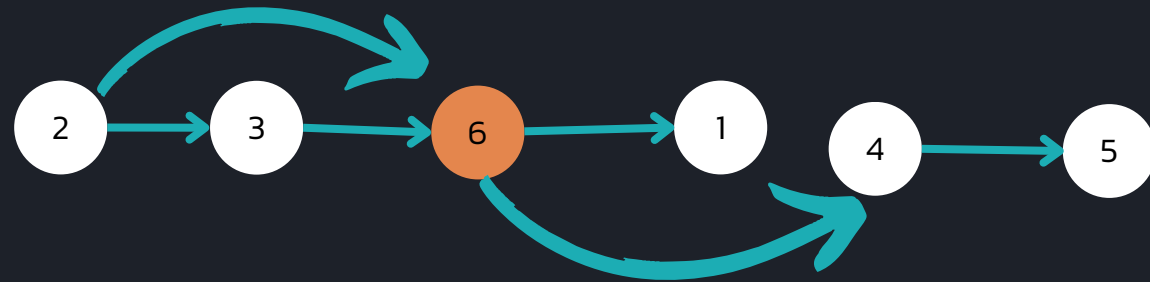
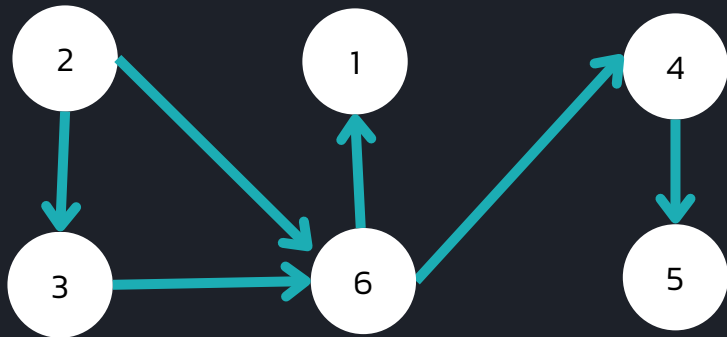


TOPOLOGICAL SORT

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 ที่ชี้เข้าทั้งหมดถูกเยี่ยมชมมาหมดแล้ว



CSES Course Schedule

[15 min]

ឯកសារ CSES COURSE SCHEDULE





CSES Longest Flight Route

[15 min]

ឆ្នោត CSES LONGEST FLIGHT ROUTE





USACO Timeline

[20 min]

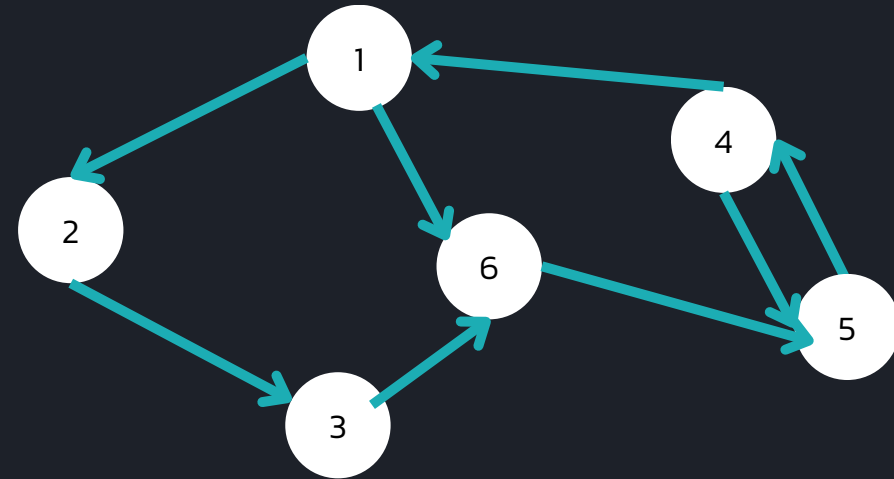
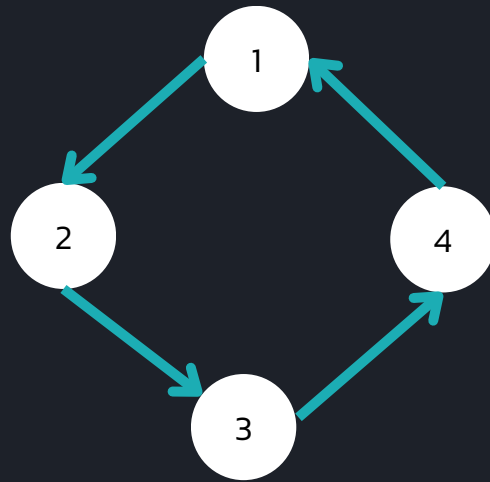
ឆ្លេរ USACO TIMELINE





STRONGLY CONNECTED COMPONENTS

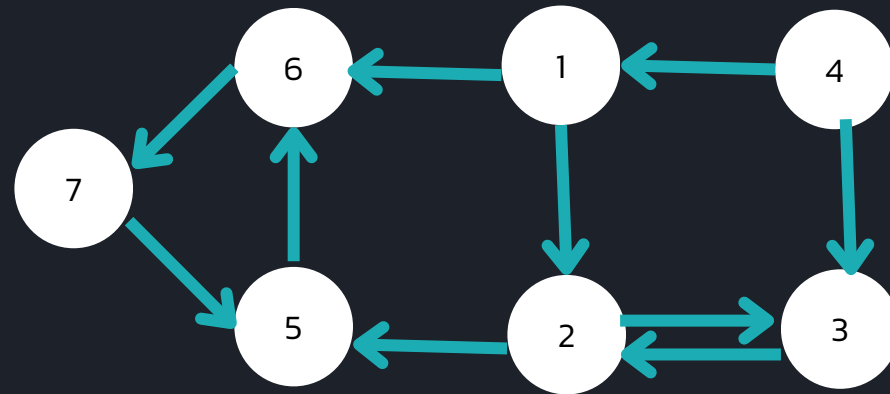
STRONGLY CONNECTED COMPONENTS คืออะไร



Strongly Connected

a directed graph ที่ node ทุกล node สามารถไปหากันได้

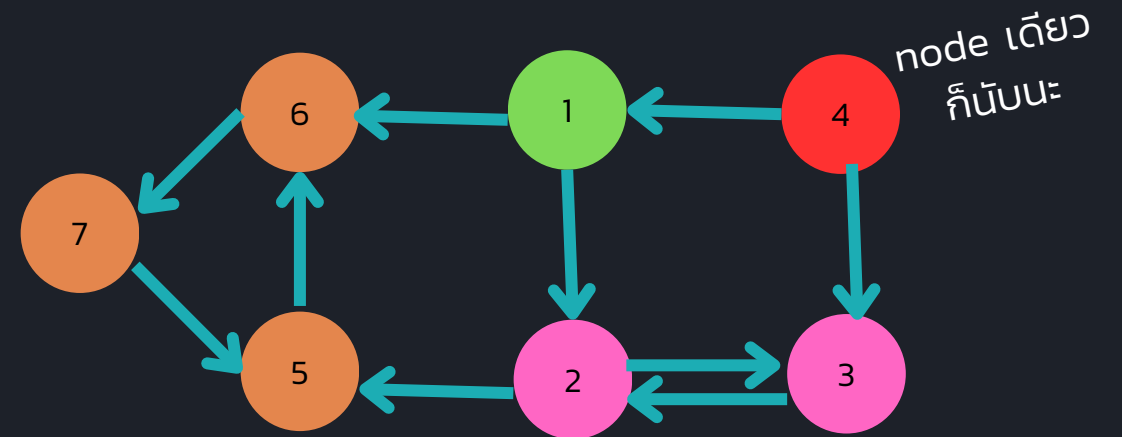
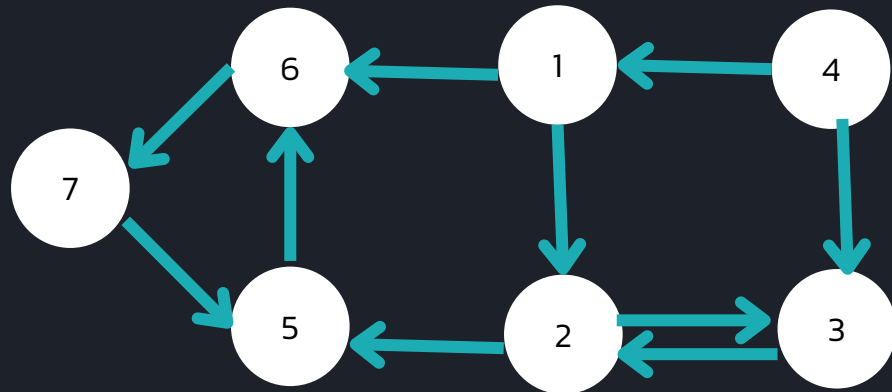
STRONGLY CONNECTED COMPONENTS คืออะไร



Strongly Connected Components

บางส่วนของ directed graph **ที่ใหญ่ที่สุด** ที่ node ทุก node สามารถไปหากันได้

STRONGLY CONNECTED COMPONENTS คืออะไร



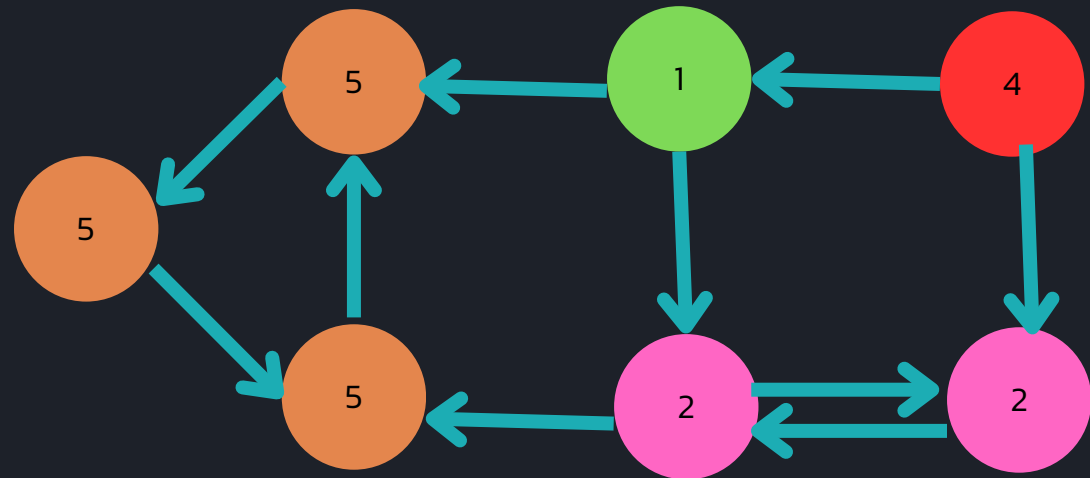
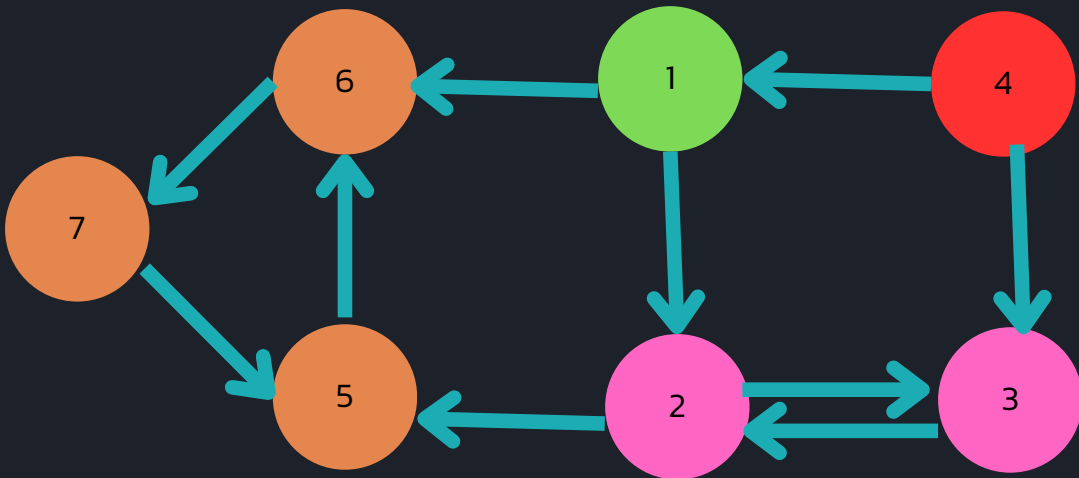
Strongly Connected Components

บางส่วนของ directed graph **ที่ใหญ่ที่สุด** ที่ node ทุก node สามารถไปหากันได้

LOW LINK คืออะไร



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

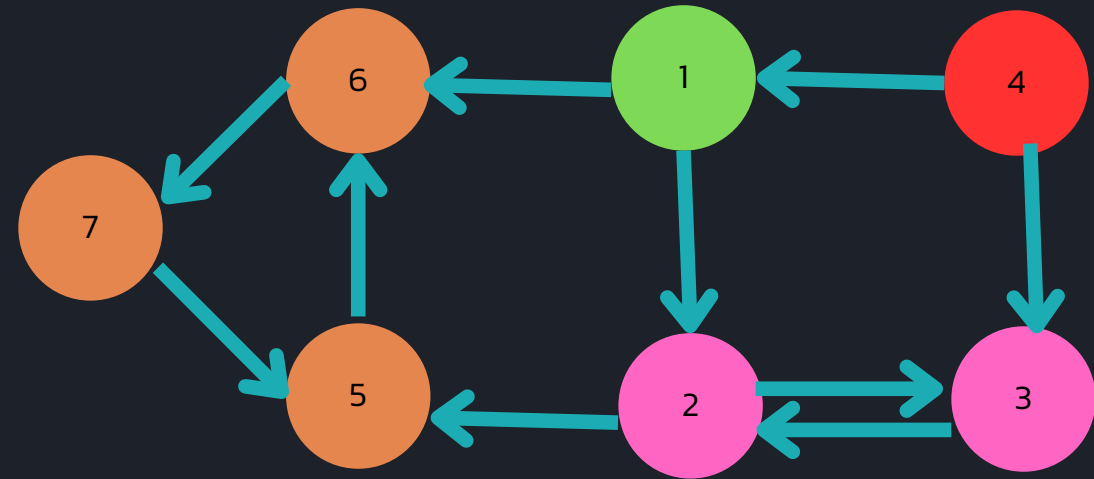


STRONGLY CONNECTED COMPONENTS คืออะไร



โจทย์ Strongly Connected Components ที่พบบ่อยๆ คือ

- หาจำนวน SCC ในกราฟ
- หา node ตัวแทน SCC แต่ละอัน (lowlink)
- หาขนาด SCC ที่ใหญ่หรือเล็กที่สุดในกราฟ



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

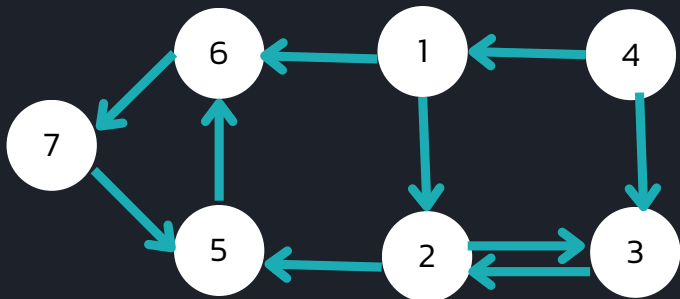


DFS สองรอบ เพื่อหา SCC ใน topological order

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

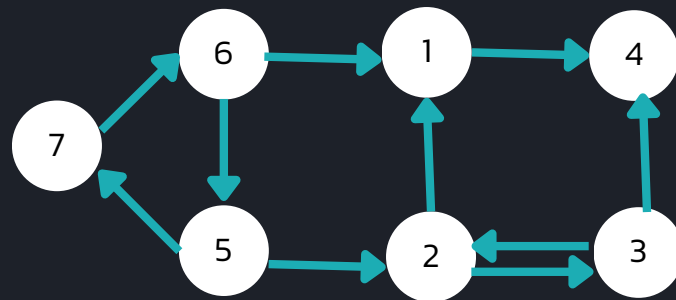
DFS รอบแรก : DFS จนเจอตัวที่ไม่มี node อื่นให้ visit แล้วเก็บตัวนั้นลงใน stack

DFS รอบสอง : 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]



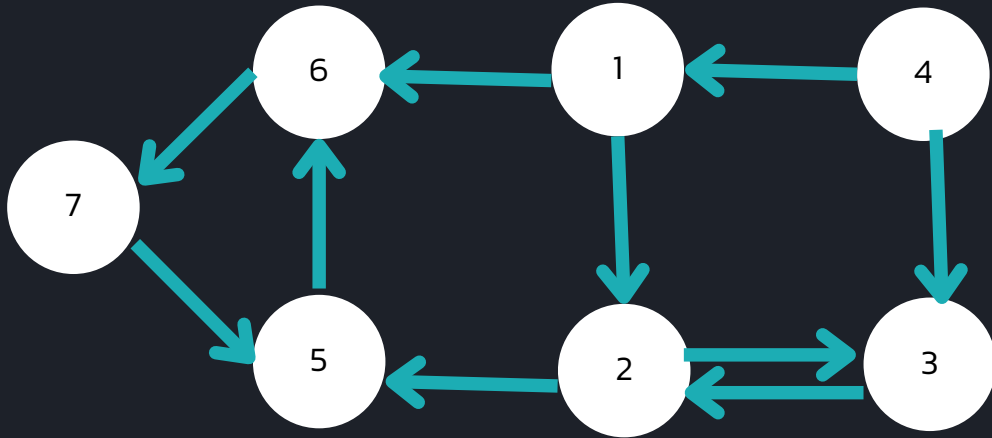
stack <int> s



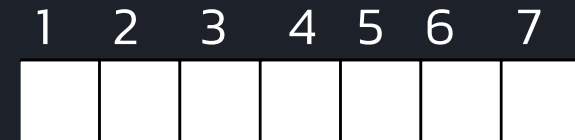
SOL 1 : KOSARAJU'S $O(N+M)$



DFS รอบแรก : DFS จนเจอตัวที่ไม่มี node อื่นให้ visit แล้วเก็บตัวนั้นลงใน stack



`bool vis1[N]`



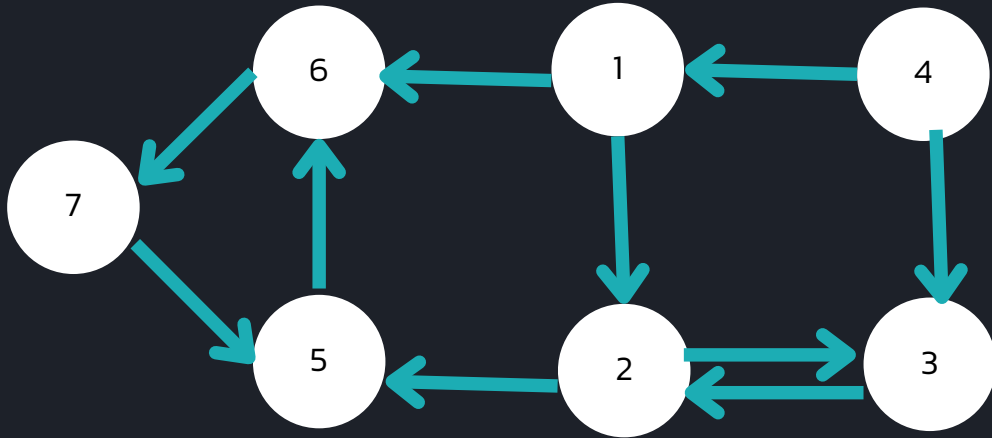
`stack <int> s`



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



DFS รอบแรก : DFS จนเจอตัวที่ไม่มี node อื่นให้ visit แล้วเก็บตัวนั้นลงใน stack



`bool vis1[N]`



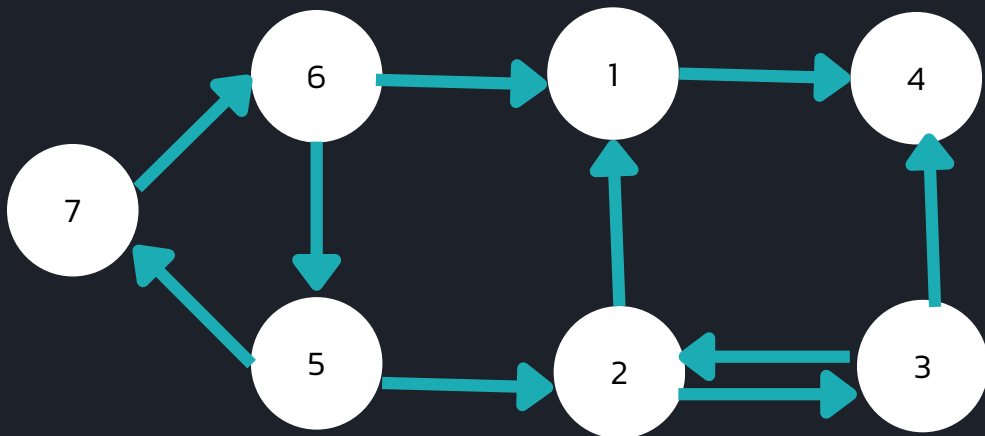
`stack <int> s`



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



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



bool vis2[N]

1	2	3	4	5	6	7

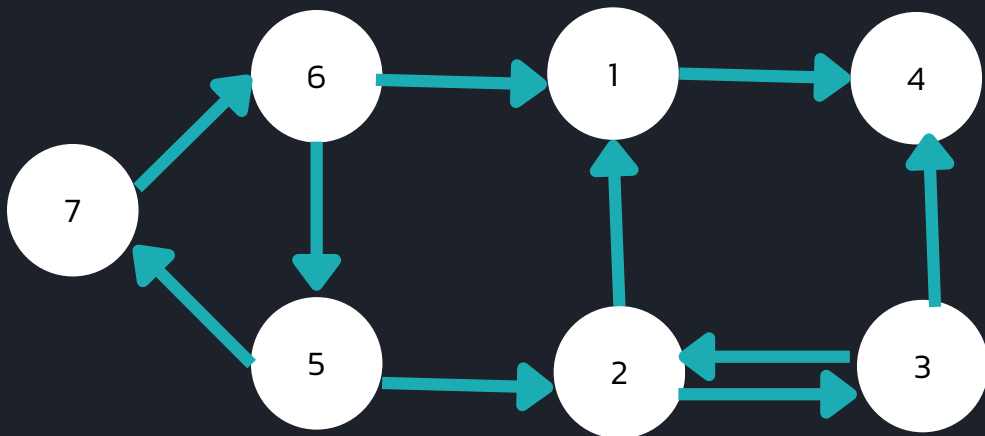
stack <int> s

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

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



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



bool vis2[N]

1	2	3	4	5	6	7

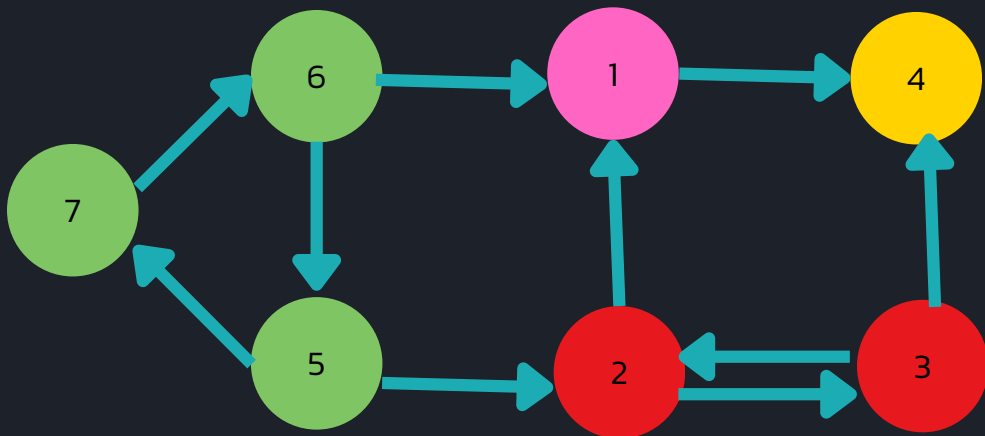
stack <int> s

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

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



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



bool vis2[N]

1	2	3	4	5	6	7

stack <int> s

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

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

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 component
int id[NX]; // node number of component
```

ฟังก์ชัน 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 รอบสอง : 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

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` รอบๆ แล้ว หาก `disc==lowlink` แปลว่า node นั้นเป็นจุดเริ่มของ SCC

การ DFS :

เก็บ node `u` ไว้ใน stack, ตั้ง `disc[u]`

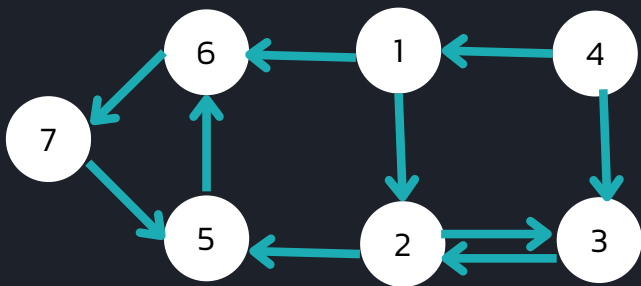
เยี่ยม node `v` รอบๆ `u`

หาก `v` ยังไม่ถูกเยี่ยม \rightarrow DFS(`v`)

หาก `v` ยังอยู่ใน stack \rightarrow set `lowlink[u] = min(lowlink[u], lowlink[v])`

หาก หลังจาก DFS `u` แล้วพบว่า `lowlink[u] = disc[u]` \rightarrow `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`



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



เก็บ node u ไว้ใน stack

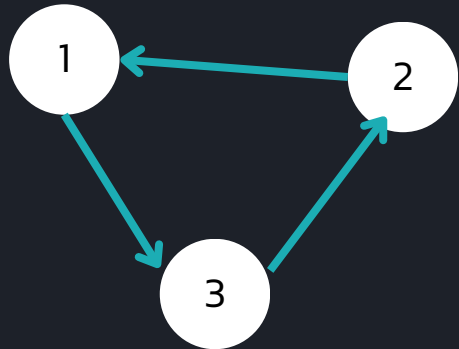
เยี่ยมชม node v รอบๆ u

หาก v ยังไม่ถูกเยี่ยมชม \rightarrow DFS(v)

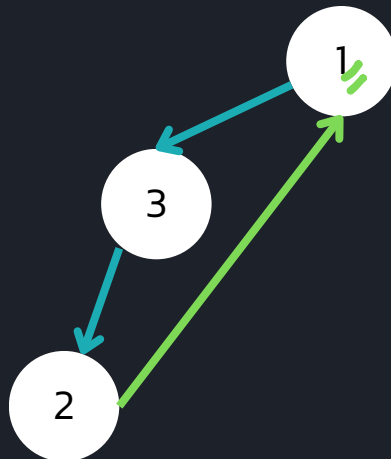
หาก v ยังอยู่ใน stack \rightarrow set low-link[u] = min(low-link[u], low-link[v])

หาก หลังจาก DFS u แล้วพบว่า low-link[u] = low-link[u] แปลว่า u เป็น low-link value ของ SCC

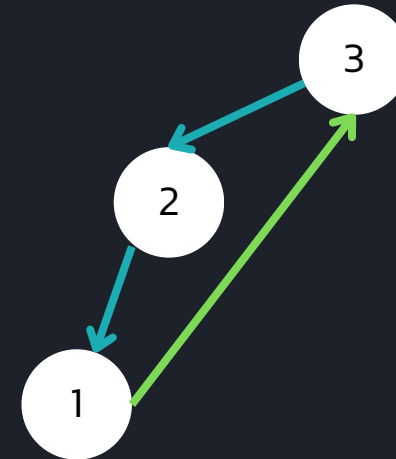
ให้ pop node ใน stack ออก จนถึง low-link[u] ใน stack



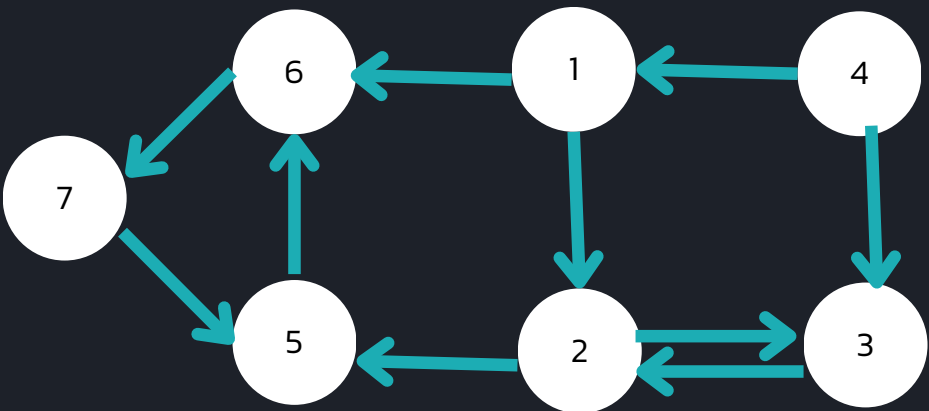
EX1 : เริ่มจาก 1



EX1 : เริ่มจาก 3



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



bool vis[N]

1	2	3	4	5	6	7

int disc[N]

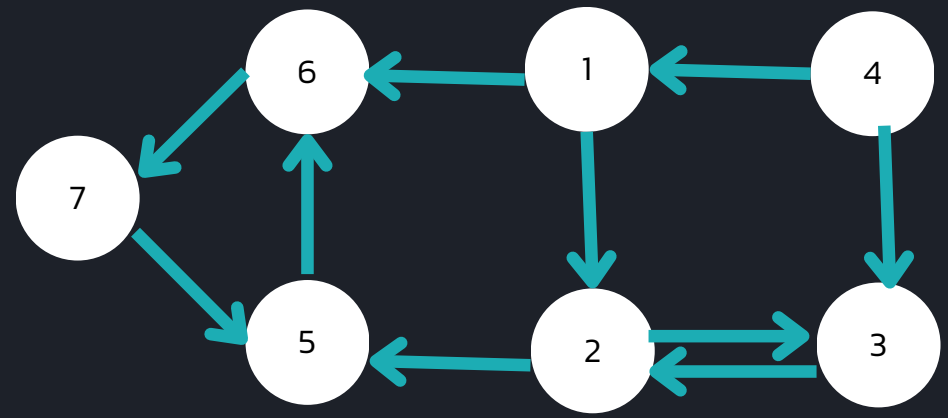
1	2	3	4	5	6	7

int lowlink[N]

1	2	3	4	5	6	7

stack <int> s

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



bool vis[N]

1	2	3	4	5	6	7

int disc[N]

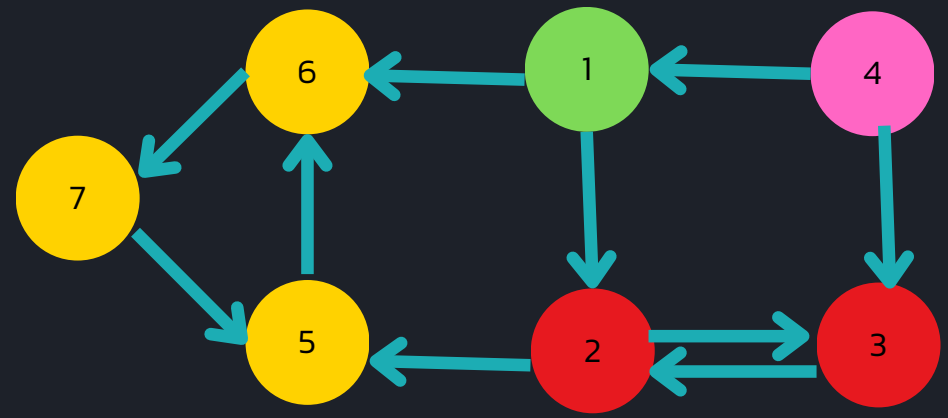
1	2	3	4	5	6	7

int lowlink[N]

1	2	3	4	5	6	7

stack <int> s

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

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
```

ฟังก์ชัน 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;
        }
    }
}
```




CSES Planets and Kingdoms

[20 min]

ឆ្នោត CSES PLANETS AND KINGDOMS



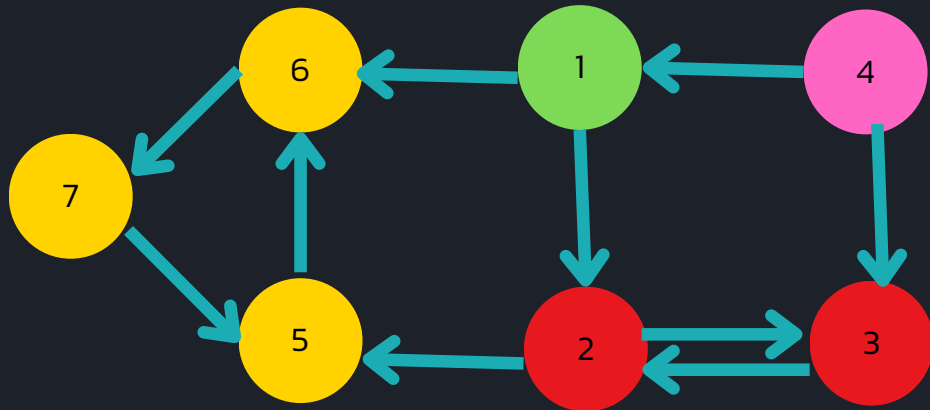


CSES Coin Collector [40 min]

ឆេស CSES COIN COLLECTOR



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 รอบสอง :

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 \rightarrow DFS(v)

หาก v ยังอยู่ใน stack \rightarrow 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

HOMework



- CSES Game Routes
- CESS Round Trip 2