



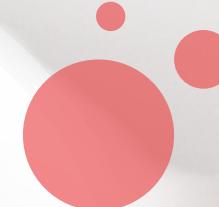
Graph

สื่อการสอนสำหรับค่ายคอมพิวเตอร์โอลิมปิก สอน. ค่าย 3/2562

โดย นางสาว นันก์ณกัส บันลือสมบัติกุล, สถาบันวิทย์สีรีเมร์ (VISTEC)

ห้ามน้ำเสียของการสอนเด็ดๆ ในใช้ได้โดยไม่ได้รับอนุญาตจากผู้จัดทำ

<http://www.free-powerpoint-templates-design.com>



สวัสดี

นันก์กุ๊ส บันลือสมบัติกุล (แบบ)

จบปริญญาตรี สาขาวิชาการคอมพิวเตอร์
มหาวิทยาลัยธรรมศาสตร์

ปัจจุบัน นักศึกษาปริญญาเอก
สาขา Information Science and Technology
สถาบันวิทย์สีเมือง (VISTEC)



Agenda

01

Recap Graph

ทบทวนความรู้จากค่ายที่แล้ว

02

DFS on 2D grid

การทำงาน DFS บนกราฟที่แสดงด้วย 2D grid

03

BFS on 2D grid

การทำงาน BFS บนกราฟที่แสดงด้วย 2D grid

04

โจทย์ปัญหา

Problems on 2D grid graph

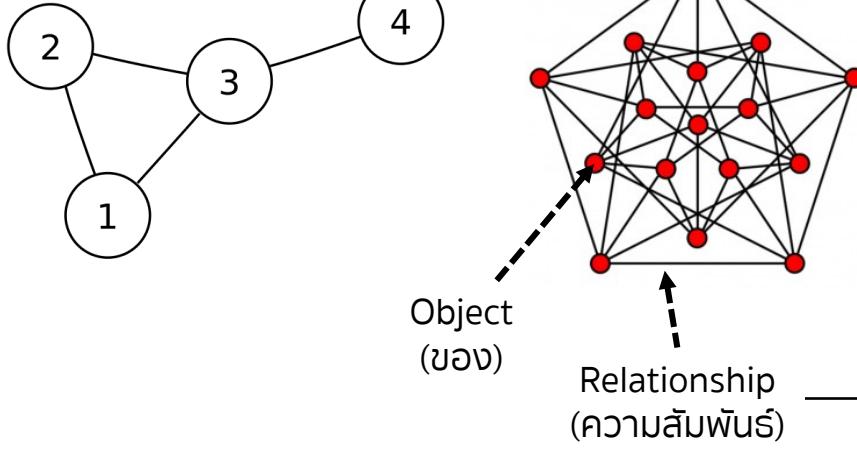
Graph

กับทวนความรู้เกี่ยวกับกราฟ และการท่องกราฟ

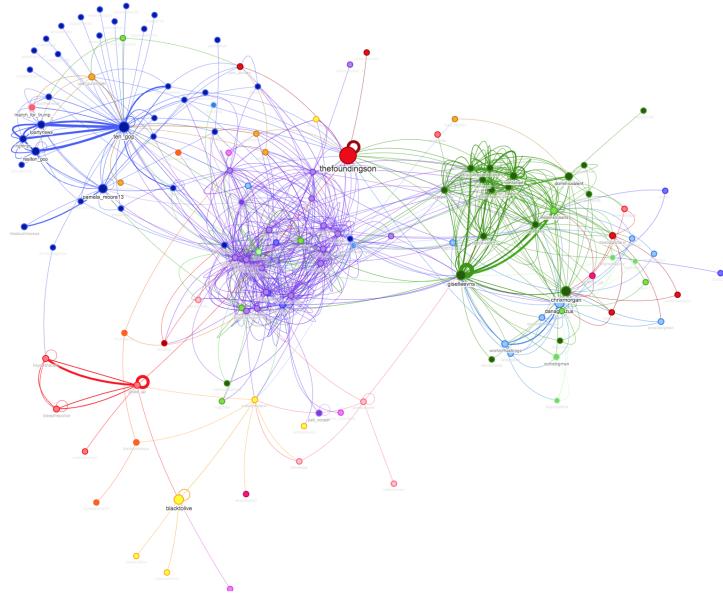


ทำความรู้จักกับกราฟ

กราฟคืออะไร



ឯកតែថា “ទាននឹងរាជរដ្ឋបាលដើម្បីក្រោយពីការបង្កើតក្រសួង”

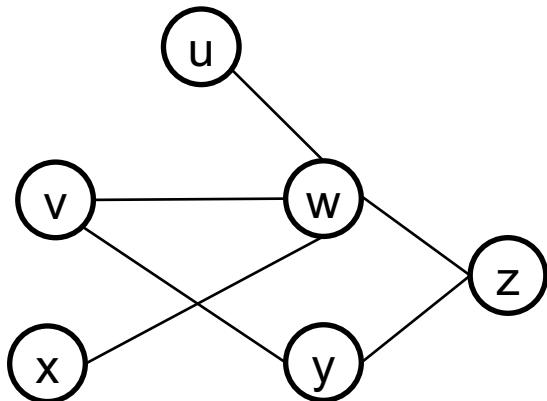


นิยามกราฟ

การพูดถึงกราฟด้วยตัวโปรแกรมคณิตศาสตร์



- **Adjacent (ติดกับ) => $v \in V$ or edge (u, v) is adjacent to u if $(u, v) \in E$**
- **Degree ของ u = จำนวน edge ที่ adjacent to u**

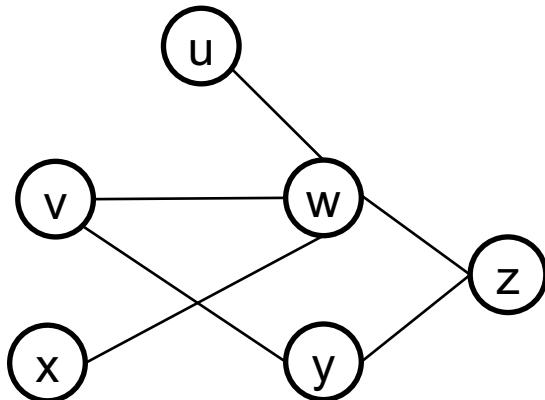


นิยามกราฟ

การพูดถึงกราฟด้วยตัวโปรแกรมคณิตศาสตร์



- **Path P** = ลำดับของ vertex v_1, v_2, \dots, v_k ที่ $(v_i, v_{i+1}) \in E$ สำหรับ $1 \leq i \leq k$
โดยมี v_1 เป็นจุดเริ่มต้น (source) และ v_k เป็นจุดสิ้นสุด (destination)
จะกล่าวว่า “P เป็น path จาก v_1 ไปยัง v_k ”



การสร้างกราฟ

วิธีการสร้างกราฟ



- การเก็บค่าของกราฟ สามารถแบ่งได้เป็น 2 วิธี

1 Adjacency Matrix

	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

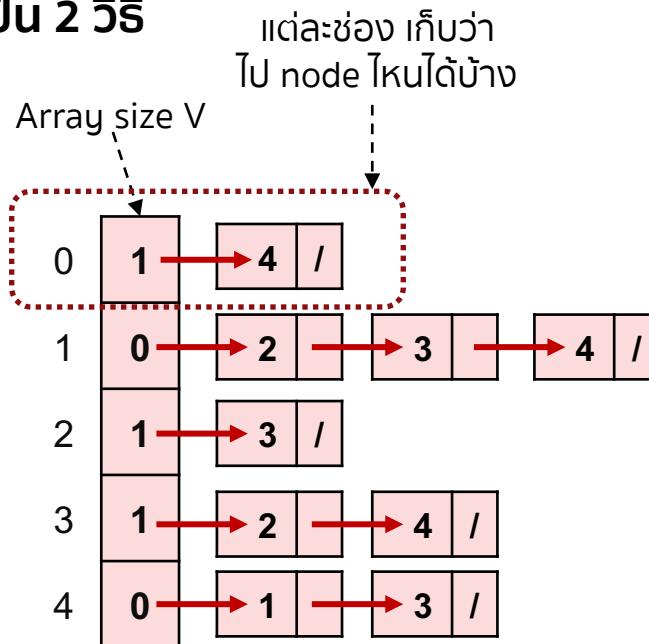
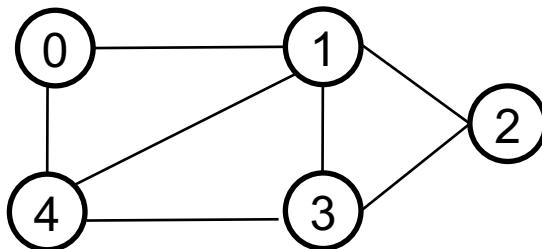
- เก็บด้วย 2D Array : size $V \times V$
- ข้อดี : เช็คว่ามี Edge E ไหม? $\rightarrow O(1)$
เพิ่ม edge, ลบ edge $\rightarrow O(1)$
- ข้อเสีย : sparse graph จะทำให้เปลืองพื้นที่โดยใช้เหตุ

การสร้างกราฟ

วิธีการสร้างกราฟ

- การเก็บค่าของกราฟ สามารถแบ่งได้เป็น 2 วิธี

2 Adjacency List:

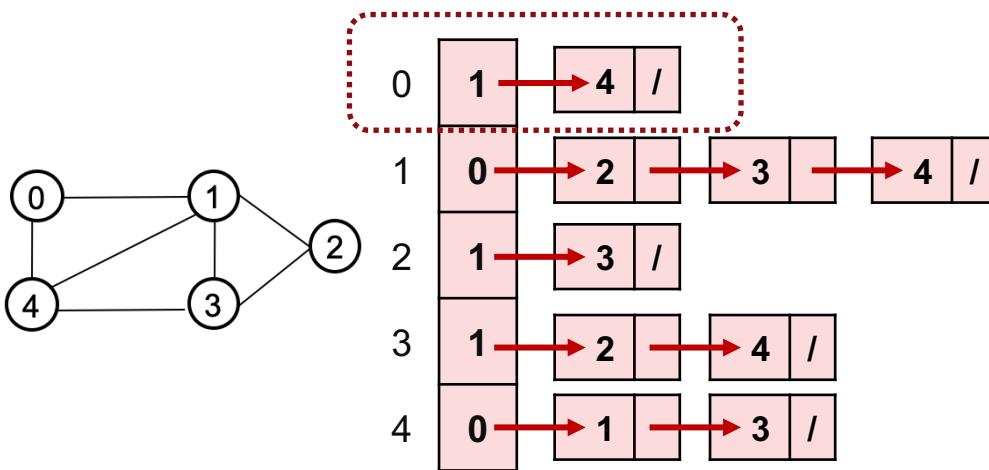


การสร้างกราฟ

วิธีการสร้างกราฟ

- การเก็บค่าของกราฟ สามารถแบ่งได้เป็น 2 วิธี

2 Adjacency List: เก็บด้วย Array of Vectors



```
void addEdge(vector<int> adj[], int u, int v)
{
    adj[u].push_back(v);
    adj[v].push_back(u);
}

int main()
{
    int V = 5;
    vector<int> adj[V];
    addEdge(adj, 0, 1);
    addEdge(adj, 0, 4);
    addEdge(adj, 1, 2);
    addEdge(adj, 1, 3);
    addEdge(adj, 1, 4);
    addEdge(adj, 2, 3);
    addEdge(adj, 3, 4);
    printGraph(adj, V);
    return 0;
}
```

การท่องกราฟ

Breadth First Search or BFS, Depth First Search or DFS



- กำหนด: Graph $G = (V, E)$, จุดเริ่มต้น (**Source**)

การท่องกราฟ คือ การหา path ที่เริ่มจากจุดเริ่มต้น ไปจนครบทุก node บนกราฟ
มี 2 วิธีหลักๆ คือ

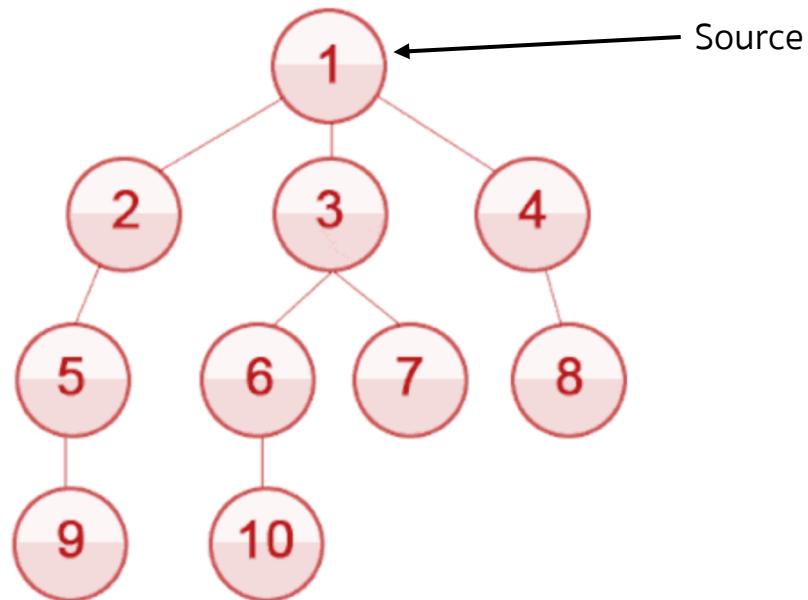
- Breadth First Search (BFS) การเดินในแนวกว้าง
- Depth First Search (DFS) การเดินในแนวลึก

การท่องกราฟ

Breadth First Search or BFS



- Breadth First Search (BFS) การเดินในแนวกว้าง



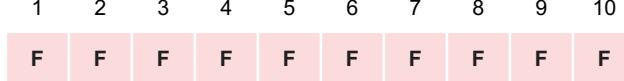
การท่องกราฟ

Breadth First Search or BFS

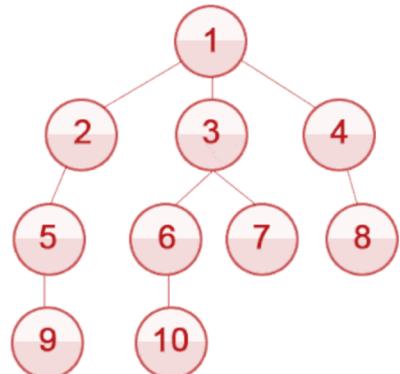


- **Breadth First Search (BFS) การเดินในแนวกว้าง**

queue → { }
visited 1 2 3 4 5 6 7 8 9 10



output



ใช้ queue เก็บ node ที่เราต้องไป
และ bool[] visited เก็บว่า เราผ่าน node มาแล้วหรือยัง

```
bool visited[V] = {false};  
void BFS(int s)  
{  
    queue<int> q;  
    visited[s] = true;  
    q.push(s);  
  
    while(!q.empty()) {  
        s = q.front();  
        cout << s << " ";  
        q.pop();  
        for (each node in adj(s)) {  
            if (!visited[node]) {  
                visited[node] = true;  
                q.push(node);  
            }  
        }  
    }  
}
```

ກາຣທົ່ວກຮາວ

Breadth First Search or BFS



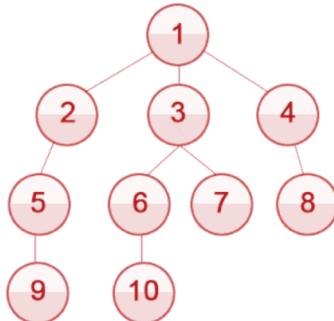
- Application of Breadth First Search (BFS)

queue → { }

visited 1 2 3 4 5 6 7 8 9 10
F F F F F F F F F F

dist 0 0 0 0 0 0 0 0 0 0

output



ໃຫ້ Shortest path ແລ້ວ Minimum number of steps

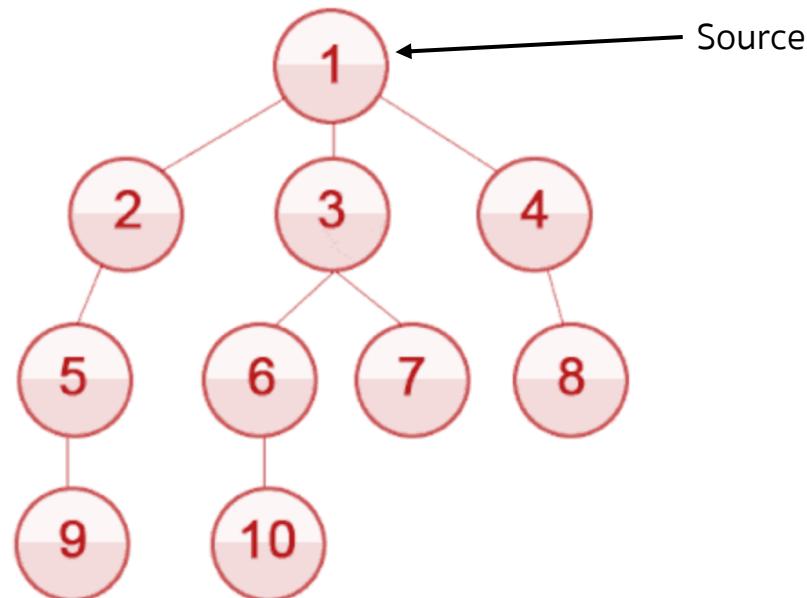
```
bool visited[V] = {false};  
void BFS(int s)  
{  
    int dist[V] = {0}; /* added */  
    queue<int> q;  
    visited[s] = true;  
    q.push(s);  
  
    while(!q.empty()) {  
        s = q.front();  
        cout << s << " ";  
        q.pop();  
        for (each node in adj(s)) {  
            if (!visited[node]) {  
                visited[node] = true;  
                dist[node] = dist[s] + 1;  
                q.push(node);  
            }  
        }  
    }  
}
```

การท่องกราฟ

Depth First Search or DFS



- Depth First Search (DFS) การเดินในแนวลึก



การท่องกราฟ

Depth First Search or DFS



- **Depth First Search (DFS) การเดินในแนวลึก**

```
bool visited[V] = {false};  
void BFS(int s)  
{  
    queue<int> q;  
    visited[s] = true;  
    q.push(s);  
  
    while(!q.empty()) {  
        s = q.front();  
        cout << s << " ";  
        q.pop();  
        for (each node in adj(s)) {  
            if (!visited[node]) {  
                visited[node] = true;  
                q.push(node);  
            }  
        }  
    }  
}
```

ใช้ **stack** เก็บ node ที่เราต้องไป
และ **bool[] visited** เก็บว่า เราผ่าน node มาแล้วหรือยัง

```
bool visited[V] = {false};  
void DFS(int s)  
{  
    stack<int> st;  
    visited[s] = true;  
    st.push(s);  
  
    while(!st.empty()) {  
        s = st.top();  
        cout << s << " ";  
        st.pop();  
        for (each node in adj(s)) {  
            if (!visited[node]) {  
                visited[node] = true;  
                st.push(node);  
            }  
        }  
    }  
}
```

การท่องกราฟ

Depth First Search or DFS

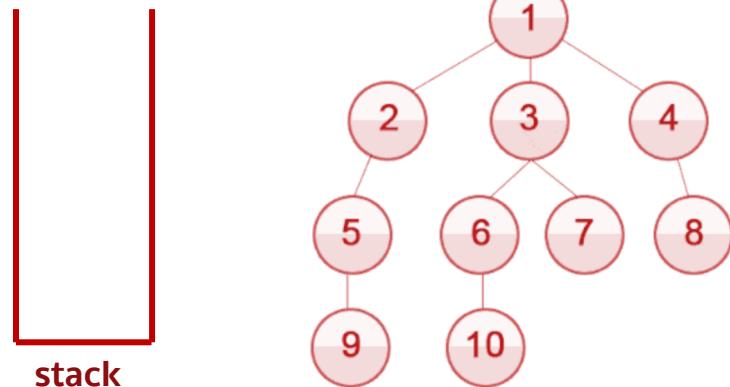


- Depth First Search (DFS) การเดินในแนวลึก

```
bool visited[V] = {false};  
  
void DFS(int s)  
{  
    stack<int> st;  
    visited[s] = true;  
    st.push(s);  
  
    while(!st.empty()) {  
        s = st.top();  
        cout << s << " ";  
        st.pop();  
        for (each node in adj(s)) {  
            if (!visited[node]) {  
                visited[node] = true;  
                st.push(node);  
            }  
        }  
    }  
}
```

1	2	3	4	5	6	7	8	9	10
visited	F	F	F	F	F	F	F	F	F

output



การท่องกราฟ

Depth First Search or DFS



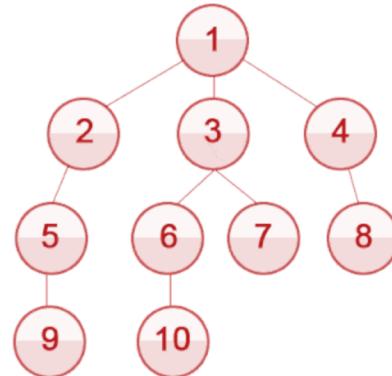
- Depth First Search (DFS) การเดินในแนวลึก (implement แบบ recursive)

```
void DFS(int v)
{
    visited[v] = true;
    cout << v << " ";

    for (each node in adj[v]) {
        if (!visited[node])
            DFS(node);
    }
}
```

1	2	3	4	5	6	7	8	9	10
visited	F	F	F	F	F	F	F	F	F

output



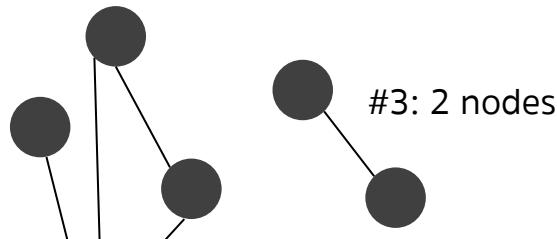
การท่องกราฟ

Depth First Search or DFS



- Application of Depth First Search (DFS) ใช้หา Connected components in graph

#1: 4 nodes



#3: 2 nodes

Connected components = 3

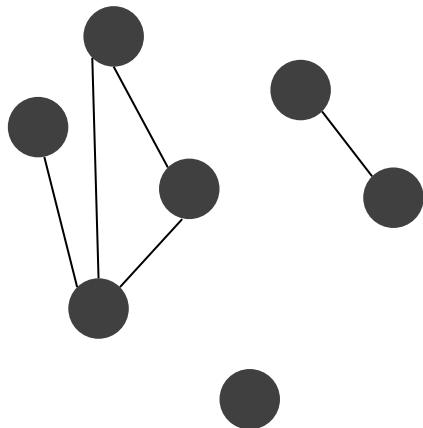
#2: 1 node

การท่องกราฟ

Depth First Search or DFS



- Application of Depth First Search (DFS) ใช้หา Connected components in graph



```
int count = 0;
for(int v=0; v<V; v++){
    if(!visited[v]) {
        DFS(adj, s, v);
        count++;
    }
}
```

การท่องกราฟ

Depth First Search or DFS



- แบบฝึกหัด (DFS)

ในชั้นเรียนประถมศึกษาเรียนทั้งหมด 8 คน นักเรียนแต่ละคนจะให้ข้อมูลว่า ตนอาจนัดวันเดียวกับเพื่อนคนใดชั้น เช่น นายเอกนัดวันเดียวกับนายบี ฯลฯ ซึ่งหลังจากจัดกลุ่มนักเรียนตามความคุ้นเคยแล้ว พบร่วมกันว่า สามารถแบ่งนักเรียนได้เป็น กลุ่มต่างๆ ที่มีจำนวนไม่เท่ากัน จึงเขียนโปรแกรม เพื่อคำนวณว่า มีทั้งหมดกี่วง ที่สามารถเลือกนักเรียนได้กลุ่มละ 1 คน

Input

บรรทัดที่ 1: จำนวนข้อมูลความคุ้นเคยทั้งหมด (E)

บรรทัดที่ 2 - $E+1$: เลขที่นักเรียน i บรรทัด เลขที่นักเรียนที่ i คุ้นเคยเดียวกัน ($i = 0$ ถึง 7)

ตัวอย่าง Input

5

0 1

1 4

4 2

3 5

6 7

ผลลัพธ์

16

Graph on 2D grid

ความรู้เบื้องต้น



การสร้างกราฟ

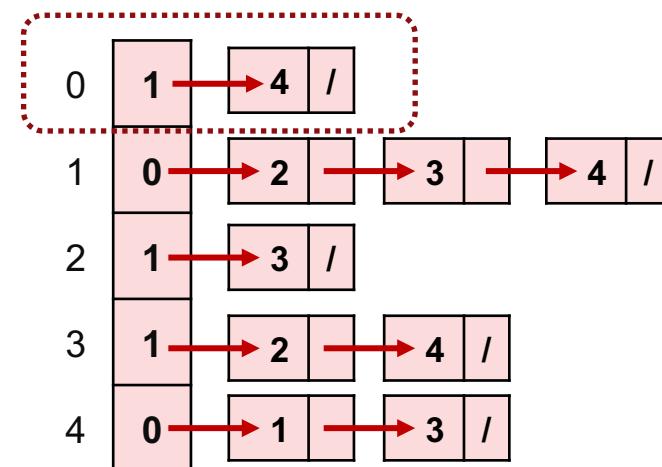
วิธีการสร้างกราฟ



1 Adjacency Matrix

	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

2 Adjacency List:



การสร้างกราฟ

วิธีการสร้างกราฟ



3 2D Grid

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

- **Node = cell**
- **Edge = การเชื่อมต่อของ cell ที่เดินถึงกันได้**
(ขึ้นอยู่กับการนิยามของโจทย์ ว่าจะเชื่อมอย่างไรบ้าง เช่น 4 กิจ คือ บน ล่าง ซ้าย ขวา เป็นต้น)

DFS on 2D grid

วิธีการท่องกราฟแบบ DFS

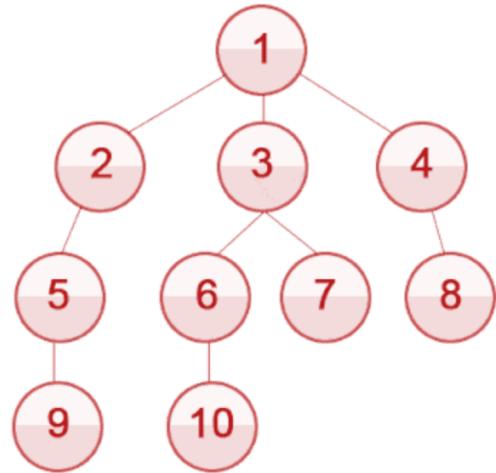


ກາຣທົ່ວໂລກຮາມ

Depth First Search or DFS on 2D grid



IIUU adjacency list



IIUU 2D grid

1	2	3
4	5	6
7	8	9

การท่องกราฟ

Depth First Search or DFS on 2D grid



IIUU adjacency list

```
void DFS(int v)
{
    visited[v] = true;
    cout << v << " ";
    for (each node in adj[v]) {
        if (!visited[node])
            DFS(adj, node, visited);
    }
}
```

IIUU adjacency list

ลงลึกไปเรื่อยๆ ตาม adjacency nodes
ของ node ที่เราเดินไป

IIUU 2D grid

รู้จำนวนทางเดินแน่นอน ตามทิศที่โจทย์กำหนด

IIUU 2D grid

```
void DFS(int current_x, int current_y){
    visited[current_x][current_y] = true;

    // up
    if(is_valid(current_x-1, current_y)){
        DFS(current_x-1, current_y);
    }

    // right
    if(is_valid(current_x, current_y+1)){
        DFS(current_x, current_y+1);
    }

    // down
    if(is_valid(current_x+1, current_y)){
        DFS(current_x+1, current_y);
    }

    // left
    if(is_valid(current_x, current_y-1)){
        DFS(current_x, current_y-1);
    }
}
```

1	2	3
4	5	6
7	8	9



เช็คว่าเดินไปได้หรือไม่

- มี cell นั้นอยู่หรือไม่
และเดินไปแล้วหรือยัง?

การท่องกราฟ

Depth First Search or DFS on 2D grid



IIUU 2D grid

1	2	3
4	5	6
7	8	9

Let's code !

[01-1-DFS-on-2D-grid.cpp]

```
void DFS(int current_x, int current_y){  
    visited[current_x][current_y] = true;  
  
    // up  
    if(is_valid(current_x-1, current_y)){  
        DFS(current_x-1, current_y);  
    }  
  
    // right  
    if(is_valid(current_x, current_y+1)){  
        DFS(current_x, current_y+1);  
    }  
  
    // down  
    if(is_valid(current_x+1, current_y)){  
        DFS(current_x+1, current_y);  
    }  
  
    // left  
    if(is_valid(current_x, current_y-1)){  
        DFS(current_x, current_y-1);  
    }  
}
```

เช็คว่าเดินไปได้หรือไม่
• มี cell นั้นอยู่หรือไม่
และเดินไปแล้วหรือยัง?

การท่องกราฟ

Depth First Search or DFS on 2D grid



- Application of Depth First Search (DFS) ใช้หาจำนวน Connected components

0	0	1	0	1	1
0	1	1	0	0	1
0	1	0	0	0	0
1	0	1	1	0	0
0	0	0	1	0	0
0	1	1	0	1	1

โจทย์กำหนดให้

- เดินได้ 4 ทิศ บน ขวา ล่าง ซ้าย
- จงหาจำนวน connected component ของ cell ที่มีค่าเป็น 1

[01-2-Connected component count.cpp]

การท่องกราฟ

Depth First Search or DFS on 2D grid



- แบบฝึกหัด

จากแบบโครงสร้างตีกที่กำหนด จงหาจำนวนห้องที่มีกั้งหมดใน
ชั้นกำหนดให้ . แทนพื้น และ # แทนกำแพงกันระหว่างห้อง

Constraints

$1 \leq n, m \leq 1000$ 亦即 $1 \leq n, m \leq 1000$

[01-DFS-ex-room-count.cpp]

Example

Input: Output:

5 8 3

#####

#..#...#

###.#.#

#..#...#

#####

การท่องกราฟ

Depth First Search or DFS on 2D grid



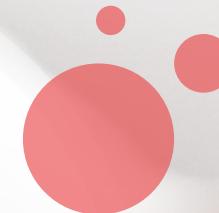
- แบบฝึกหัด

กำหนดให้หมายเลขอต่ำสุดของกราฟ แทนเส้นของกราฟ เป็นเส้นที่ไม่ซ้ำกัน จงเขียนโปรแกรมโดยใช้ DFS เพื่อหาว่ากราฟเป็นเส้นที่ติดกันกว้างที่สุด และมีขนาดกี่ช่อง

กำหนดให้

พื้นที่ติดกัน หมายถึงพื้นที่ที่เดินถึงกันได้ใน 4 ทิศ
ได้แก่ บน ขวา ล่าง ซ้าย

1	4	4	4	4	4	3	3	1
2	1	1	4	3	3	1	1	1
3	2	1	1	2	3	2	1	1
3	3	2	1	2	2	2	2	2
3	1	3	1	1	4	4	4	4
1	1	3	1	1	4	4	4	4



พอย่างนี้ก่อน



BFS on 2D grid

วิธีการท่องกราฟแบบ BFS

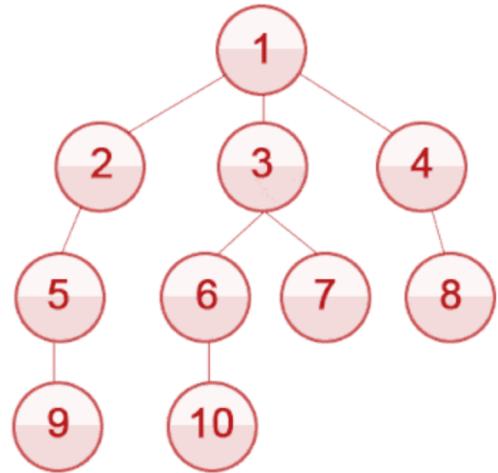


ກາຣທົ່ວໂລກຮາມ

Breath First Search or BFS on 2D grid



IIUU adjacency list



IIUU 2D grid

1	2	3
4	5	6
7	8	9

การท่องกราฟ

Breath First Search or BFS on 2D grid

```
void BFS(int s)
{
    queue<int> q;
    visited[s] = true;
    q.push(s);

    while(!q.empty()) {
        s = q.front();
        cout << s << " ";
        q.pop();
        for (each node in adj(s)) {
            if (!visited[node]) {
                visited[node] = true;
                q.push(node);
            }
        }
    }
}
```

||UU adjacency list

เดินแนวกว้างไปเรื่อย ๆ ตาม adjacency nodes
ของ node ที่เราเดินไป

||UU 2D grid

รู้จำนวนทางเดินแน่นอน ตามทิศที่โจทย์กำหนด

```
void BFS(int src_x, int src_y){
    q.push(make_pair(src_x, src_y));
    visited[src_x][src_y] = true;

    while(!q.empty()){
        int current_x = q.front().first;
        int current_y = q.front().second;
        cout << G[current_x][current_y] << "\n" ;
        q.pop();

        // up
        if(is_valid(current_x-1, current_y)){
            q.push(make_pair(current_x-1, current_y));
            visited[current_x-1][current_y] = true;
        }
        // right
        if(is_valid(current_x, current_y+1)){
            q.push(make_pair(current_x, current_y+1));
            visited[current_x][current_y+1] = true;
        }
        // down
        if(is_valid(current_x+1, current_y)){
            q.push(make_pair(current_x+1, current_y));
            visited[current_x+1][current_y] = true;
        }
        // left
        if(is_valid(current_x, current_y-1)){
            q.push(make_pair(current_x, current_y-1));
            visited[current_x][current_y-1] = true;
        }
    }
}
```

1	2	3
4	5	6
7	8	9



ກາຣທົວກຣາວ

Breath First Search or BFS on 2D grid

Let's code !

[02-BFS-on-grid.cpp]

```
void BFS(int src_x, int src_y){  
    q.push(make_pair(src_x, src_y));  
    visited[src_x][src_y] = true;  
  
    while(!q.empty()){  
        int current_x = q.front().first;  
        int current_y = q.front().second;  
        cout << G[current_x][current_y] << "\n" ;  
        q.pop();  
  
        // up  
        if(is_valid(current_x-1, current_y)){  
            q.push(make_pair(current_x-1, current_y));  
            visited[current_x-1][current_y] = true;  
        }  
        // right  
        if(is_valid(current_x, current_y+1)){  
            q.push(make_pair(current_x, current_y+1));  
            visited[current_x][current_y+1] = true;  
        }  
        // down  
        if(is_valid(current_x+1, current_y)){  
            q.push(make_pair(current_x+1, current_y));  
            visited[current_x+1][current_y] = true;  
        }  
        // left  
        if(is_valid(current_x, current_y-1)){  
            q.push(make_pair(current_x, current_y-1));  
            visited[current_x][current_y-1] = true;  
        }  
    }  
}
```

1	2	3
4	5	6
7	8	9



การท่องกราฟ

Breath First Search or BFS on 2D grid



- Application of Breadth First Search (BFS) ໃຫ້ Shortest path ແລ້ວ Minimum number of steps

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

Minimum number of steps from 1 to 8 = ?

[02-BFS-on-grid-shortest.cpp]

การท่องกราฟ

Breath First Search or BFS on 2D grid



- แบบฝึกหัด

นักเรียนเดินอยู่ในป่าแห่งหนึ่ง ซึ่งมีพื้นที่ดังกราฟที่ได้รับ ขนาด $N \times N$ โดยกำหนดให้ S คือจุดเริ่มต้น P คือทางเดิน และ T คือตัวเป้า ที่ไม่สามารถเดินผ่านหรือกระโดดข้ามได้ จงเขียนโปรแกรมเพื่อหาว่า จำนวนก้าวเดินที่น้อยที่สุด ที่จะเดินจาก S ไป E เป็นเท่าไร

Input

5

S P P P P

T P T P P

T P P P P

P T E T T

P T P T T

Output

5

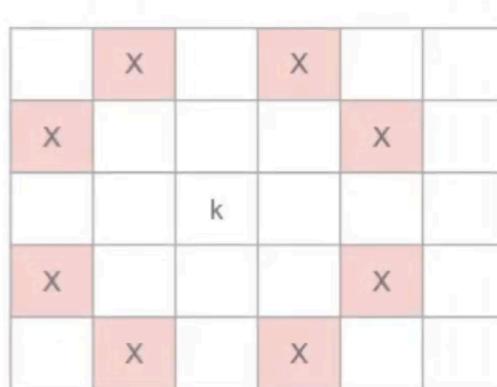
ກາຣທົ່ວກຮາວ

Breath First Search or BFS on 2D grid



- ແບບືກຫັດ

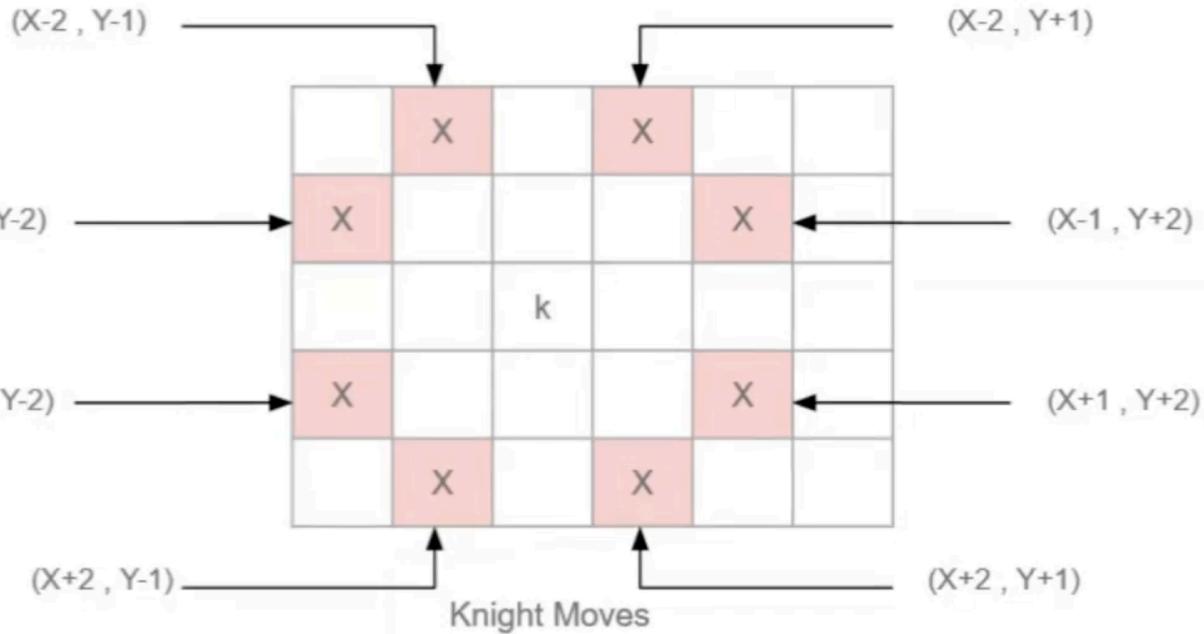
You are given a Maze of size $N \times M$, there is a Knight on point(x , y) and a target cell (a , b) , find minimum number of steps to reach the target



Knight Moves

ກາຣທົວການ

Breath First Search or BFS on 2D grid



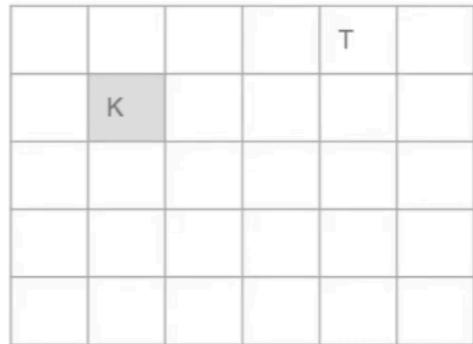
$dx[] = \{-2, -1, 1, 2, 2, 1, -1, -2\}$
 $dy[] = \{1, 2, 2, 1, -1, -2, -2, -1\}$

ກາຣທົ່ວການ

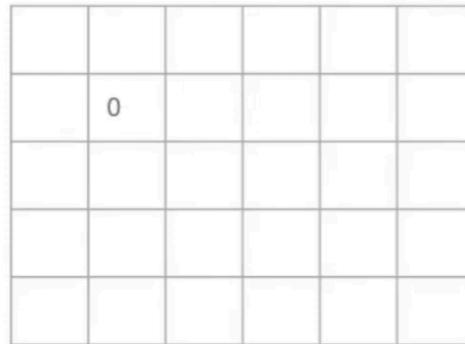
Breath First Search or BFS on 2D grid



Graph



distance



Queue

Current:

การท่องกราฟ

Breath First Search or BFS on 2D grid



แบบฝึกหัด

ส้มเน่า (Rotten Oranges)

ช่วงstanเก็บส้มใส่ตะกร้าที่มีขนาด $M \times N$ ช่อง แล้วว่างที่ไว้ เข้าพบว่ามีอ่อนเสื่อมไป 1 วัน ส้มที่เน่าจะส่งผลให้ส้มสดที่อยู่ติดกันในแนวตั้งและแนวนอน เน่าตามไปด้วย

ให้ขนาดตะกร้าส้ม $M \times N$ ที่ค่าแต่ละช่อง มีค่าเป็น 0, 1 หรือ 2 ซึ่งความหมายแต่ละตัวเลขมีดังนี้

0 : ช่องว่าง

1 : ช่องที่มีส้มสด

2 : ช่องที่มีส้มเน่า

กล่าวคือ ส้มเน่าที่ช่อง $[i, j]$ สามารถเน่าส้มสดอื่นๆ ที่ช่อง $[i-1, j], [i+1, j], [i, j-1], [i, j+1]$ (ขึ้น, ลง, ซ้ายและขวา)

ภายใน 1 วัน

ข้อมูลนำเข้า

บรรทัดแรกมีจำนวนเต็ม T ($1 \leq T \leq 100$) แสดงถึงจำนวนกราฟทดสอบ

แต่ละกราฟทดสอบมีจำนวนเต็มสองจำนวน M ($1 \leq M \leq 100$) และ N ($1 \leq N \leq 100$) โดยที่ M คือจำนวนแถวและ N คือจำนวนคอลัมน์ในอาร์เรย์ A

บรรทัดถัดไปมีช่องว่างคั่นค์ประจำ $M * N$ และแสดงถึงค่าในแต่ละเซลล์ในอาร์เรย์

ข้อมูลส่งออก

พิมพ์จำนวนเต็มซึ่งแสดงจำนวนวันต่ำสุดที่ใช้ในการทำให้ส้มเน่าทั้งหมด (พิมพ์ -1 ถ้าส้มไม่สามารถเน่าทั้งหมดได้)

Example:

Input:

2

3 5

2 1 0 2 1

1 0 1 2 1

1 0 0 2 1

3 5

2 1 0 2 1

0 0 1 2 1

1 0 0 2 1

Output:

2

-1

การท่องกราฟ

Breath First Search or BFS on 2D grid

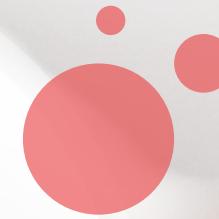
- อธิบายขั้นตอนวิธีทำ

2	1	0	2	1
1	0	1	2	1
1	0	0	2	1

queue:

ndays =





Thank you

ສູ້າ ບະຄະທຸກຄນ