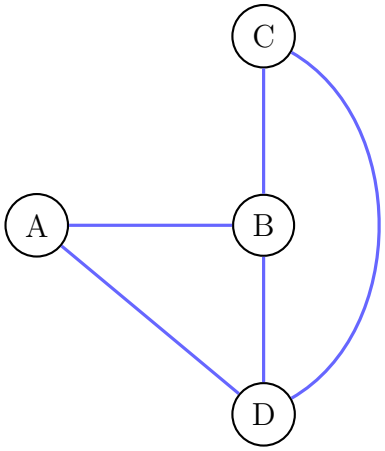
Lab08: Graph I

19 มีนาคม 2568

"ทุกต้นไม้ (Tree) คือกราฟ (Graph) แต่ไม่ใช่ทุกกราฟที่เป็นต้นไม้"

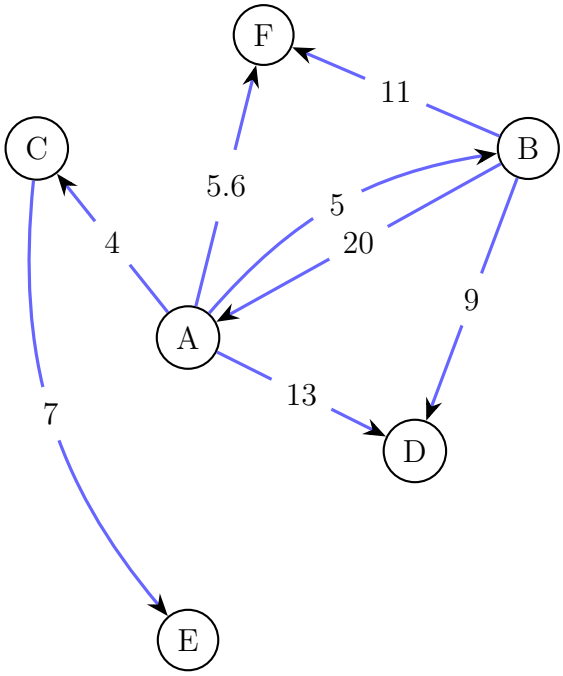
วันนี้ในพาร์ทแรก เรามาสร้างกราฟ (Graph) กัน ซึ่งกราฟก็เป็น Non-linear data structure ที่สำคัญอีกตัวหนึ่งที่มีการประยุกต์ใช้ในหลาย ๆ เรื่อง เช่นการทำแผนที่ การหาเส้นทางที่สั้นที่สุด และอื่น ๆ อีกมากมาย

ตัวอย่างเช่น: กำหนดให้โครงสร้างกราฟดังต่อไปนี้



คำถามพิเศษ

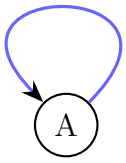
กราฟแบบนี้เป็นกราฟประเภทไหน _____
กราฟนี้มีจำนวนกี่ Vertices _____
กราฟนี้มีจำนวนกี่ Edges _____
$deg(A)$ มีค่าเท่าใด _____



คำถามพิเศษ

โหนด A มี In-degree เท่าใด _____
Edge ที่มี Weight มากที่สุดคือ Edge ใด _____
ผลรวม Weight ของ Out-degree ของโหนด B เป็นเท่าใด _____

หากว่าในกราฟมีลูป (Loop) *(หมายถึงโหนดที่ชี้ไปที่โหนดตัวเอง)* เช่น



ในกรณีของ Directed graph เราจะได้ว่า

$$deg^{+}(A) = deg^{-}(A) = 1$$
$$deg(A) = 2$$

หรือว่าในกรณี Undirected graph:

เราจะได้ว่า

$$deg(Z) = 2$$

วันนี้เราจะมาสร้างกราฟ และเก็บกราฟในรูปแบบของ **Adjacency list** ซึ่งจะมีข้อดีในเรื่องของความยืดหยุ่นกว่าตัว Adjacency matrix โดยจะใช้โค้ดที่อาจารย์ได้เขียนให้ดูในสัปดาห์ที่แล้วมาปรับแก้เล็กน้อยให้สามารถรองรับประเภทของกราฟได้หลาย ๆ ประเภทนั่นเอง

เริ่มจากคลาสหลักซึ่งก็คือคลาส **Graph** *(Syntax ต่อไปนี้อาจจะไม่คุ้นเคยเท่าไหร่ ขอให้ฟังการอธิบายจาก TA เสริมด้วย)*

```
public abstract class Graph{
    protected int nVertices;
    protected Map<String, List<Edge>> adjacencyList;
    protected boolean isWeighted;
}
```

ในคลาส **Graph** ก็จะมี Operation หลัก ๆ ดังนี้:

1. **addNode()**: เพิ่มโหนดเข้าไปอยู่ใน Adjacency list ของกราฟ
2. **addEdge()**: เพิ่มเส้นเชื่อมเข้าไปใน Adjacency list ของโหนดนั้น ๆ ในกราฟ

ตัว Adjacency list เราจะเก็บในรูปแบบของ List ของ Object ที่เป็นคลาสชื่อว่า **Edge** โดยโครงสร้างของคลาส **Edge** เป็นดังนี้

```
class Edge{
    String destination;
    int weight;
}
```

และมี Subclass ของคลาส **Graph** คือคลาส **DirectedGraph** และ **UndirectedGraph** โดยมีการเขียนโครงสร้างของคลาสดังนี้

```
class DirectedGraph extends Graph{

}
```

```
class UndirectedGraph extends Graph{

}
```

งานของนักศึกษาวันนี้

จงสร้างกราฟขึ้นมาโดยให้เก็บและแสดงผลในรูปของ Adjacency list ที่สามารถเก็บกราฟได้ทั้งหมด 4 ประเภทดังนี้

1. Directed unweighted graph
2. Directed weighted graph
3. Undirected unweighted graph
4. Undirected weighted graph

เมื่อนักศึกษาสร้างกราฟมาแล้ว ให้เราใส่ข้อมูลของเส้นเชื่อมเป็นข้อมูลนำเข้า (Input) สมมติว่ากราฟมีเส้นเชื่อม (Edge) ทั้งหมด E เส้นเชื่อม แล้วหลังจากนั้นก็เป็นการใส่ข้อมูลของเส้นเชื่อมโดยอยู่ในรูปแบบดังนี้

- ถ้ากราฟเป็นแบบ Unweighted ให้ใส่ข้อมูลในรูปของ `src dest`
- ถ้ากราฟเป็นแบบ Weighted ให้ใส่ข้อมูลในรูปของ `src dest weight`

เมื่อรับข้อมูลโหนดเรียบร้อยแล้ว ให้ทำการรับจำนวนเต็ม n เพื่อบอกจำนวนจุดยอด (Vertices) ที่อยากทราบข้อมูล

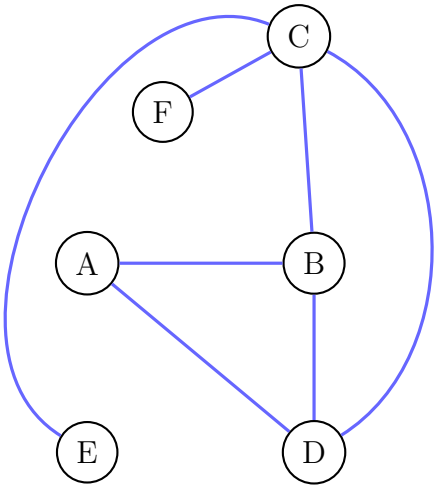
หลังจากนั้นให้ใส่ชื่อของโหนดที่อยากทราบข้อมูล เมื่อใส่ข้อมูลแล้วจะได้ผลดังนี้

- ถ้าเป็น Directed graph จะแสดงผล $deg^+(N_i)$ และผลรวมของ Weight ใน $deg^+(N_i)$
- ถ้าเป็น Undirected graph จะแสดงผล $deg(N_i)$ และผลรวมของ Weight ใน $deg(N_i)$

ทั้งนี้ หากกราฟเป็นแบบ Unweighted เราจะแสดงเพียงแค่ $deg^+(N_i)$ หรือ $deg(N_i)$ เท่านั้น

รูปของกราฟที่จะทดสอบอยู่ในหน้าถัดไป

กราฟที่ 1



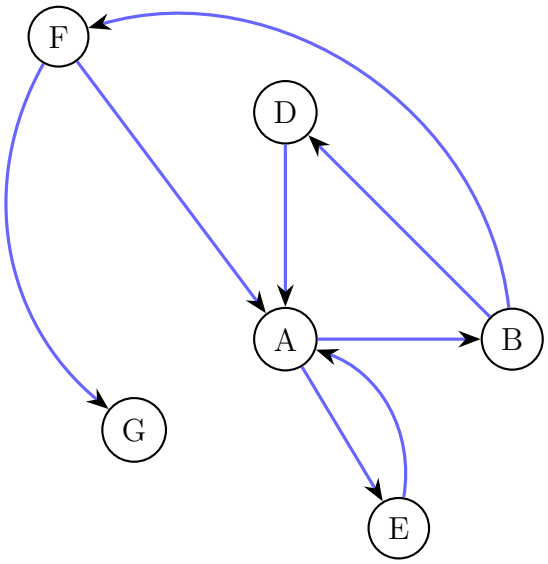
ส่วนการสร้างกราฟ

```
7
A B
A D
B C
B D
C F
C E
C D
```

ส่วนการเรียกข้อมูลของกราฟ

```
2
A // output is 2
C // output is 4
```

กราฟที่ 2



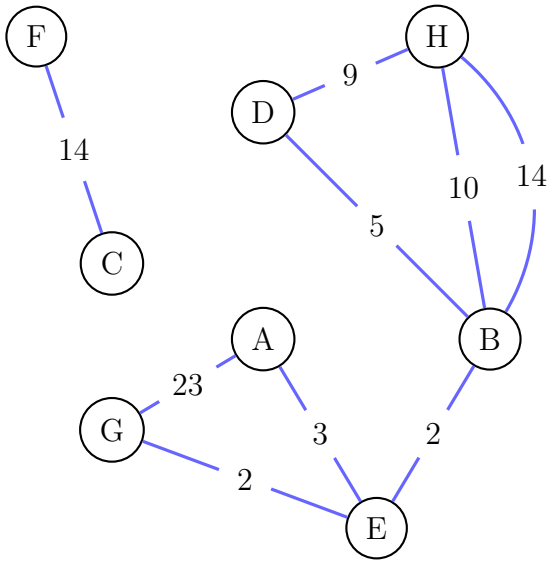
ส่วนการสร้างกราฟ

```
8
A B
A E
B D
B F
D A
E A
F A
F G
```

ส่วนการเรียกข้อมูลของกราฟ

```
3
E // output is 1
A // output is 2
G // output is 0
```

กราฟที่ 3



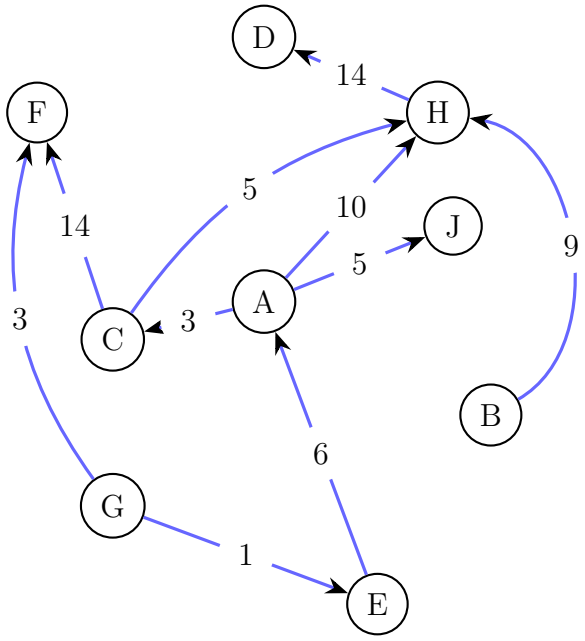
ส่วนการสร้างกราฟ

```
9
A E 3
A G 23
B E 2
B H 10
B D 5
C F 14
D H 9
E G 2
H B 14
```

ส่วนการเรียกข้อมูลของกราฟ

```
3
E // output is 3 7
H // output is 3 33
F // output is 1 14
```

กราฟที่ 4



ส่วนการสร้างกราฟ

```
10
A H 10
A J 5
A C 3
B H 9
C H 5
C F 14
E A 6
G E 1
G F 3
H D 14
```

ส่วนการเรียกข้อมูลของกราฟ

```
4
A // output is 3 18
J // output is 0 0
C // output is 2 19
H // output is 1 14
```