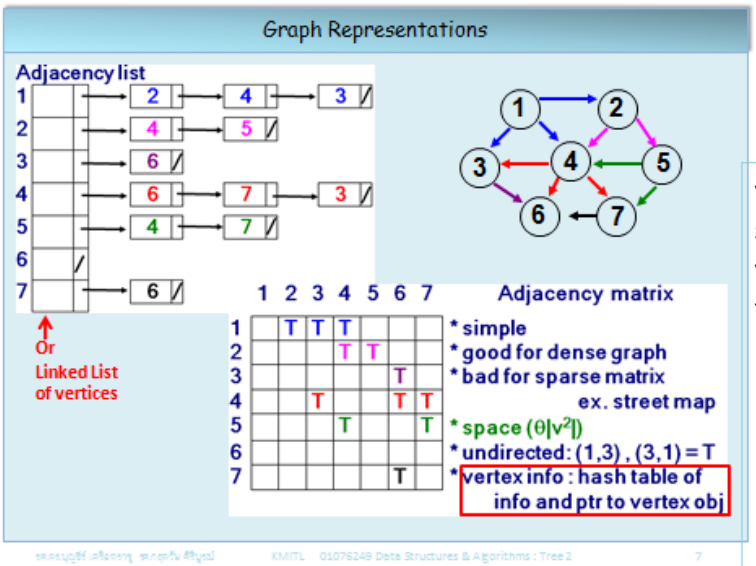
Lab 10 : Graph

วัตถุประสงค์ เขียน Graph Application Shortest Path ของ Dijkstra

ทฤษฎี กราฟประกอบด้วย set ของ vertices (nodes) และ set ของ arcs(edges) ในรูปแสดง data structures ของ arcs Application ของกราฟมีมากมาย เช่น การหาระยะที่สั้นที่สุด (shortest path) ข้างล่างแสดง algorithm ในการหา Shortest Path ของ Dijkstra ซึ่งใช้ Greedy Algorithm เลือก vertex ที่รู้ระยะทางที่สั้นที่สุดขณะนั้นๆ

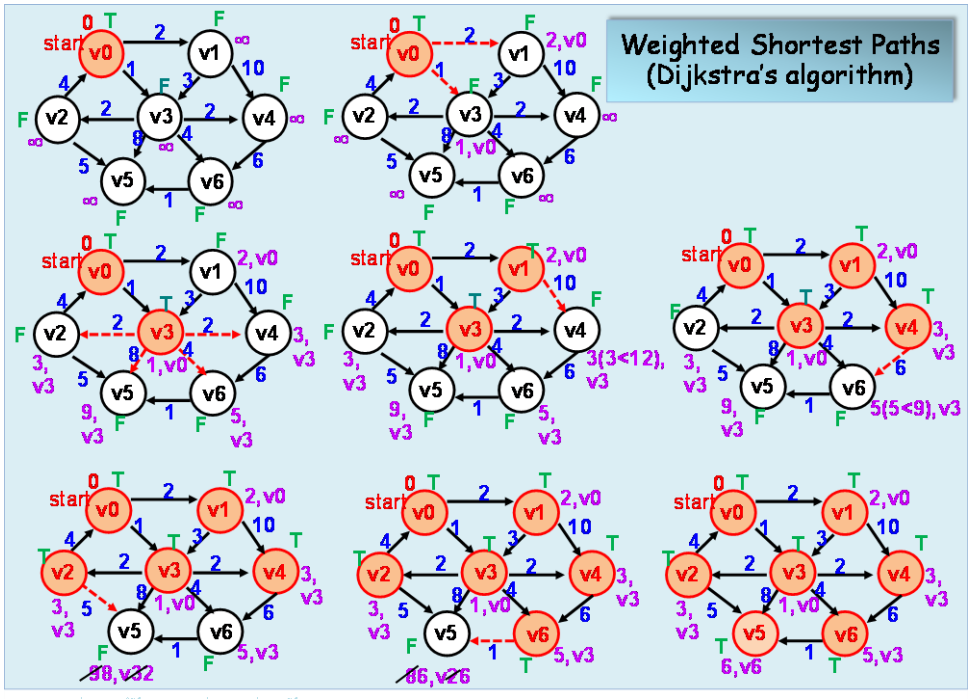


สำหรับ vertex v :
distance = ระยะจากจุด start ไปยัง vertex นั้นๆ
known เป็นจริง เมื่อทราบระยะ distance ที่สั้นที่สุดแล้ว
path = vertex ก่อนหน้ามันใน shortest path

```
vertices ทั้งหมด : known = flase;
start_vertex :  distance = 0;
vertices อื่นๆ : distance = ∞;
for( ; ;)
    v = vertex ที่มี dist. น้อยที่สุด ที่ known ยังเป็น false
    if (ไม่มี v )
        break;
    v.known = true;
    for each w adjacent to v ซึ่งยังไม่ถูก process
        if (w.dist > v.dist + weight(vw))
            ปรับ w.dist เป็นค่าใหม่ซึ่งน้อยกว่า
            w.path = v;
```

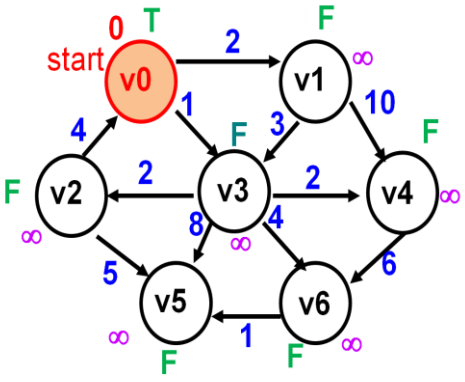
การทดลอง

1. สร้าง data structure ของกราฟ
   a. เพื่อเก็บข้อมูล distance, known และ path ของ vertex ทั้งหมด
   b. เก็บ edge ทุก edge และ weight ของมัน
2. ลองใส่ input graph โดยให้กำหนดเอง เขียน routine print() เพื่อทดสอบ input ว่าใส่ถูกต้องหรือไม่
3. หา shortest path และ เขียน `printPath ()` แสดง shortest path จาก Start vertex ไปยัง vertex ที่ต้องการ



Code อาจใช้ data structure ได้หลายแบบ ในตัวอย่างข้างล่างใช้ adjacency matrix เก็บ weight (นักศึกษาอาจใช้ adjacency list ก็ได้) เพื่อความสะดวกไม่ได้เก็บชื่อ vertex ใช้เลขเป็นชื่อ โดยใช้ข้อมูลตามรูป ส่วน distance known และ path ใช้ python list โดย known เก็บเฉพาะ vertex ที่ทราบระยะสั้นที่สุดแล้ว

	0	1	2	3	4	5	6
0		2		1			
1				3	10		
2	4					5	
3			2		2	8	4
4							6
5							
6						1	




```
# Dijkstra's algorithm for shortest path
def printPath(start, toV, distance, path):
    print('From V',start, ' to V', toV, sep = '')
    print('\tShortest path =', distance[toV])
    print('\tpath :', end = ' ' )
    p = path[toV]
    while p is not None:
        print('V', p, sep = '', end = ', ')
        p = path[p]
    print('\n')
#         0 1 2 3 4 5 6
adj = [  [0,2,0,1,0,0,0], # 0
         [0,0,0,3,10,0,0], # 1
         [4,0,0,0,0,5,0], # 2
         [0,0,2,0,2,8,4], # 4
         [0,0,0,0,0,0,6], # 5
         [0,0,0,0,0,0,0], # 6
         [0,0,0,0,0,1,0]] # 6

n = 7  # total nodes
known = []  # เก็บเฉพาะ vertex ที่รู้ระยะสั้นที่สุดแล้ว
distance = [float('inf')]*n  #set to infinity
path = [None]*n

# set start vertex
start = 0
distance[start] = 0
while len(known) != n:
    # greedy : find vertex of smallest distance
    min = float('inf')
    for i in range(n):
         if i not in known and distance[i] < min:
             min_i = i # index
             min = distance[i]
    known.append(min_i)
    for i in range(n): #for every vertex i that
        w = adj[min_i][i]
        if w and distance[i] > min + w:
           # adjacent to min_i(has weight > 0)
           distance[i] = min + w
           path[i] = min_i
toV = 5  # to vertex

printPath(start,toV, distance, path)
```