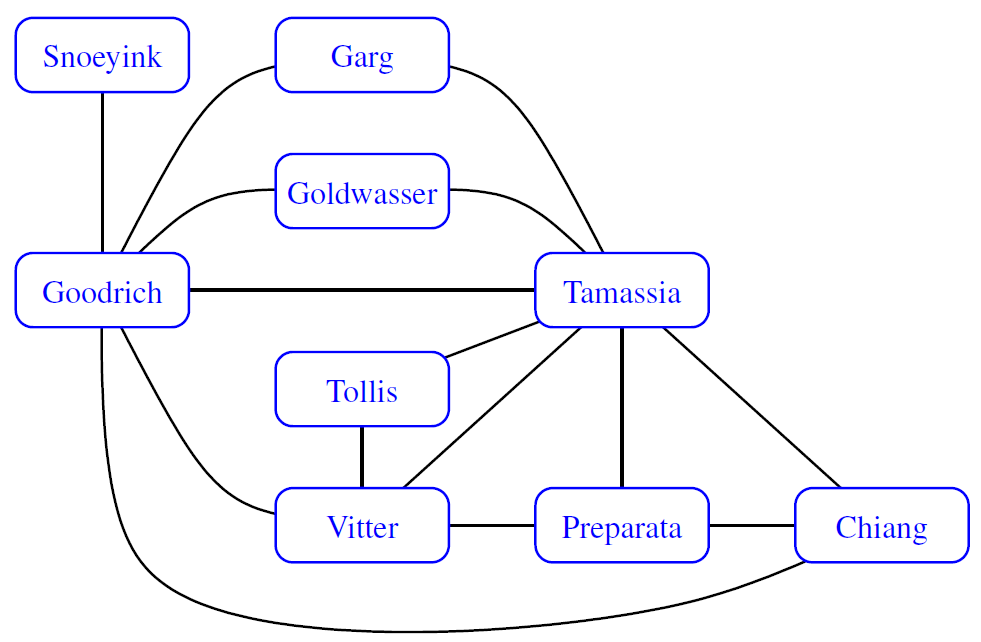
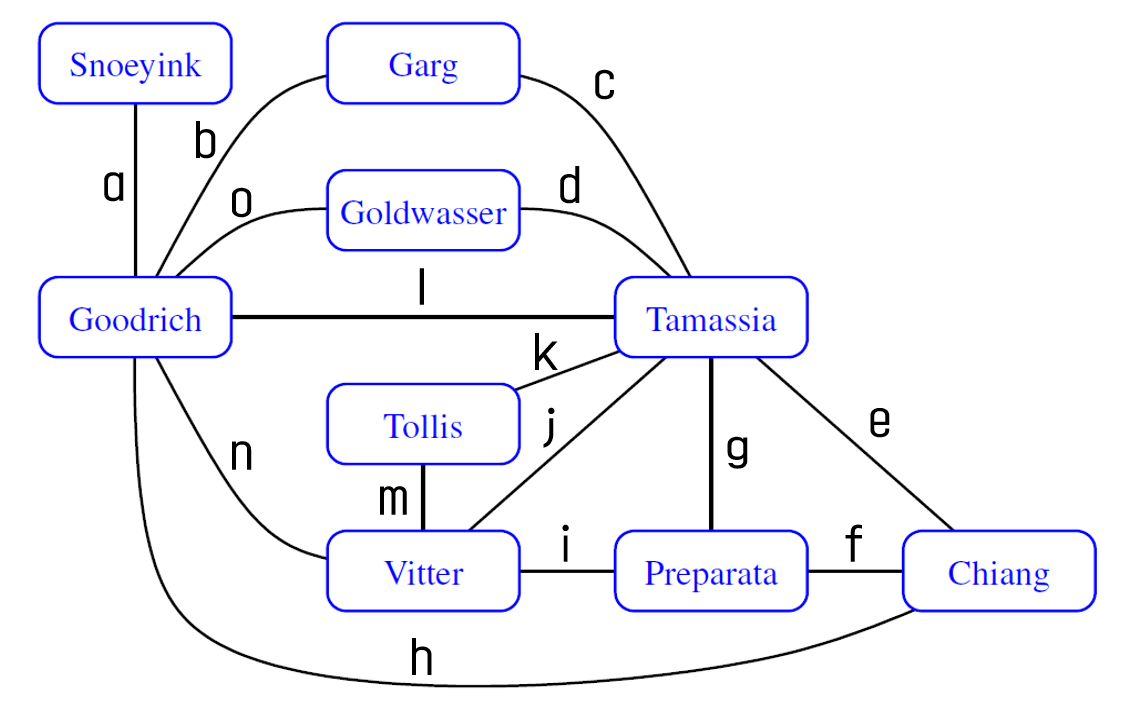
**Data Structure Assignment 4**

**201911013 곽현우**

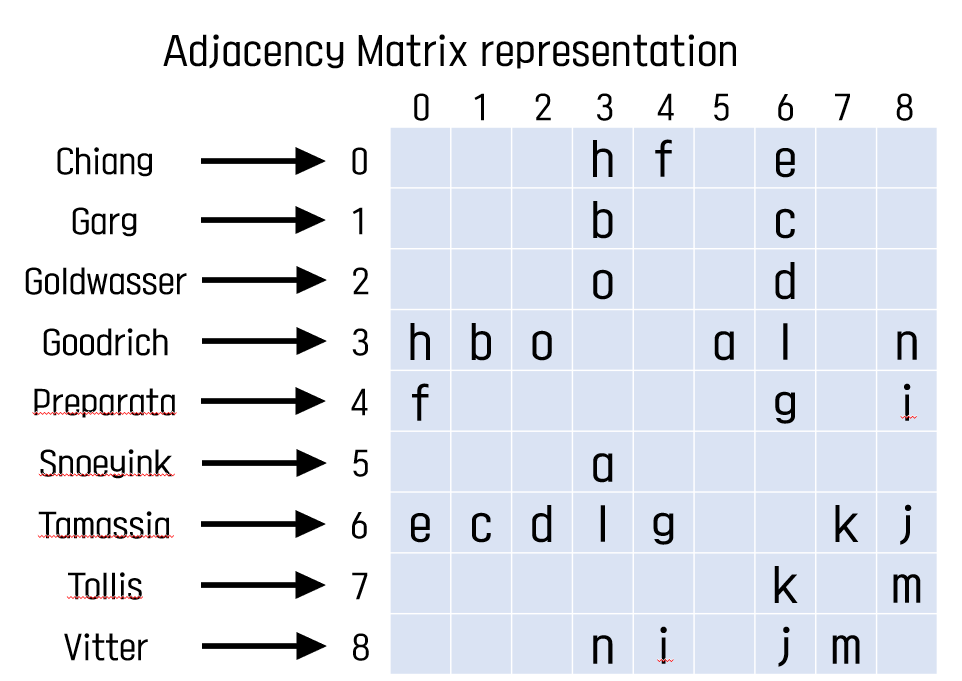
**A9 - Graphs (15+3 points)**

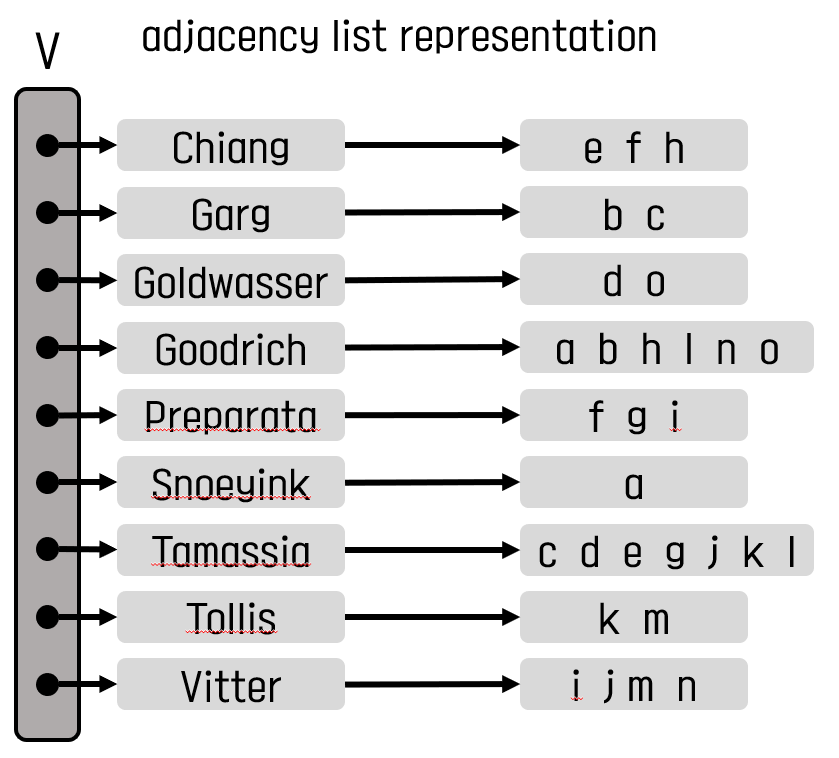
**A9-1 (2 points)** (Enumerate the vertices in alphabetical order.)



Edge의 이름은 다음과 같이 하였다.

1. Draw an adjacency matrix representation of the undirected graph shown above.



1. Draw an adjacency list representation of the undirected graph shown above. 

**A9-2 (3 points)**

Would you use the adjacency matrix structure or the adjacency list structure in each of the following cases? Justify your choice.

1. The graph has 10,000 vertices and 20,000 edges, and it is important to use as little space as possible.
2. The graph has 10,000 vertices and 20,000,000 edges, and it is important to use as little space as possible.
3. You need to answer the query get edge(u,v) as fast as possible, no matter how much space you use.

Vertices의 개수를 n개, Edge의 개수를 m개라 할 때, Adjacent matrix structure의 저장공간은 O(), Adjacent List Structure의 저장공간은 O(n+M)이므로 1번의 경우에는 보다 10000+20000이 더 작기 때문에 Adjacent List Structure이 적절하고 2번의 경우도 보다 20010000가 더 작기 때문에 Adjacenct List Structure가 더 적절하다.

3번의 경우 가장 빠르게 Edge(u,v)를 찾기 위해서는 Adjacent Matrix Structure을 사용하는 것이 효율적이다. 왜나하면 u,v 각각에 대응되는 행과 열을 통해 Matrix에 저장된 Edge를 찾는데 걸리는 시간이 O(1)으로 Adjacent List로 구현할 때 O(min(deg(u),deg(v)))에 비해 매우 빠르기 때문이다.

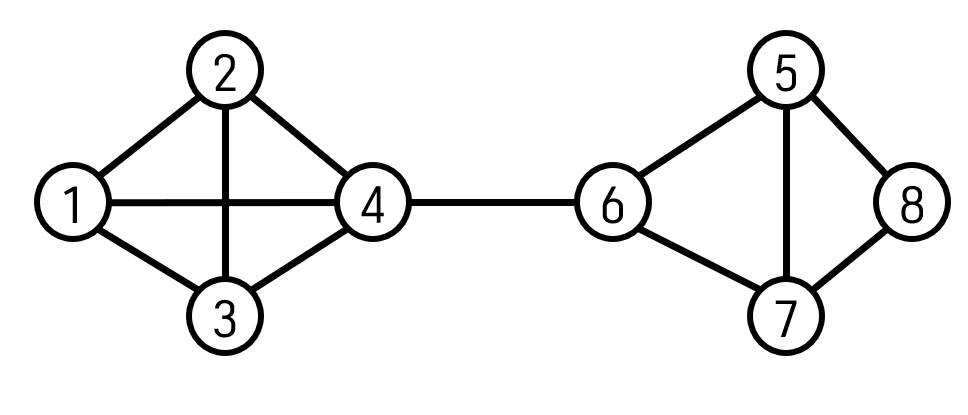
**A9-3 (3 points)**

Let G be an undirected graph whose vertices are the integers 1 through 8, and let the adjacent vertices of each vertex be given by the table below:

| **vertex** | **adjacent vertices** |
| --- | --- |
| 1 | (2, 3, 4) |
| 2 | (1, 3, 4) |
| 3 | (1, 2, 4) |
| 4 | (1, 2, 3, 6) |
| 5 | (6, 7, 8) |
| 6 | (4, 5, 7) |
| 7 | (5, 6, 8) |
| 8 | (5, 7) |

Assume that, in a traversal of G, the adjacent vertices of a given vertex are returned in the same order as they are listed in the table above.

1. Draw G.



1. Give the sequence of vertices of G visited using a DFS traversal starting at vertex 1.

{1, 2, 4, 6, 5, 8 ,7, 3}

1. Give the sequence of vertices visited using a BFS traversal starting at vertex 1.

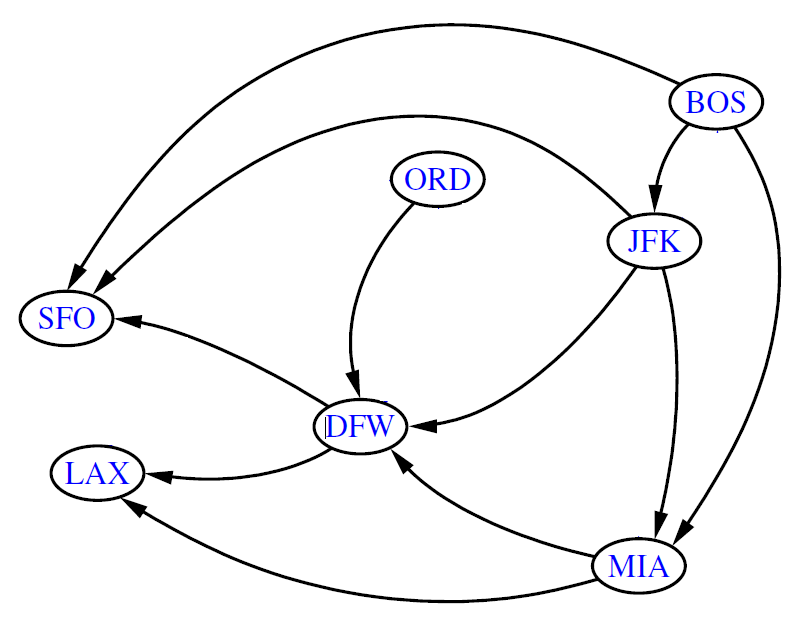
{1, 2, 4, 3, 6, 5 ,7, 8}

**A9-4 (2 point)**

How many edges are in the transitive closure of a graph that consists of a simple directed path of n vertices? Explain why.

Vertex 1,2,…n이 차례로 Simple directed path를 이루는 경우를 생각해보자. Vertex 1은 Incoming Edge가 없고 Vertex n은 Outgoing Edge가 없기 때문에 나머지 Vertices들에 대하여 생각해본다. Vertex2는 1과 3을 연결해주기 때문에 1과 3을 연결하는 Transitive closure가 생긴다. Vertex3 의 경우는 2와 4, 1과 4를 연결해주기 때문에 2개의 Transitive closure가 생긴다. 이렇게 Vertex k의 경우 Trasitive Closure가 k-1개 생긴다.(1<k<n) 따라서 모든 operation이 수행된 후의 Edge의 개수는 (1+2+…+n-2)(Transitive Closure)+n-1(원래있던 Edge의 개수) = 개 이다.

**A9-5 (1 point)**

Compute a topological ordering for this directed graph.(The Following operations are operating in the copy of above graph)

1) Start DFS From BOS

2) LAX is reached, and LAX has no outgoing edges. So LAX is no.7(the number of vertices is 7)

Delete incident edges.

3) Next, SFO is reached. So SFO is no.6. Then, Delete incident edges.

4) Next, DFW is reached. So DFW is no.5. Then, Delete incident edges.

5) Next, MIA is reached. So MIA is no.4. Then, Delete incident edges.

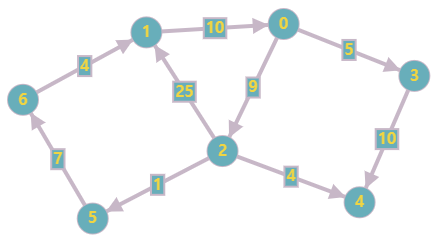
6) Next, JFK is reached. So JFK is no.3. Then, Delete incident edges.

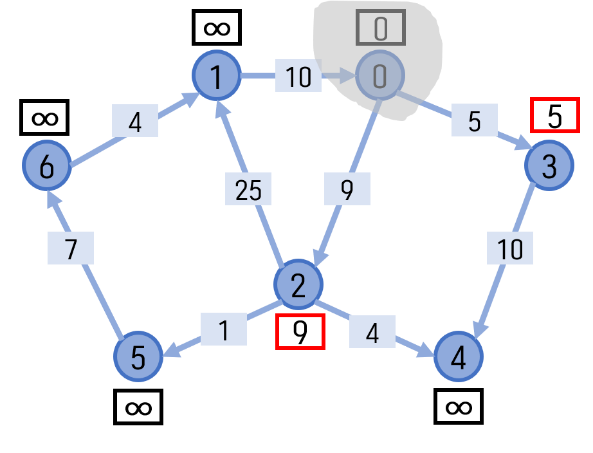
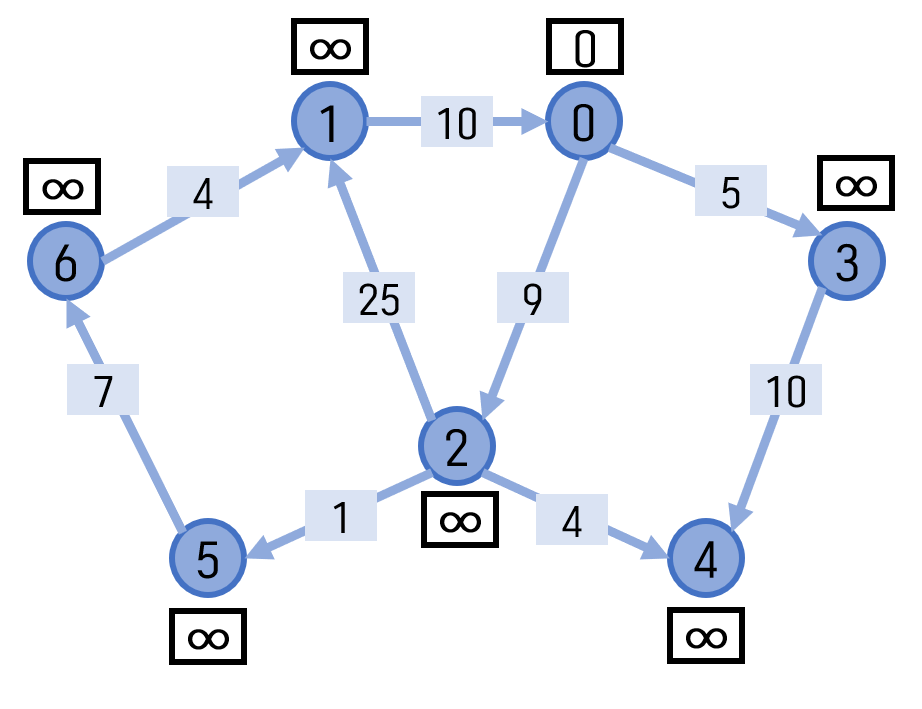
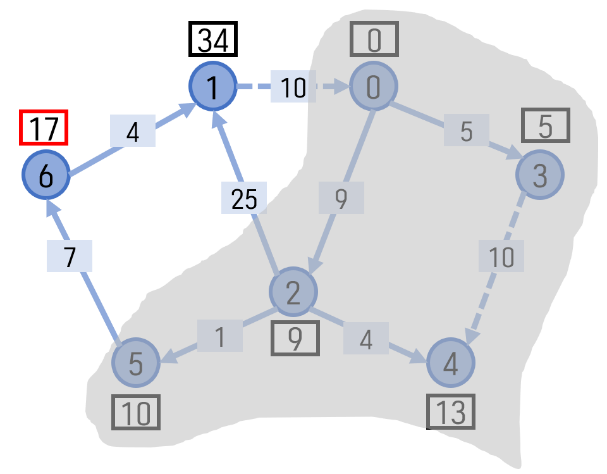
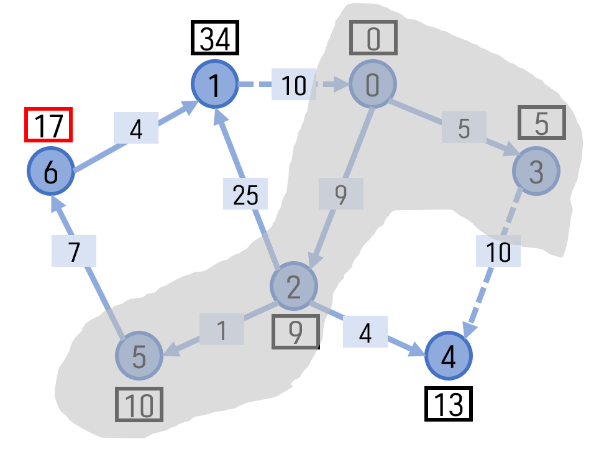
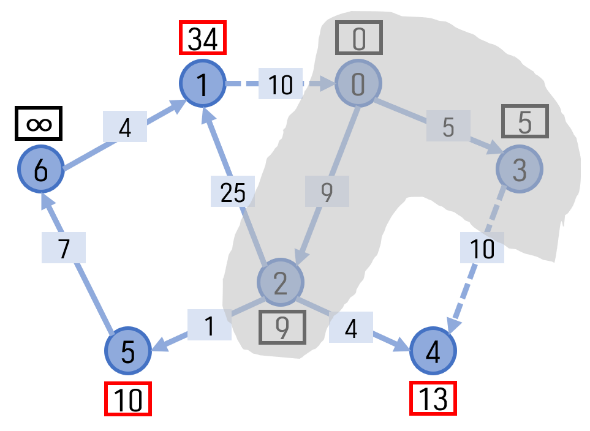
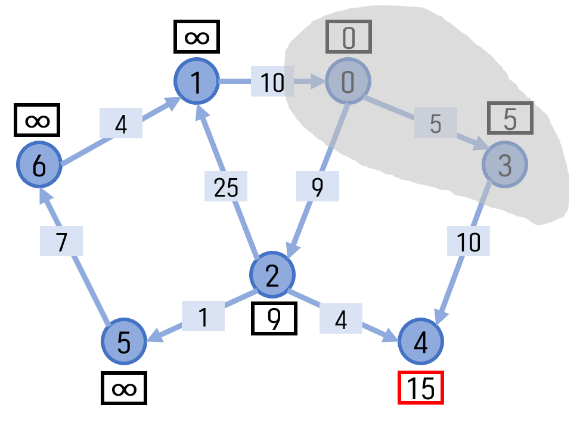
7) Next, BOS is reached. So BOS is no.2. Then, Delete incident edges.

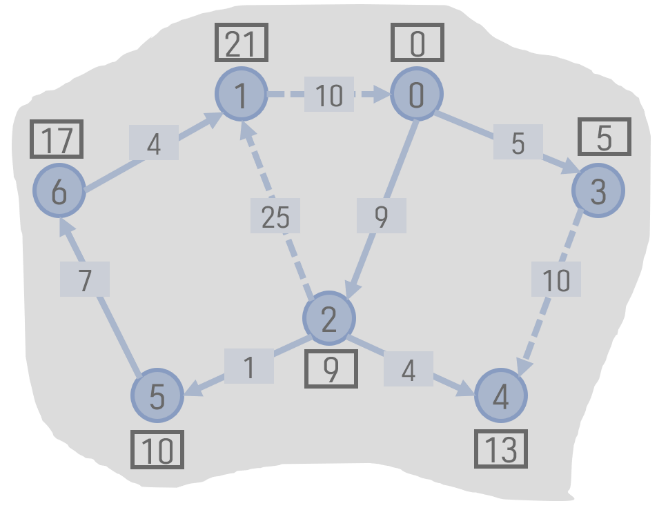
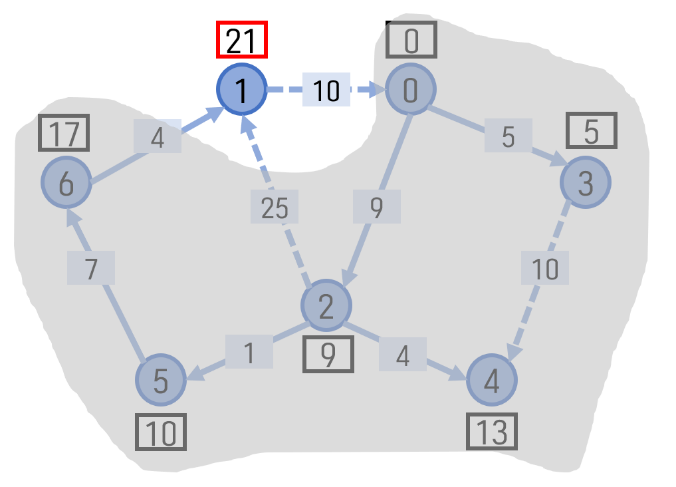
8) The Last vertex, ORD is no.1

So a topological ordering in this directed graph is ORD, BOS, JFK, MIA, DFW, SFO, LAX

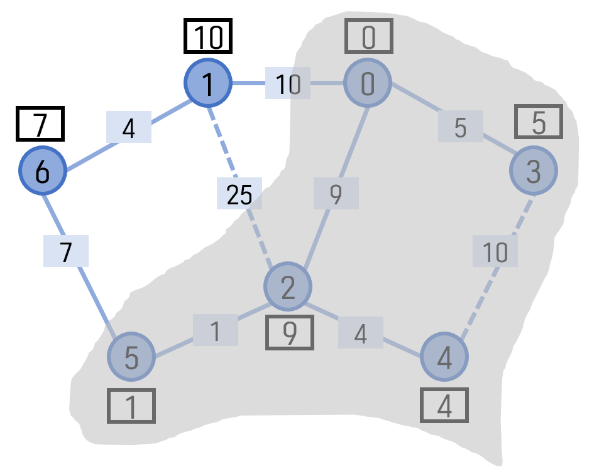
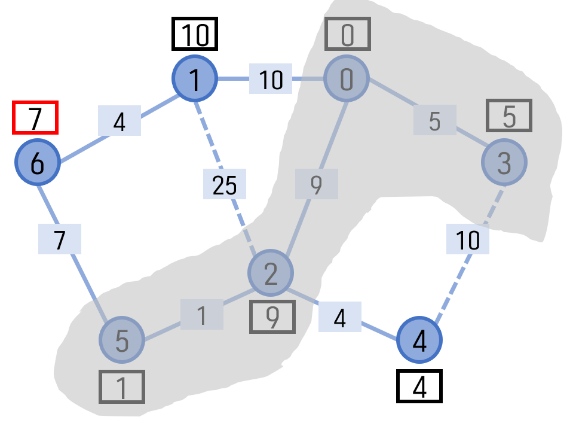
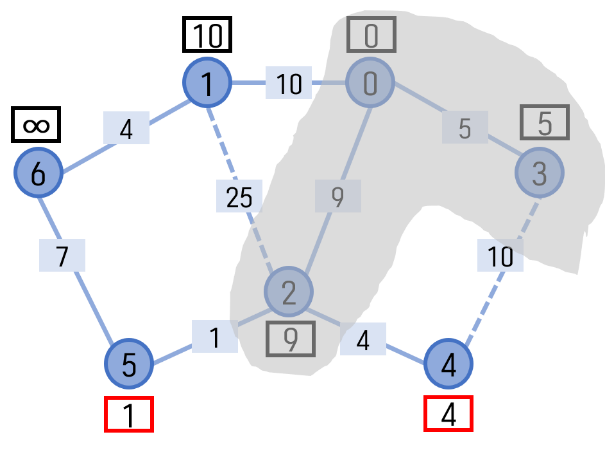
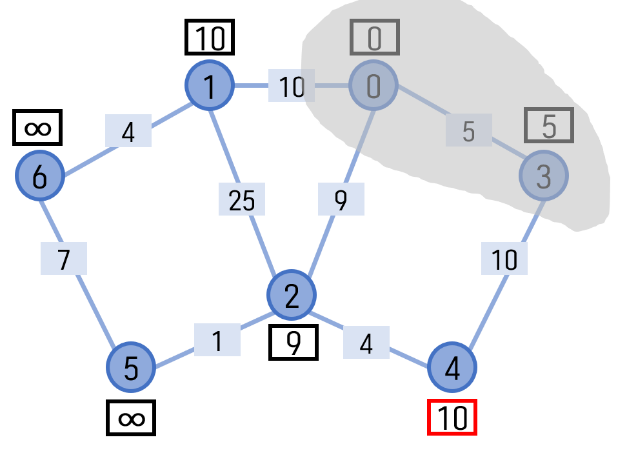
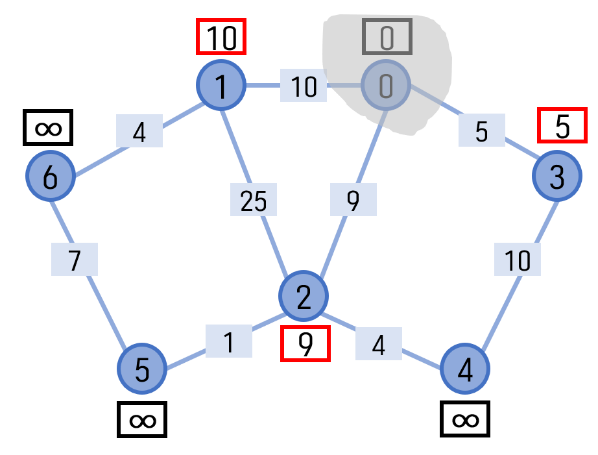
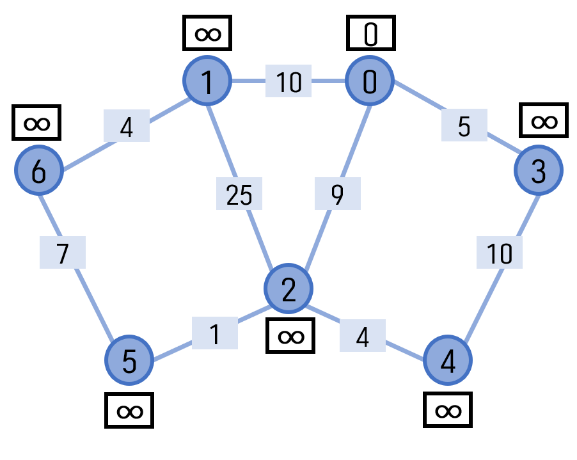
**A9-6 (4 points)**

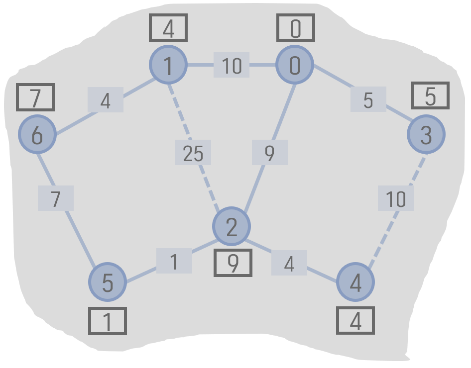
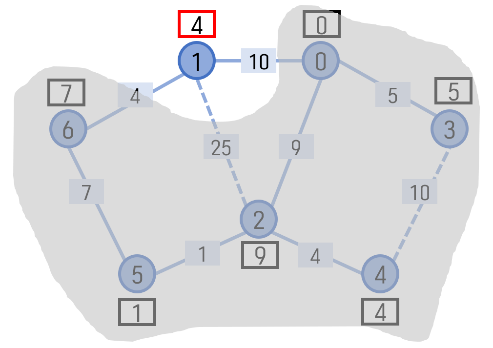


1. Illustrate a running of Dijkstra’s algorithm on this graph, starting from the vertex 0. 



1. Illustrate the execution of the Prim-Jarnik algorithm for computing the minimum spanning tree of this graph.( Assume all the edges are undirected.)





**A9-Bonus (3 points)**

Consider the following greedy strategy for finding a shortest path from vertex *start* to vertex *goal* in given connected graph.

1. Initialize *path* to *start*

2. Initialize *set* visited to {start}

3. If *start = goal*, return *path* and exit. Otherwise, continue.

4. Find the edge (*start*,v) of minimum weight such that v is adjacent to start and v is not in *visited*.

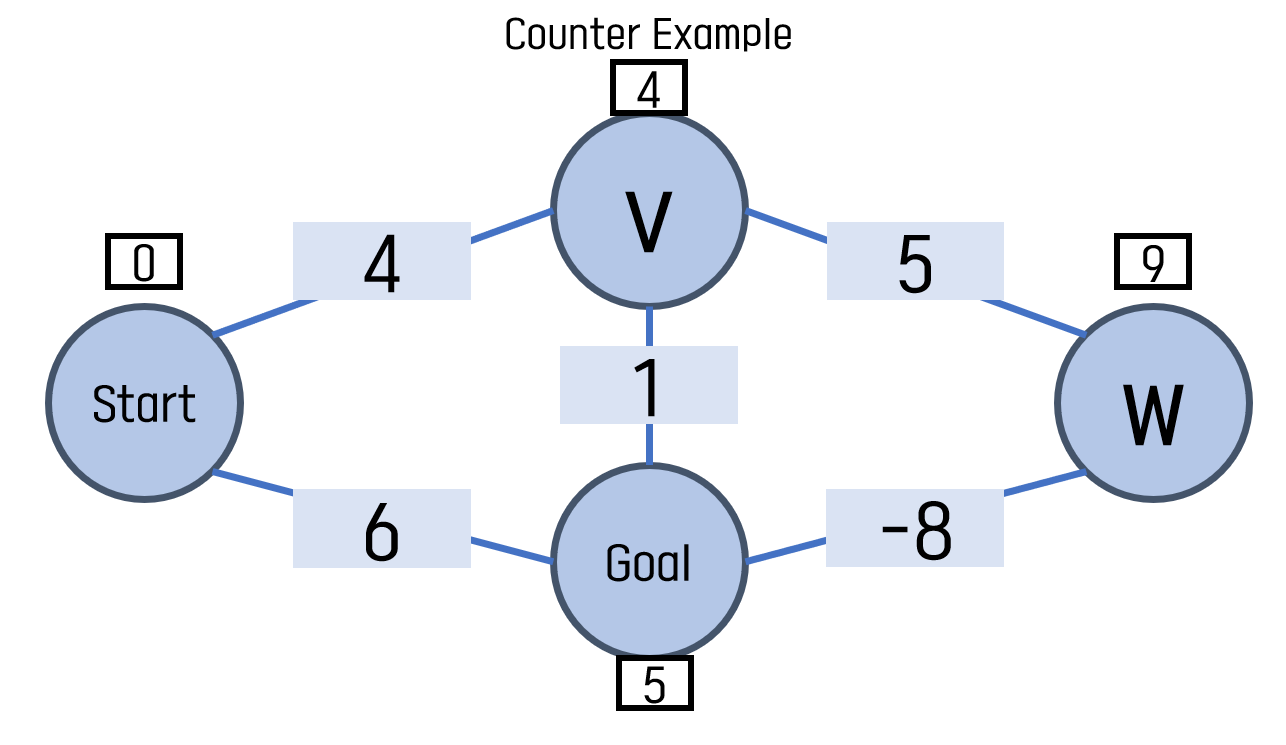
5. Add v to path

6. Add v to visited.

7.Set start equal to v and go to step 3.

Does this greedy strategy always find a shortest path from *start* to *goal*?

Either explain intuitively why it works, or give a counterexample.

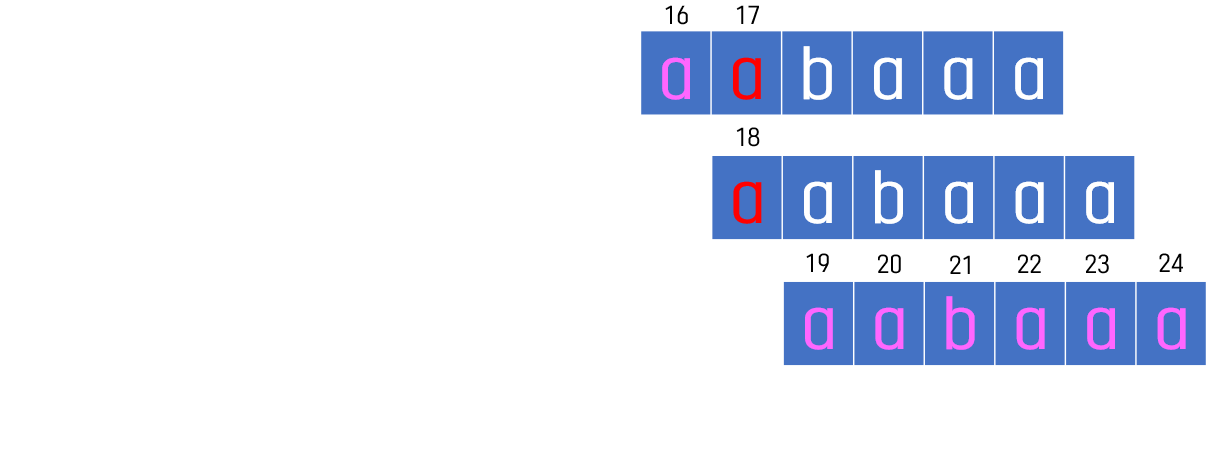
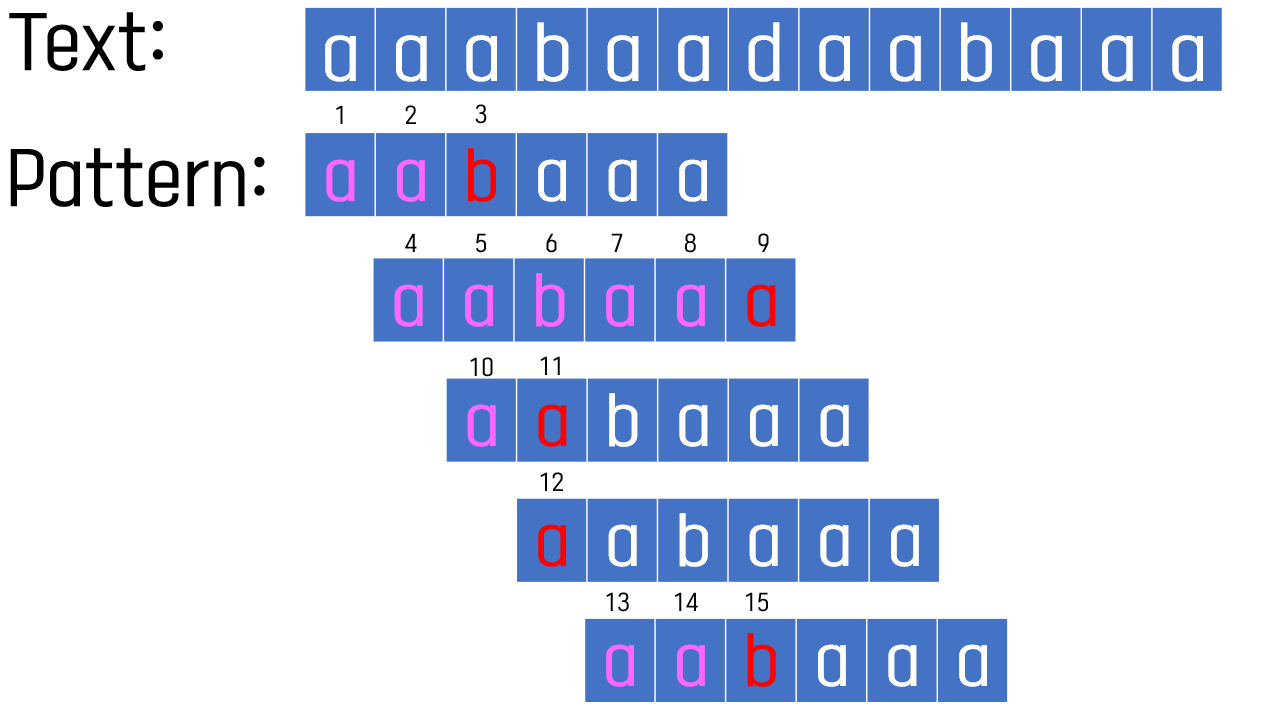


위 상황에서 알고리즘 대로 최단경로를 찾는다면 start – v – goal이 되겠지만 실제로는 음의 값을 가지고 있는 edge를 거치는 start – v – w – goal이 가장 최단 경로가 된다. 따라서 위 알고리즘은 음의 경로가 있다면 최단경로를 잘 찾지 못하므로 항상 잘 작동한다고는 할 수 없을 것이다.

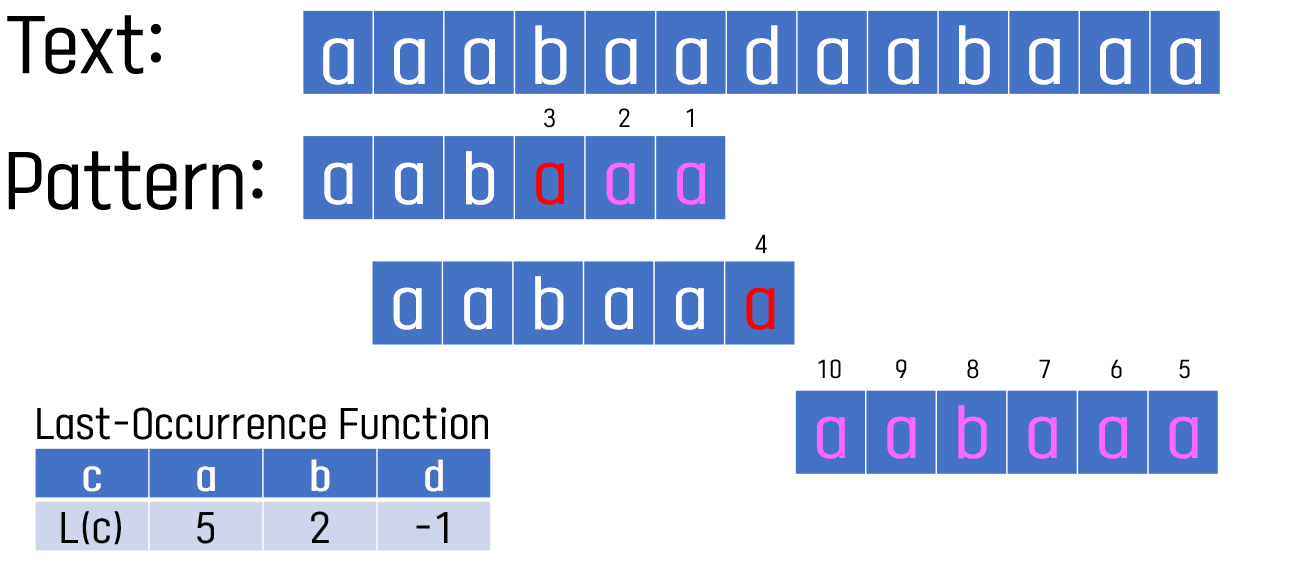
**A10 - Text Processing (5 points)**

**A10-1 (1 point)**

Illustrate the comparisons done by brute-force pattern matching for the text aaabaadaabaaa and pattern aabaaa.



**A10-2 (2 points)**

Repeat A10-1 with Boyer-Moore algorithm. Present **last** function table as well. ****

**A10-3 (2 points)**

Repeat A10-1 with Knuth-Morris-Pratt algorithm. Present **failure** function table as well.

