Ch.6.4-6.5 Trees 習題

Q1 (1/2)

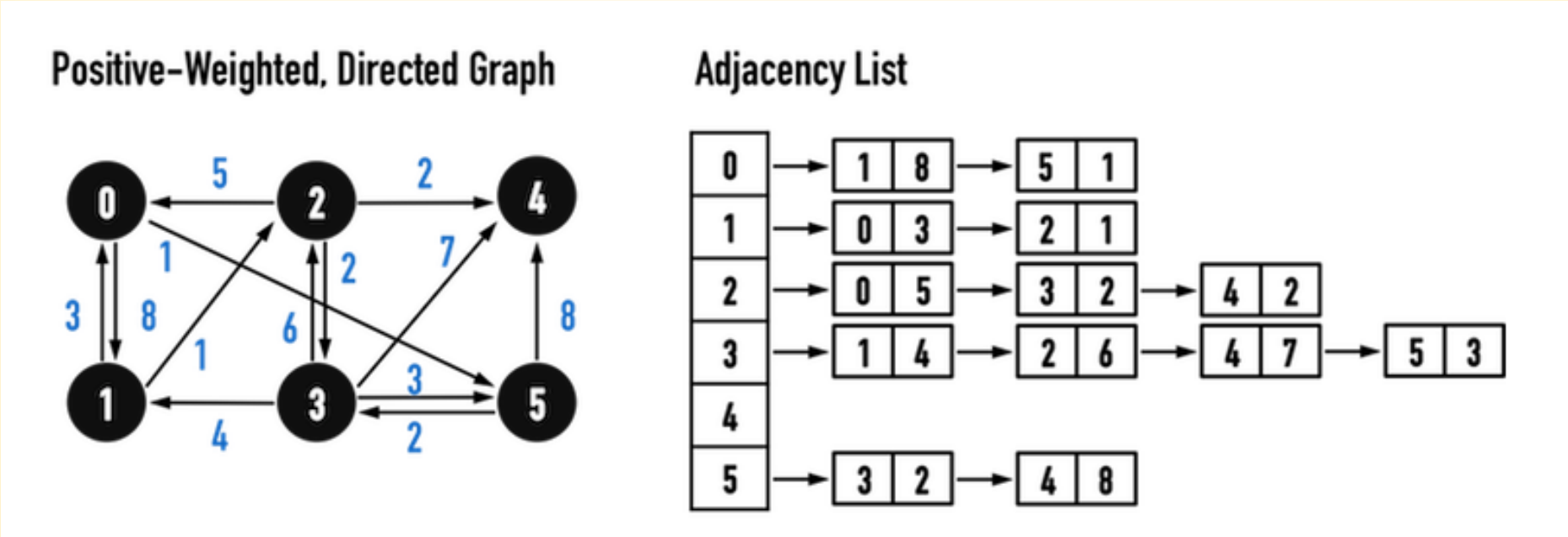
- Many shortest path algorithms are based on two important concepts, INITIALIZE-SINGLE-SOURCE and RELAX. Please explain how RELAX works following the way INITIALIZE-SINGLE SOURCE is explained in the following.

```
INITIALIZE-SINGLE-SOURCE(G,s)
        for each vertex v belongs to G.v
                dist[v] = infinity
                predecessor[v] = -1
        dist[s] = 0
```

```
RELAX(u,v,weight)    //length(u,v) means edge length of (u,v)
        If dist[v] > dist[u]+weight (u,v)
                dist[v] = dist[u]+weight (u,v)
                predecessor[v]= u
```

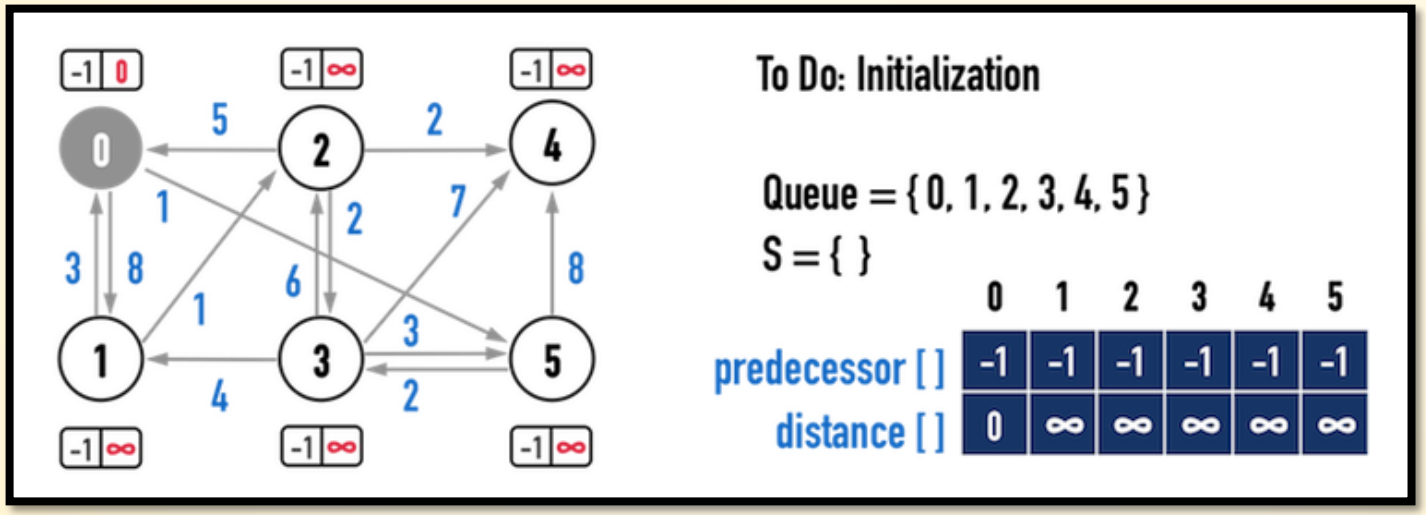
Q1 (2/2)

For a given graph G, please compute the shortest path (Dijkstra's Algorithm) to all vertices in the input graph G starting from the single source vertex s{0}.



Q1_Ans

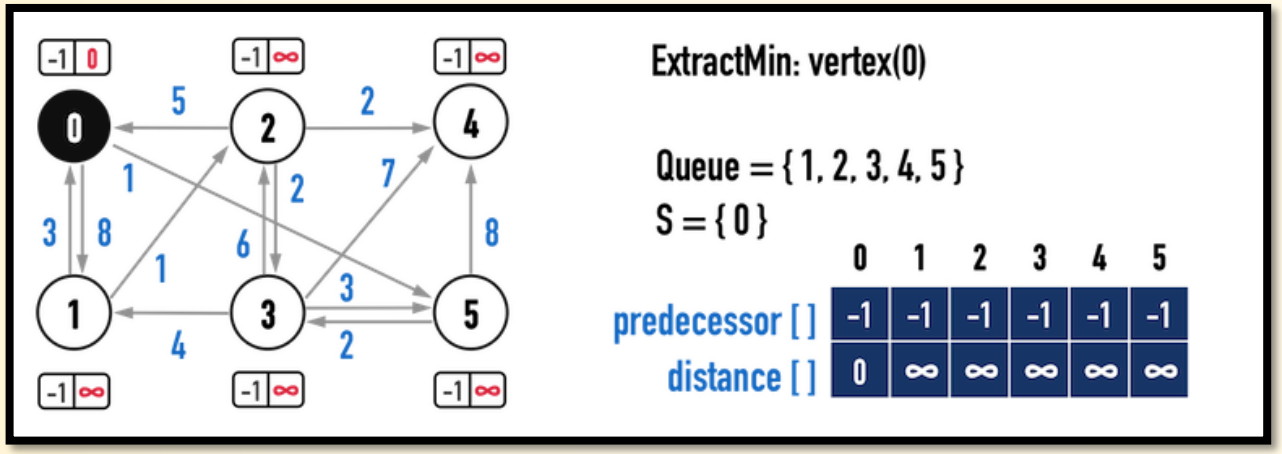
(Single Source Shortest Paths): Using the "INITIALIZE-SINGLE-SOURCE" and "RELAX" to do Dijkstra's Algorithm.

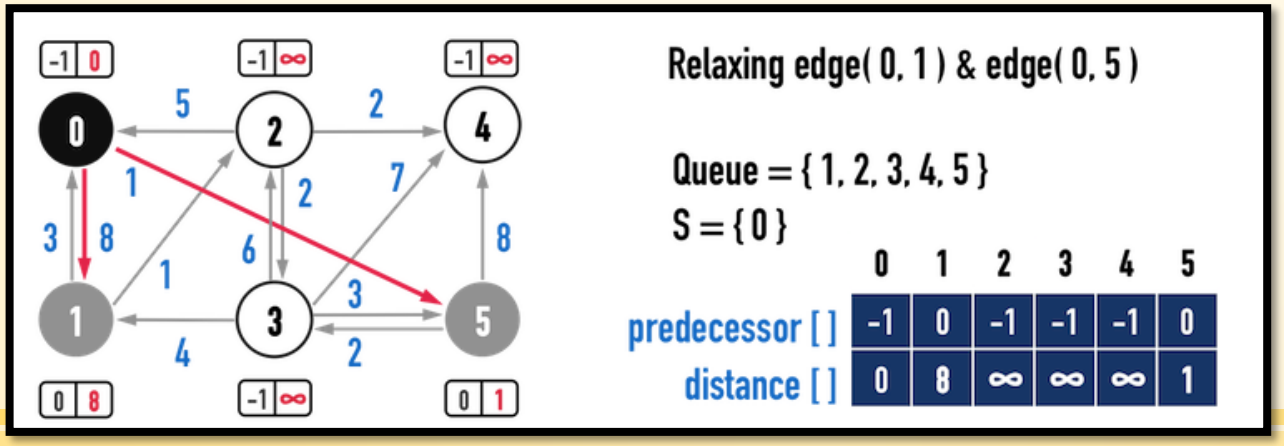


Ref: http://alrightchiu.github.io/SecondRound/single-source-shortest-pathdijkstras-algorithm.html

Q1_Ans

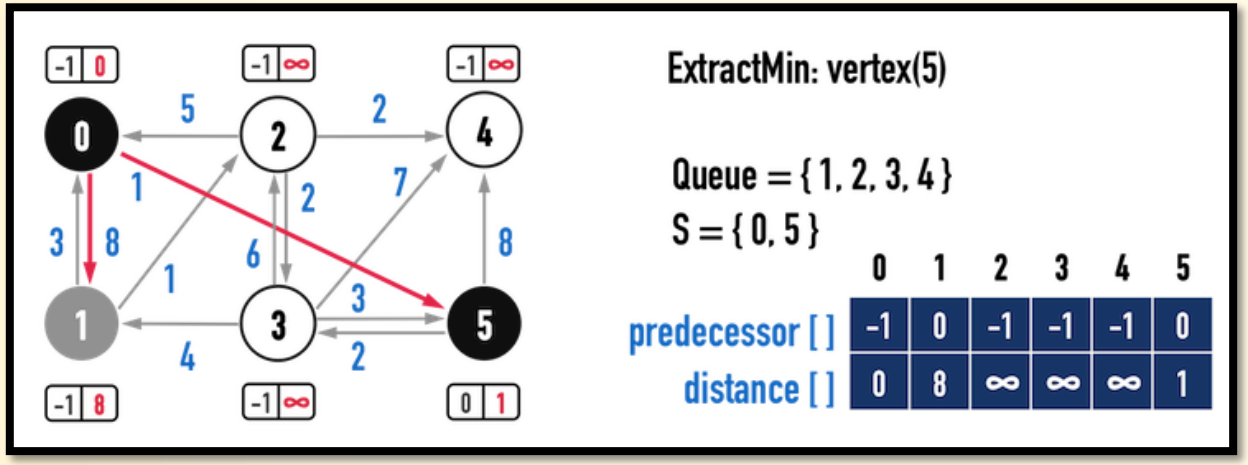
Loop 1:

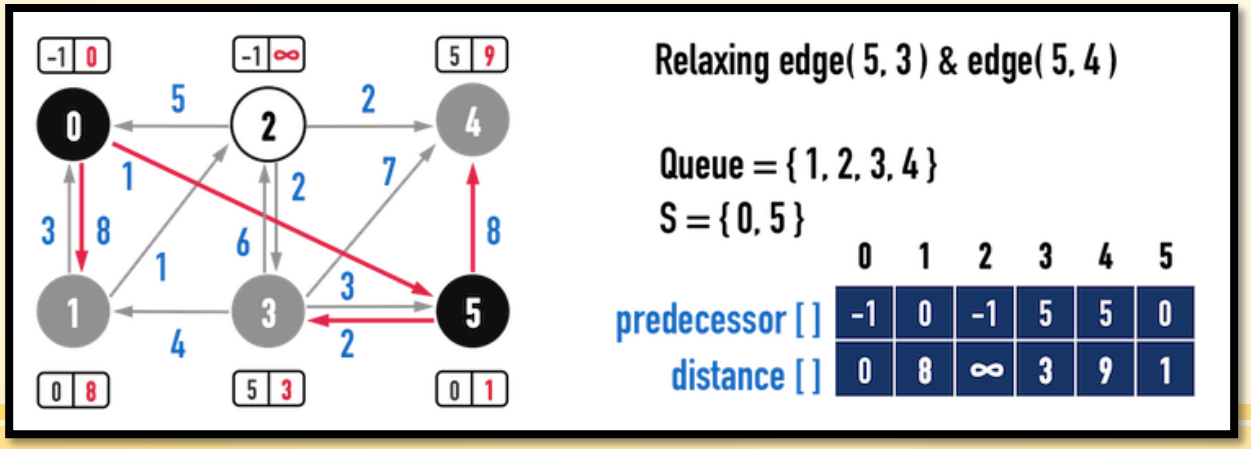

-1 0
-1 ∞
-1 ∞
0
2
4
1
3
5
-1 ∞
-1 ∞
-1 ∞
5
2
1
2
7
3
8
6
8
1
3
4
2
ExtractMin: vertex(0)
Queue = { 1, 2, 3, 4, 5 }
S = { 0 }
0 1 2 3 4 5
predecessor [ ] -1 -1 -1 -1 -1 -1
distance [ ] 0 ∞ ∞ ∞ ∞ ∞


-1 0
-1 ∞
-1 ∞
0
2
4
1
3
5
0 8
-1 ∞
0 1
5
2
1
2
7
3
8
6
8
1
3
4
2
Relaxing edge(0, 1) & edge(0, 5)
Queue = { 1, 2, 3, 4, 5 }
S = { 0 }
0 1 2 3 4 5
predecessor [ ] -1 0 -1 -1 -1 0
distance [ ] 0 8 ∞ ∞ ∞ 1

Q1_Ans

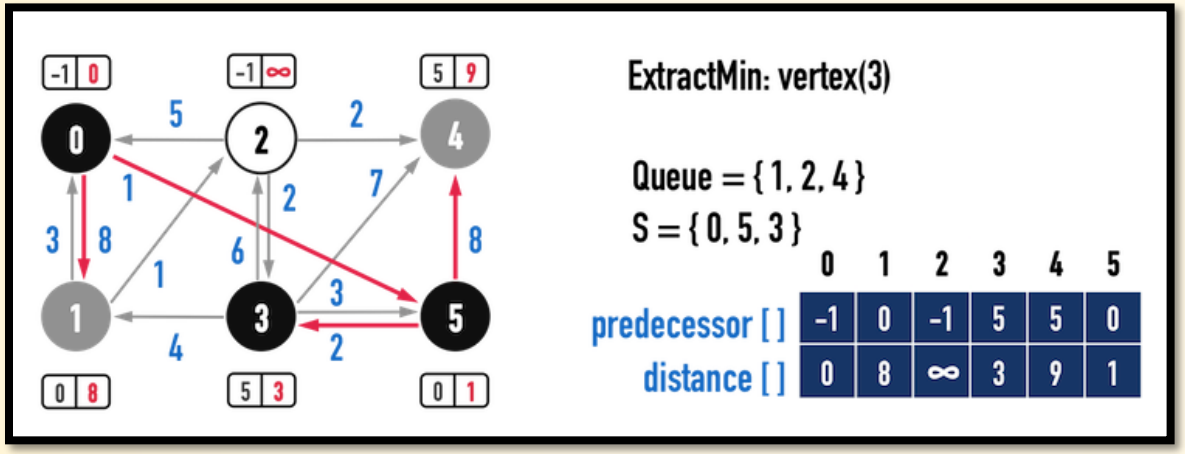
Loop 2:

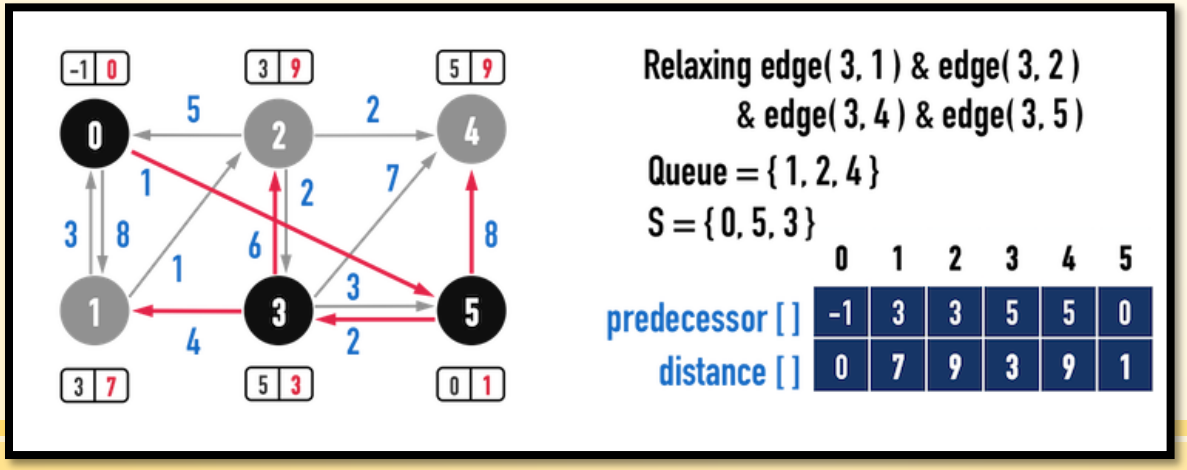

-1 0
-1 ∞
-1 ∞
0
2
4
1
3
5
-1 8
-1 ∞
0 1
5
2
2
1
7
2
3
8
6
8
1
3
4
2
ExtractMin: vertex(5)
Queue = { 1, 2, 3, 4 }
S = { 0, 5 }
0 1 2 3 4 5
predecessor [ ] -1 0 -1 -1 -1 0
distance [ ] 0 8 ∞ ∞ ∞ 1


-1 0
-1 ∞
5 9
0
2
4
1
3
5
0 8
5 3
0 1
5
2
2
1
7
2
3
8
6
8
1
3
4
2
Relaxing edge(5, 3) & edge(5, 4)
Queue = { 1, 2, 3, 4 }
S = { 0, 5 }
0 1 2 3 4 5
predecessor [ ] -1 0 -1 5 5 0
distance [ ] 0 8 ∞ 3 9 1

Q1_Ans

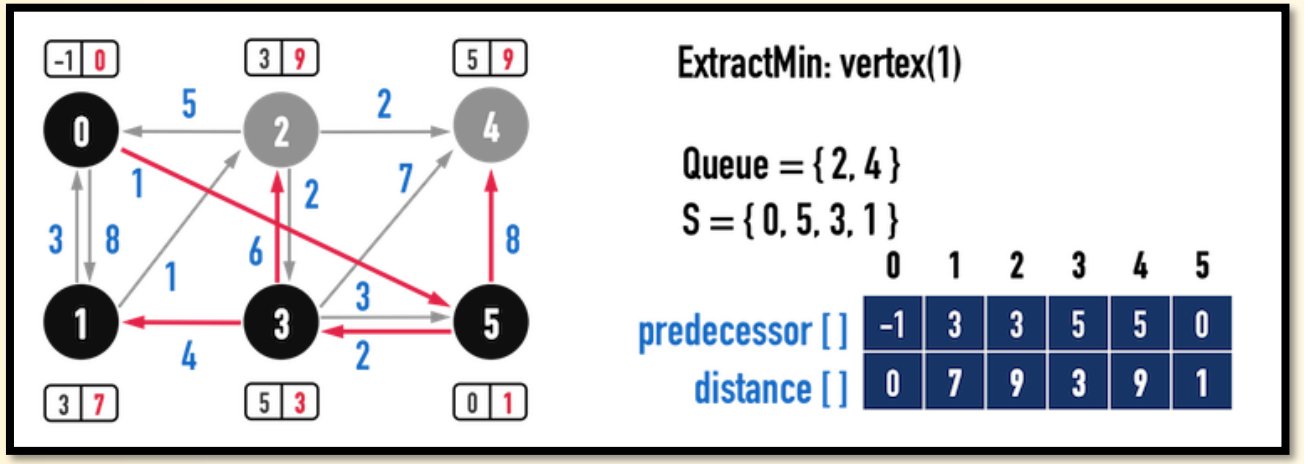
Loop 3:

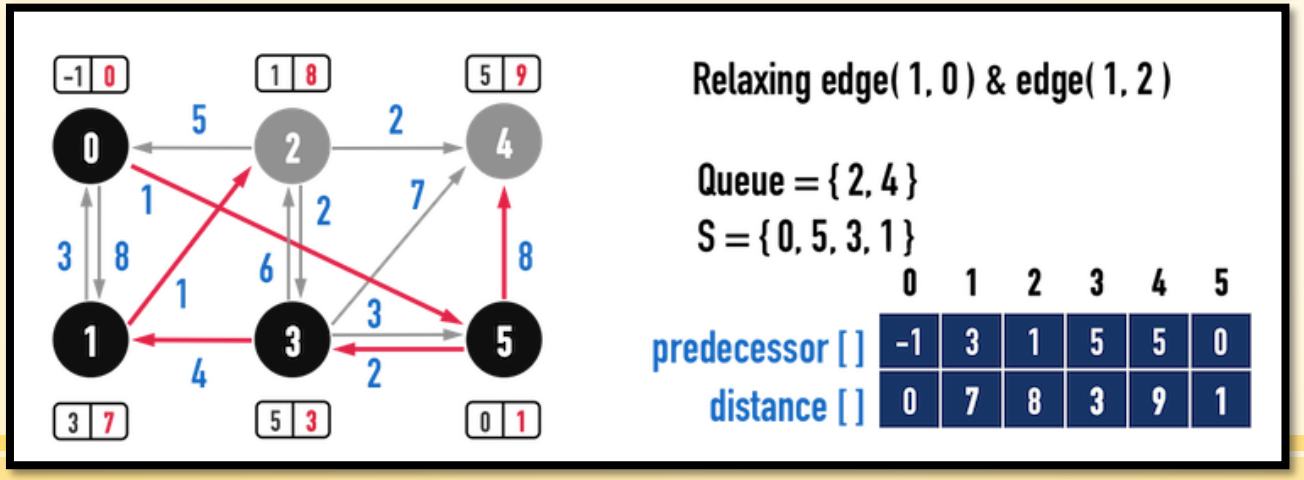

-1 0
-1 ∞
5 9
0
2
4
1
3
5
0 8
5 3
0 1
ExtractMin: vertex(3)
Queue = { 1, 2, 4 }
S = { 0, 5, 3 }
0 1 2 3 4 5
predecessor [ ] -1 0 -1 5 5 0
distance [ ] 0 8 ∞ 3 9 1


-1 0
3 9
5 9
0
2
4
1
3
5
3 7
5 3
0 1
Relaxing edge(3, 1) & edge(3, 2)
& edge(3, 4) & edge(3, 5)
Queue = { 1, 2, 4 }
S = { 0, 5, 3 }
0 1 2 3 4 5
predecessor [ ] -1 3 3 5 5 0
distance [ ] 0 7 9 3 9 1

Q1_Ans

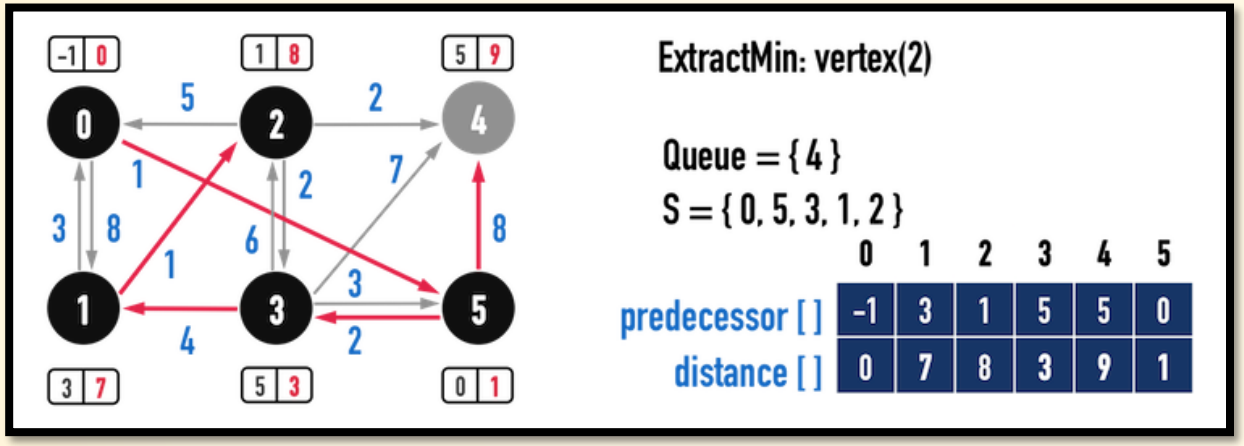
Loop 4:

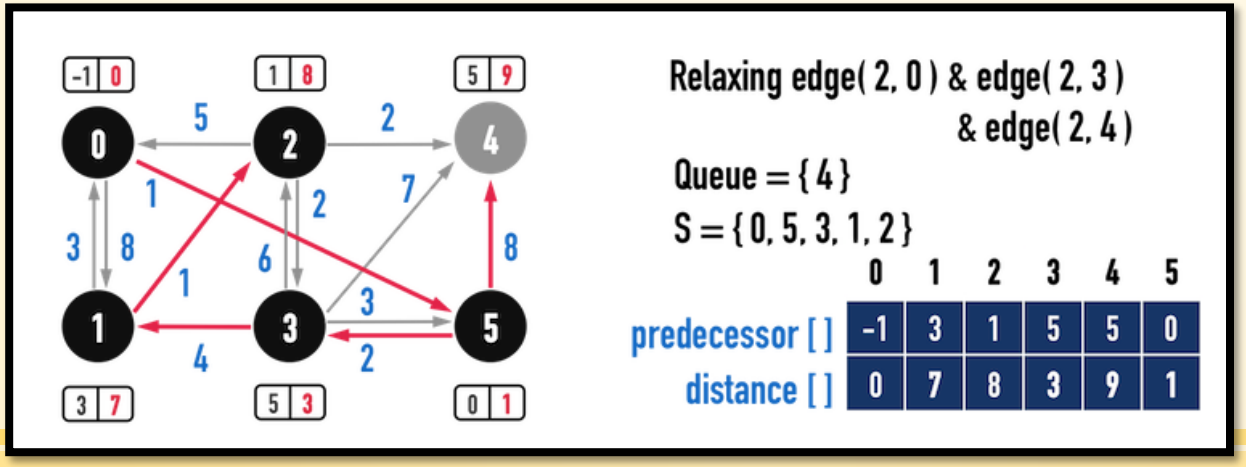

-1 0
3 9
5 9
0
2
4
1
3
5
3 7
5 3
0 1
5
2
7
8
3
8
1
6
1
3
2
4
ExtractMin: vertex(1)
Queue = { 2, 4 }
S = { 0, 5, 3, 1 }
0 1 2 3 4 5
predecessor [ ] -1 3 3 5 5 0
distance [ ] 0 7 9 3 9 1


-1 0
1 8
5 9
0
2
4
1
3
5
3 7
5 3
0 1
5
2
7
8
3
8
1
6
1
3
2
4
Relaxing edge(1, 0) & edge(1, 2)
Queue = { 2, 4 }
S = { 0, 5, 3, 1 }
0 1 2 3 4 5
predecessor [ ] -1 3 1 5 5 0
distance [ ] 0 7 8 3 9 1

Q1_Ans

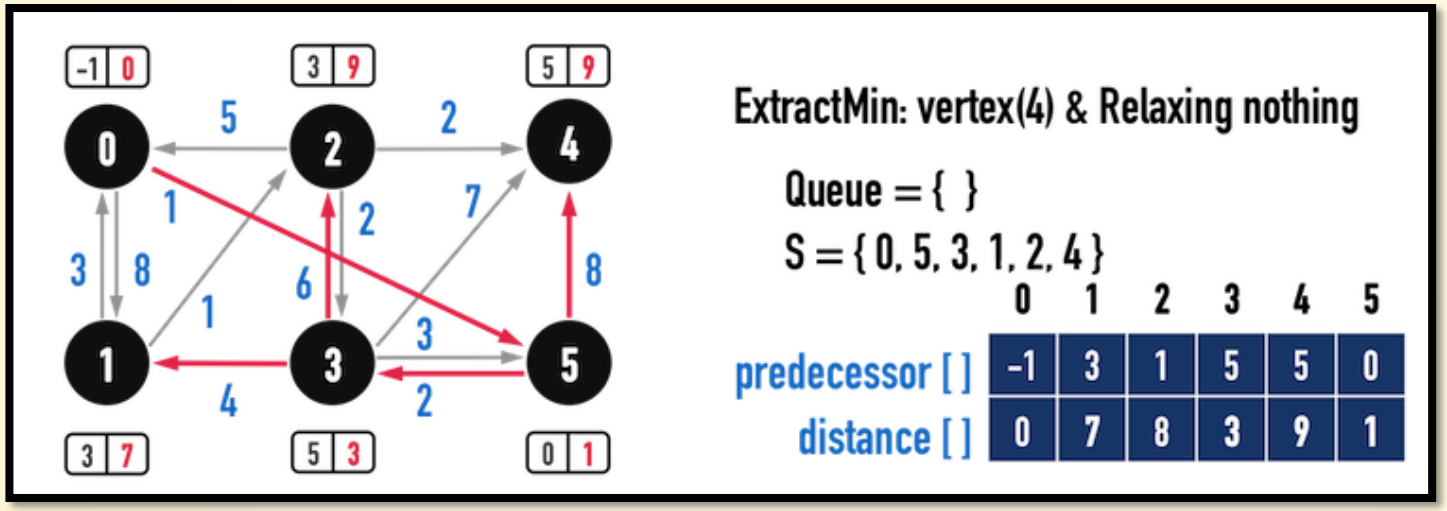
Loop 5:


-1 0
1 8
5 9
0
2
4
1
3
5
5
2
1
2
7
3
8
6
1
3
8
4
2
3 7
5 3
0 1
ExtractMin: vertex(2)
Queue = { 4 }
S = { 0, 5, 3, 1, 2 }
0 1 2 3 4 5
predecessor [ ] -1 3 1 5 5 0
distance [ ] 0 7 8 3 9 1


-1 0
1 8
5 9
0
2
4
1
3
5
5
2
1
2
7
3
8
6
1
3
8
4
2
3 7
5 3
0 1
Relaxing edge(2, 0) & edge(2, 3)
& edge(2, 4)
Queue = { 4 }
S = { 0, 5, 3, 1, 2 }
0 1 2 3 4 5
predecessor [ ] -1 3 1 5 5 0
distance [ ] 0 7 8 3 9 1

Q1_Ans

Loop 6:


-1 0
3 9
5 9
0
2
4
1
3
5
3 7
5 3
0 1
5
2
1
2
7
3
8
6
8
1
3
4
2
ExtractMin: vertex(4) & Relaxing nothing
Queue = { }
S = { 0, 5, 3, 1, 2, 4 }
0 1 2 3 4 5
predecessor [ ] -1 3 1 5 5 0
distance [ ] 0 7 8 3 9 1

Q2 (1/2)

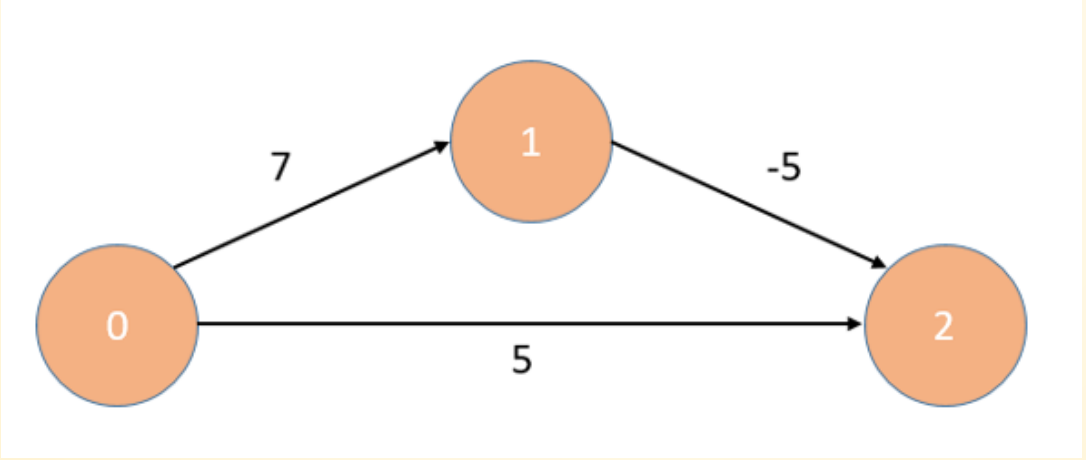
- Read the following pseudocode for Dijkstra shortest path algorithm and then answer the questions.

```
void MatrixWDigraph::ShortestPath(const int n,const int v){
    // dist[j], 0<=j<n, is set to the length of the shortest path from v to j
    // in a digraph G with n vertices and edge lengths given by length[i][j]
    for(int i=0;i<n;i++){
        s[i]=false;
        dist[i]=length[v][i];
    }
    s[v]=true;
    dist[v]=0;

    for(i=0;i<n-2;i++){  // determine n-1 paths from vertex v
        int u=Choose(n); // choose returns a value u such that
                         // dist[u]=minimum dist[w], where s[w]=false
        s[u]=true;
        for(int w=0;w<n;w++)
            if(!s[w]&&dist[u]+length[u][w]<dist[w])
                dist[w]=dist[u]+length[u][w];
    }
}
```

Q2 (2/2)

1) Explain how the pseudocode work.
2) When applying Dijkstra algorithm to the graph below starting from the vertex 0, what issue may occur?
3) Try to implement Dijkstra algorithm by yourself.



Q2_Ans (1)

(1) Explain how the pseudocode work.

Ans: (Ref: textbook for 6.4 Single Source All Destinations)

- The algorithm ShortestPath as first given by Dijkstra, makes use of these observations to determine the cost of the shortest paths from v to all other vertices in G.
- It is assumed that the n vertices of G are numbered 0 through n-1. The set S is maintained as a Boolean array with S(i) = false if vertex i is not in S and S(i) = true if it is.
- It is assumed that the graph itself is represented by its length adjacency matrix with length[i][j] being the weight of the edge <i, j> in G.

Q2_Ans (1)

(1) (continue)

(i) If the next shortest path is to vertex u, then the path begins at v, ends at u and goes through only those vertices which are in S. To prove this we must show that all of the intermediate vertices on the shortest path to u must be in S.

```
for (i=0; i<n-2; i++) //determine (n-1) path from vertex v
```

(ii) The destination of the next path generated must be that vertex u which has the minimum distance, dist(u), among all vertices not in S.

```
int u = Choose(n);    s[u] = true;
```

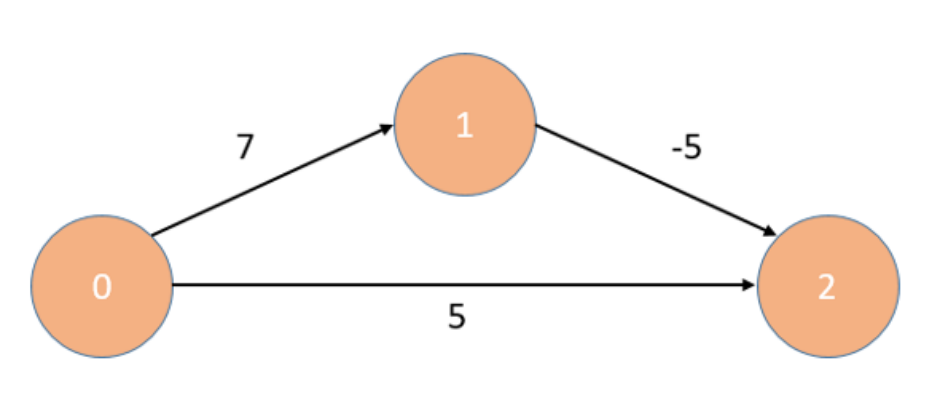
(iii) Having selected a vertex u as in (ii) and generated the shortest v to u path, vertex u becomes a member of S. At this point the length of the shortest paths starting at v, going through vertices only in S and ending at a vertex w not in S may decrease.

```
for (int w=0; w<n; w++)

 if (! s[w]&&dist[u]+length[u][w] < dist[w]) dist[w] = dist[u]+length[u][w];
```

Q2_Ans (2)

(2) When applying Dijkstra algorithm to the graph below starting from the vertex 0, what issue may occur?




```
for(int i=0;i<n;i++){
    s[i]=false;
    dist[i]=length[v][i];
}
s[v]=true;
dist[v]=0;
```

Set n = 3 and v = 0

The result:

s[0]=true

s[1]=false

s[2]=false

dist[0]=0

dist[1](=length[0][1])=7

dist[2](=length[0][2])=5

Q2_Ans (2)

```
for(i=0;i<n-2;i++){  // determine n-1 paths from vertex v
    int u=Choose(n); // choose returns a value u such that
                     // dist[u]=minimum dist[w], where s[w]=false
```

The result:

u=2

s[0]=true

s[1]=false

s[2]=true

```
    for(int w=0;w<n;w++)
        if(!s[w]&&dist[u]+length[u][w]<dist[w])
            dist[w]=dist[u]+length[u][w];
}
```

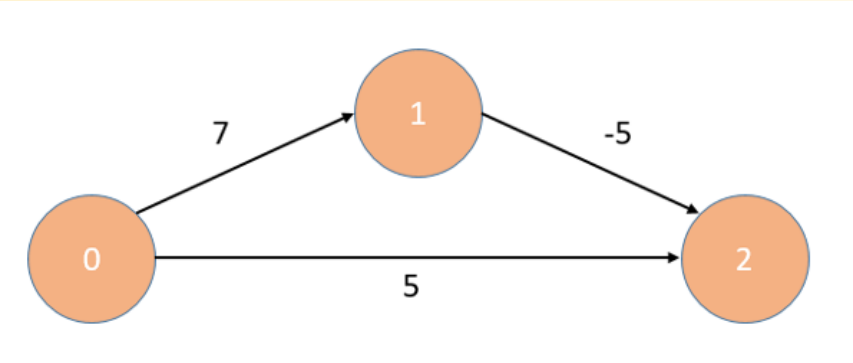
We known:

!s[w] when w=0 or 2 => if(0 && condition) = 0

The result:

w=1 => dist[2]+length[2][1] (not exist)

=> dist[2]+Large number>dist[1]



Ans:

dist[0] = 0

dist[1] = 7

dist[2] = 5 (not 2)

Q2_Ans (3)

Please refer to:

https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/

Q3

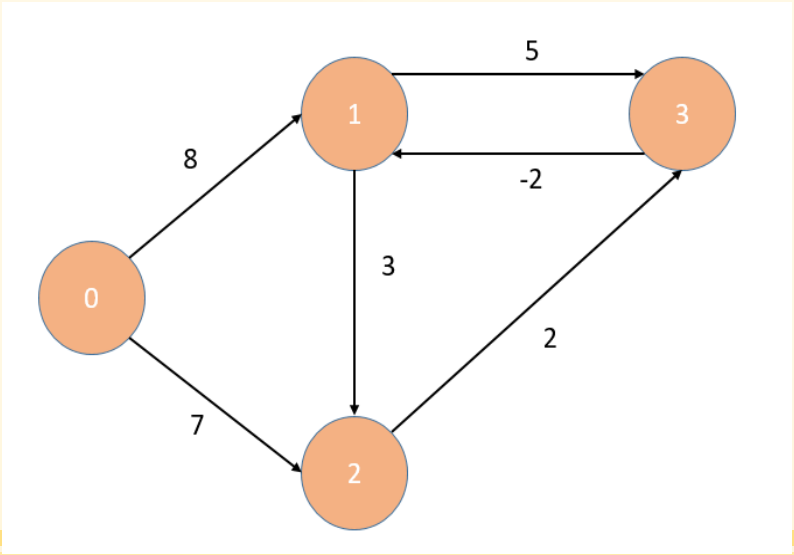
- In addition to the Dijkstra algorithm, BellmanFord algorithm can also solve the single-source shortest paths problem. A sample BellmanFord pseudocode is listed below for your reference.

```
void MatrixWDigraph::BellmanFord(const int n, const int v){
    // Single source all destination shortest paths with negative edge lengths.
    for(int i=0;i<n;i++){
        dist[i]=length[v][i];
    }
    for(int k=2;k<=n-1;k++){
        for(each u such that u!=v and u has at least one incoming edge){
            for(each<i,u> in the graph){
                if(dist[u]>dist[i]+length[i][u])
                    dist[u]=dist[i]+length[i][u]
            }
        }
    }
}
```

Q3

(1) Please explain the difference between the BellmanFord and the Dijkstra algorithm. Your answer should at least contain the difference of basic idea, and the time complexity.

(2)Please explain how can you apply the BellmanFord algorithm to find the shortest paths of the graph below. You may try to practice how to implement the algorithm.



Q3_Ans (1)

(1) Please explain the difference between the BellmanFord and the Dijkstra algorithm. Your answer should at least contain the difference of basic idea, and the time complexity.

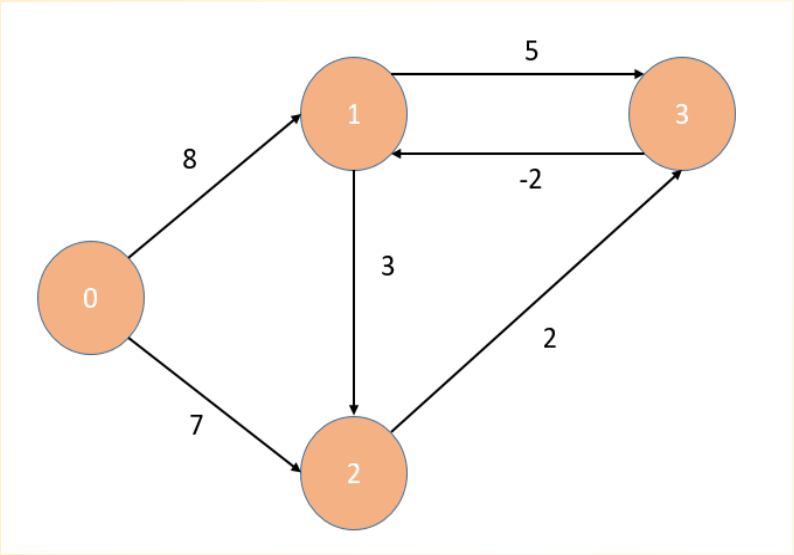
- Ans:(Ref:
  - Dijkstra alg. : Similar to Prim's algorithm for minimum spanning tree. Starting from the source vertex, if the extended vertex is found to be the shortest path, it is added to the set (S), and continues until all vertices are added to the set. It searches for the next vertex through O(n) and updates the dist length through O(n) time, so it takes $O(n^2)$ to complete the shortest distance from the source vertex to all vertices.

Q3_Ans (1)

- BellmanFord Alg.: The shortest path from the starting v to all the ending u can be completed by dynamic programming. Suppose that the shortest path from v to u is the shortest path from v to j followed by <j, u> in a Graph. We say i on the <i, u> are all candidates for j, and we are interested in the shortest path, so $dist^k[u]$ = min {$dist^{k-1}[u]$, mini{$dist^{k-1}$ [i]+length[i][u]}}. The smallest i is the correct value of j. BellmanFord can find the shortest path length from the starting v to each of the other vertices in Graph (G) by recursively $dist^k$ k= 2, …,n-1. It takes $O(n^2)$ for one iteration, and it takes $O(n^3)$ after all iterations.

Q3_Ans (2)

(2) Please explain how can you apply the BellmanFord algorithm to find the shortest paths of the graph below. You may try to practice how to implement the algorithm. (Ref: textbook for 6.4 Digraph with Negative Costs)



k	$dist^k[4]$			
	0	1	2	3
1	0	8	7	∞
2	0	8	7	9
3	0	7	7	9

Q4

- Although BellmanFord cannot find the shortest path of a graph with negative-cycle, it can be modified to check whether the graph has negative-cycles. Please explain how this can be done.

Q4_Ans

the Bellman–Ford algorithm simply relaxes all the edges, and does this |V|-1 times (if there are no negative cycles).

```
// Step 1: Initialize
// Step 2: RELAX edges repeatedly |V|-1
// Step 3: check for negative-weight cycles
for each edge (u, v) with weight w in edges do
   if distance[u] + w < distance[v] then
      error "Graph contains a negative-weight cycle“
```

// If the shortest path can be found after updating |V|-1 times, it means that between any edge (u, v), the weights add up to be negative, resulting in a continuous decrease in the path. At this point, we can determine that Graph (G) has a negative cycle.

Ref: https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm

Q5

- The Floyd-Warshall algorithm is used to solve the All-Pairs Shortest Path problem. Please answer the following questions :
  (1) What does All-Pairs Shortest Path mean?
  (2) Explain the concept of the Floyd-Warshall algorithm.

Q5_Ans (1)

(1) What does All-Pairs Shortest Path mean? (Ref: textbook for 6.4 All Pairs Shortest Paths)

The all-pairs shortest path calls for finding the shortest paths between all pairs of vertices u, v ($u \neq v$).

Q5_Ans (2)

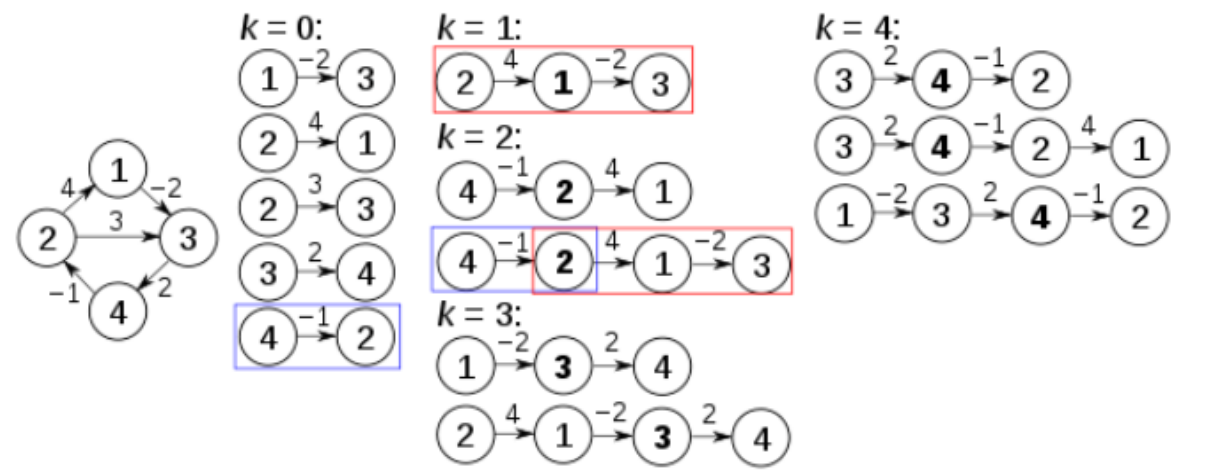
(2) Explain the concept of the Floyd-Warshall algorithm.

Regarding each vertex in Graph as a starting vertex, then this problem can be solved as a problem of n independent “single source all destinations”.

Pseudocode: ($O(n^3)$)

```
let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
for each edge (u, v) do
    dist[u][v] ← w(u, v)  // The weight of the edge (u, v)
for each vertex v do
    dist[v][v] ← 0
for k from 1 to |V|
    for i from 1 to |V|
        for j from 1 to |V|
            if dist[i][j] > dist[i][k] + dist[k][j]
                dist[i][j] ← dist[i][k] + dist[k][j]
            end if
```

Q5_Ans (2): Example



$k=0$	j			
i	**1**	**2**	**3**	**4**
1	0	∞	−2	∞
2	4	0	3	∞
3	∞	∞	0	2
4	∞	−1	∞	0

$k=1$	j			
i	**1**	**2**	**3**	**4**
1	0	∞	−2	∞
2	4	0	**2**	∞
3	∞	∞	0	2
4	∞	−1	∞	0

$k=2$	j			
i	**1**	**2**	**3**	**4**
1	0	∞	−2	∞
2	4	0	2	∞
3	∞	∞	0	2
4	**3**	−1	**1**	0

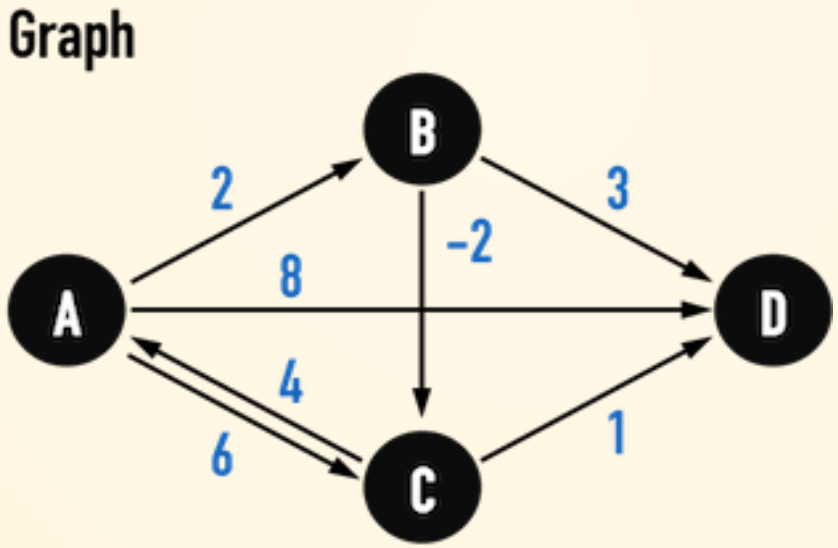
$k=3$	j			
i	**1**	**2**	**3**	**4**
1	0	∞	−2	**0**
2	4	0	2	**4**
3	∞	∞	0	2
4	3	−1	1	0

$k=4$	j			
i	**1**	**2**	**3**	**4**
1	0	**−1**	−2	0
2	4	0	2	4
3	**5**	**1**	0	2
4	3	−1	1	0

Ref: https://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm

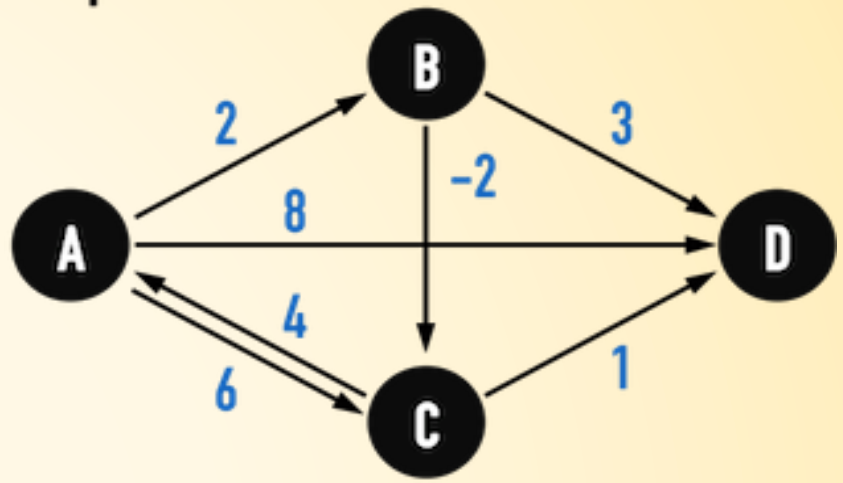
Q6

- Please use Floyd-Warshall algorithm to find the shortest path of all vertices in following graph.



Q6_Ans

Graph



A^{-1}	A	B	C	D
A	0	2	6	8
B	∞	0	-2	3
C	4	∞	0	1
D	∞	∞	∞	0

1

A^{A}	A	B	C	D
A	0	2	6	8
B	∞	0	-2	3
C	4	6	0	1
D	∞	∞	∞	0

2

A^{B}	A	B	C	D
A	0	2	0	5
B	∞	0	-2	3
C	4	6	0	1
D	∞	∞	∞	0

3

A^{C}	A	B	C	D
A	0	2	0	1
B	2	0	-2	-1
C	4	6	0	1
D	∞	∞	∞	0

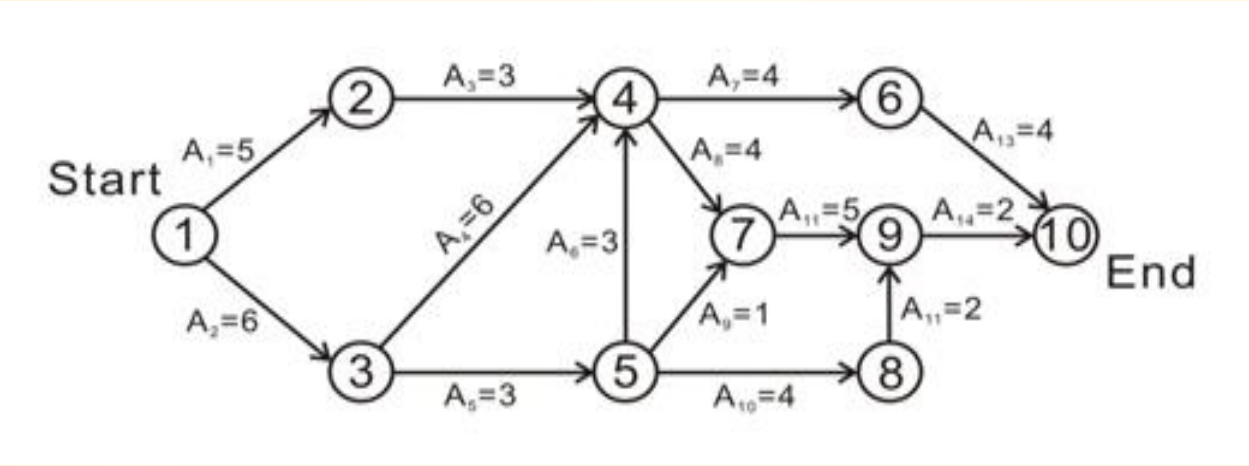
4

A^{D}	A	B	C	D
A	0	2	0	1
B	2	0	-2	-1
C	4	6	0	1
D	∞	∞	∞	0

Q7

(1) Explain the concept of the AOE Network (Activity on Edge) and define what is a critical path.

(2) What is the critical path of the AOE network below? Explain your method.



Q7_Ans (1)

(1) Explain the concept of the AOE Network (Activity on Edge) and define what is a critical path. (Ref: textbook for 6.5 AOE network)

AOE:

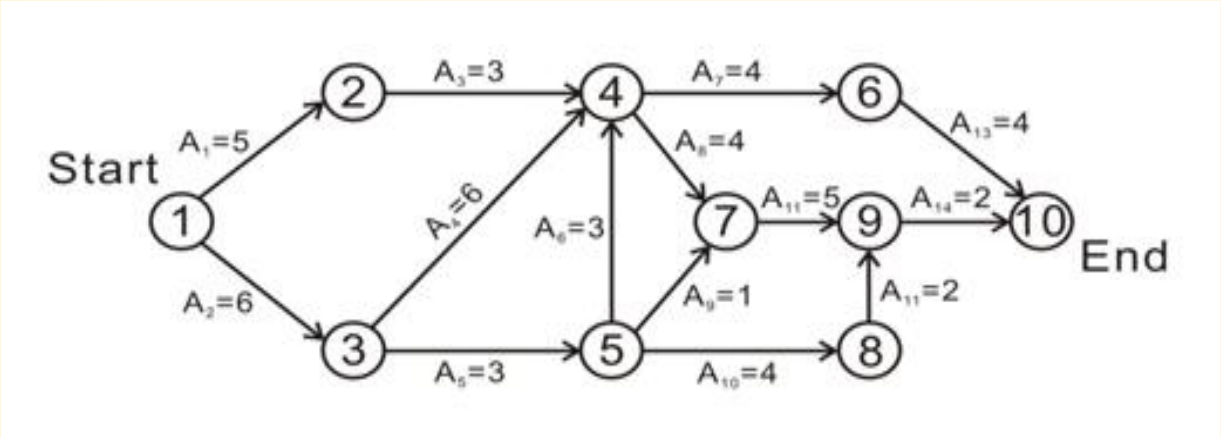
- Directed edge: tasks or activities to be performed
- Vertex: events which signal the completion of certain activities
- Number: associated with each edge (activity) is the time required to perform the activity

Critical path:

- A path that has the longest length and the total costs of all irreducible tasks to complete the project.

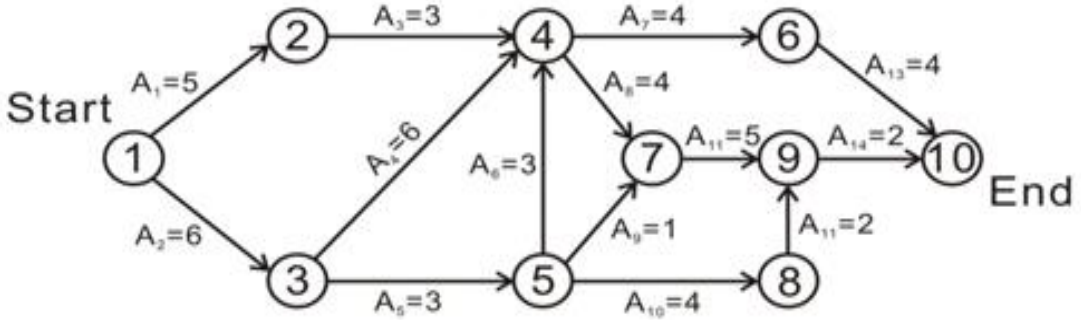
Q7_Ans (2)

(2) What is the critical path of the AOE network below? Explain your method.



Hint:
Step 1: Find ee (earliest time of edge)
Step 2: Find le (latest time of edge)
***If le[k] == ee [k] (Vertex of event k), iff e(i) == l(i) (Edge of task A_i). It represents a certain task before a certain event that the time must be executed. This path of longest length (we cannot ignore these execution times) is a critical path.**

Q7_Ans (2)



[#]: Vertex of event

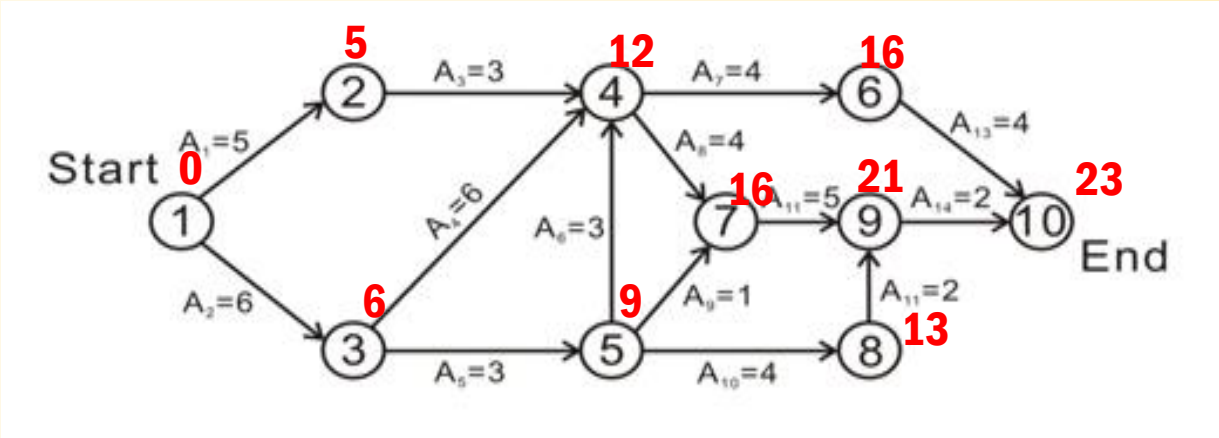
- Step 1_ee: (Ref: textbook for 6.5 AOE)

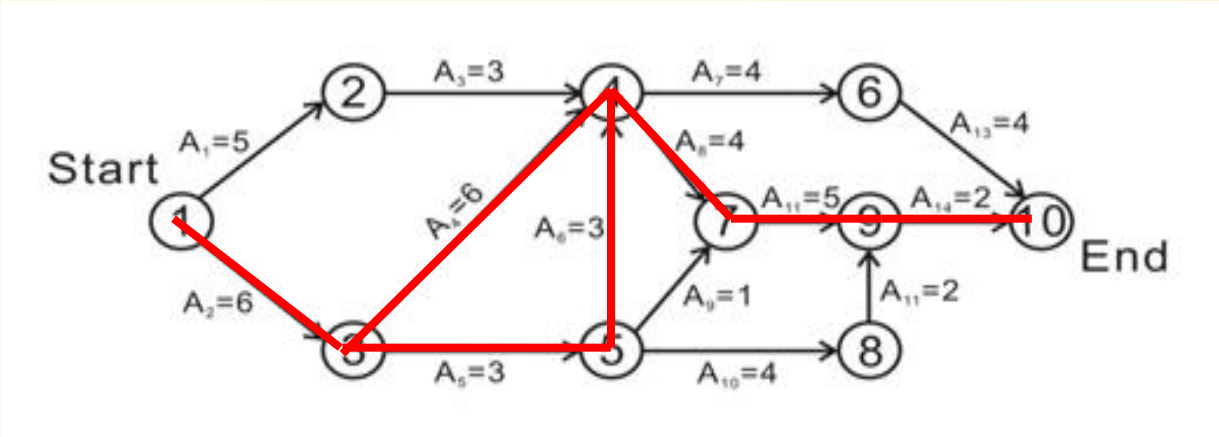
ee	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	Stack
Start	0	0	0	0	0	0	0	0	0	0	[1]
Output 1	0	5	6	0	0	0	0	0	0	0	[3, 2]
Output 3	0	5	6	12	9	0	0	0	0	0	[5, 2]
Output 5	0	5	6	12	9	0	10	13	0	0	[8, 2]
Output 8	0	5	6	12	9	0	10	13	15	0	[2]
Output 2	0	5	6	12	9	0	10	13	15	0	[4]
Output 4	0	5	6	12	9	16	16	13	15	0	[7, 6]
Output 7	0	5	6	12	9	16	16	13	21	0	[9, 6]
Output 9	0	5	6	12	9	16	16	13	21	23	[6]
Output 6	0	5	6	12	9	16	16	13	21	23	[10]
Output 10											

Q7_Ans (2)

- Step 2_le:

le[10] = ee[10] = 23
le[6] = min{ le[10] - 4 }=19
le[9] = min{ le[10] - 2 } = 21
le[7] = min{ le[9] - 5 } = 16
le[4] = min{ le[6] - 4, le[7] - 4} = 12
le[2] = min{ le[4] - 3} = 9
le[8] = min{ le[9] – 2 } = 19
le[5] = min{ le[7] – 1, le[8] - 4} = 9
le[3] = min{ le[4] – 6, le[5] - 3} = 6
le[1] = min{ le[3] – 6, le[2] - 5} = 0

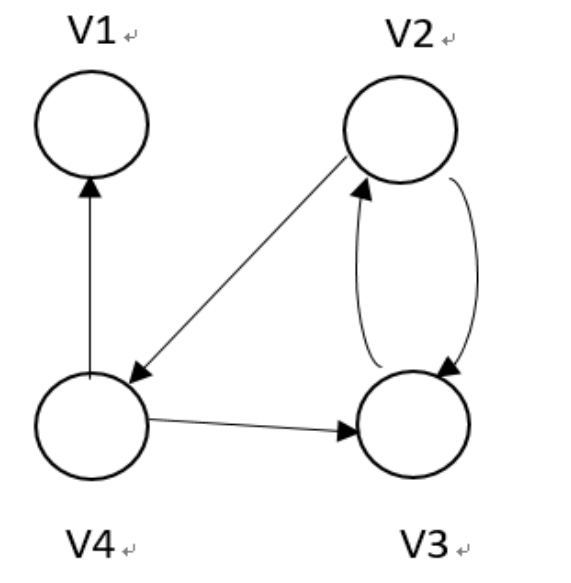




The red is critical path, which represents the execution time that must be executed without margin.

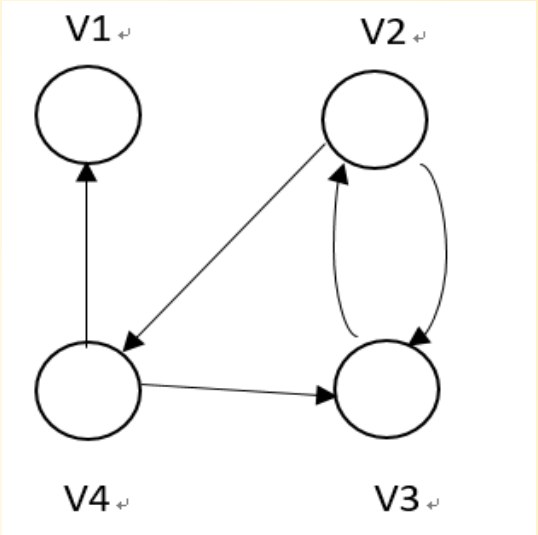
Q8

- Please write a pseudo code that can implement your method to find a transitive closure of the graph below.



Q8_Ans

- (Ref: slide for 6.4 Transitive Closure)



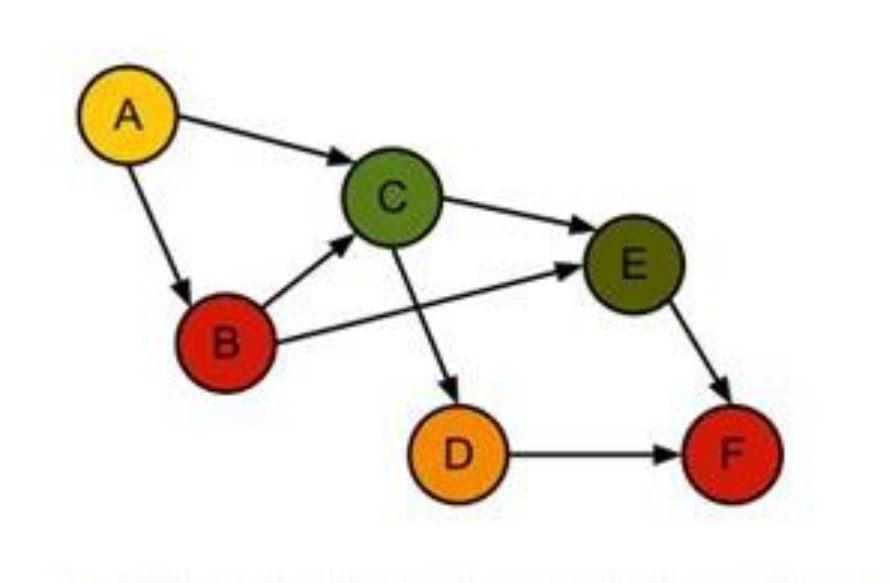
The **transitive closure matrix** A^+:

- A^+ is a matrix such that $A^+[i][j] = 1$ if there is **a path of length > 0 from *i* to *j*** in the graph; otherwise, $A^+[i][j] = 0$.

A^+	V1	V2	V3	V4
V1	0	0	0	0
V2	1	1	1	1
V3	1	1	1	1
V4	1	1	1	1

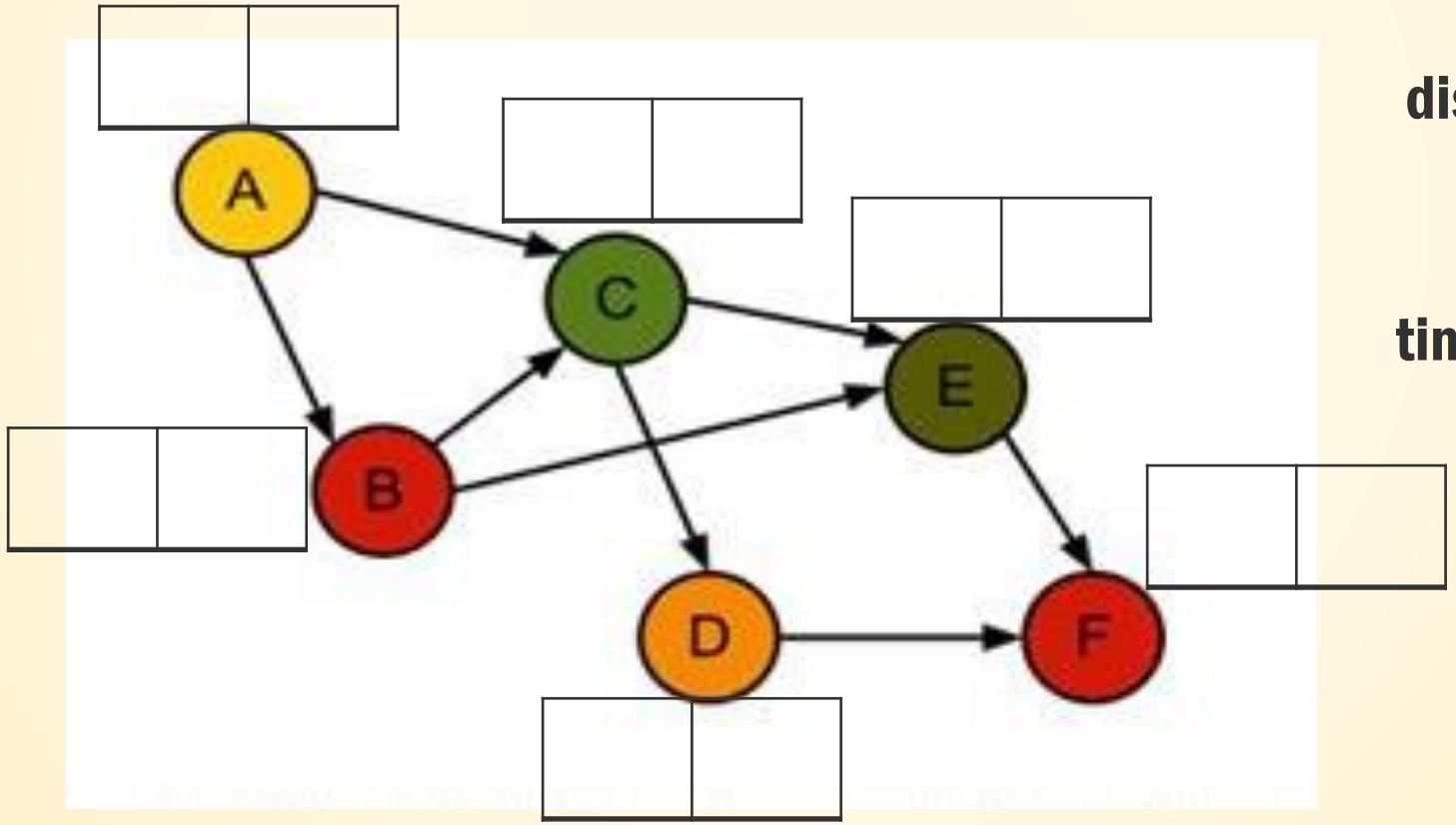
Q9

- Please show step-by-step how you apply the DFS to find the topological order of the graph vertices.



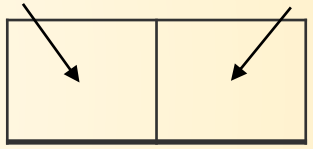
Q9_Ans (1/11)

- We use “time”, “discover/finish array” and “predecessor array” to implement DFS.



Predecessor: { }

discover finish

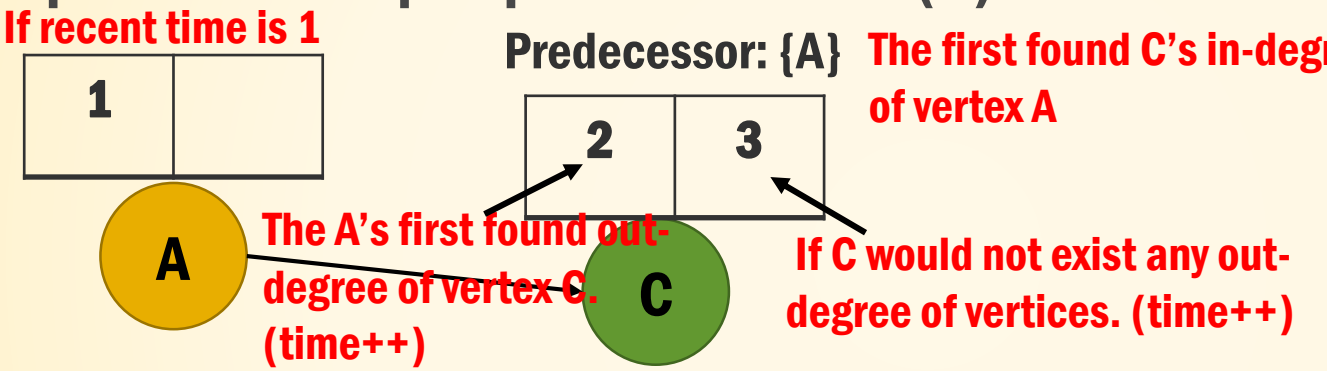


time = 0 (not vertex found)

Ref: http://alrightchiu.github.io/SecondRound/graph-depth-first-searchdfsshen-du-you-xian-sou-xun.html

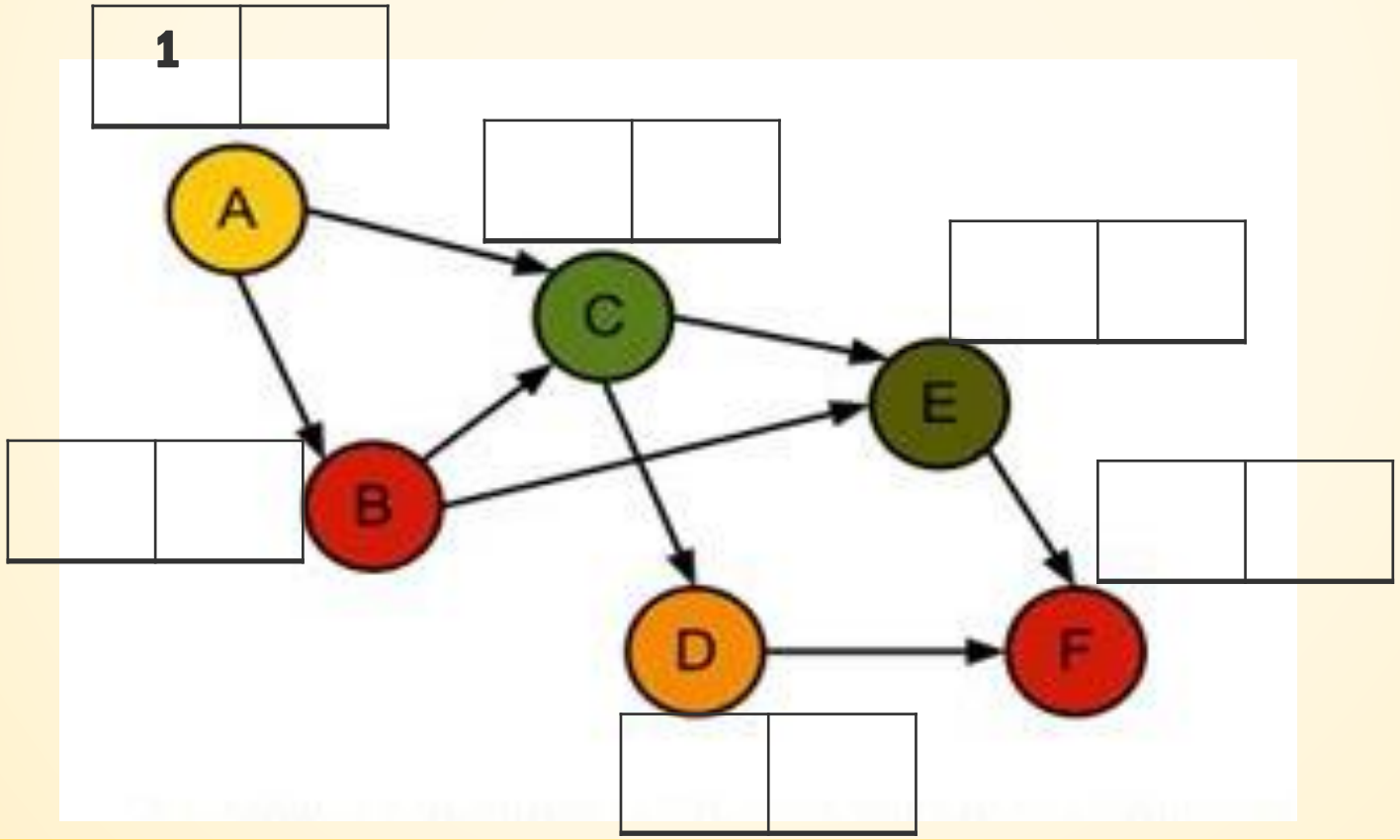
Q9_Ans (2/11)

- Explain from the perspective of Vertex (C):



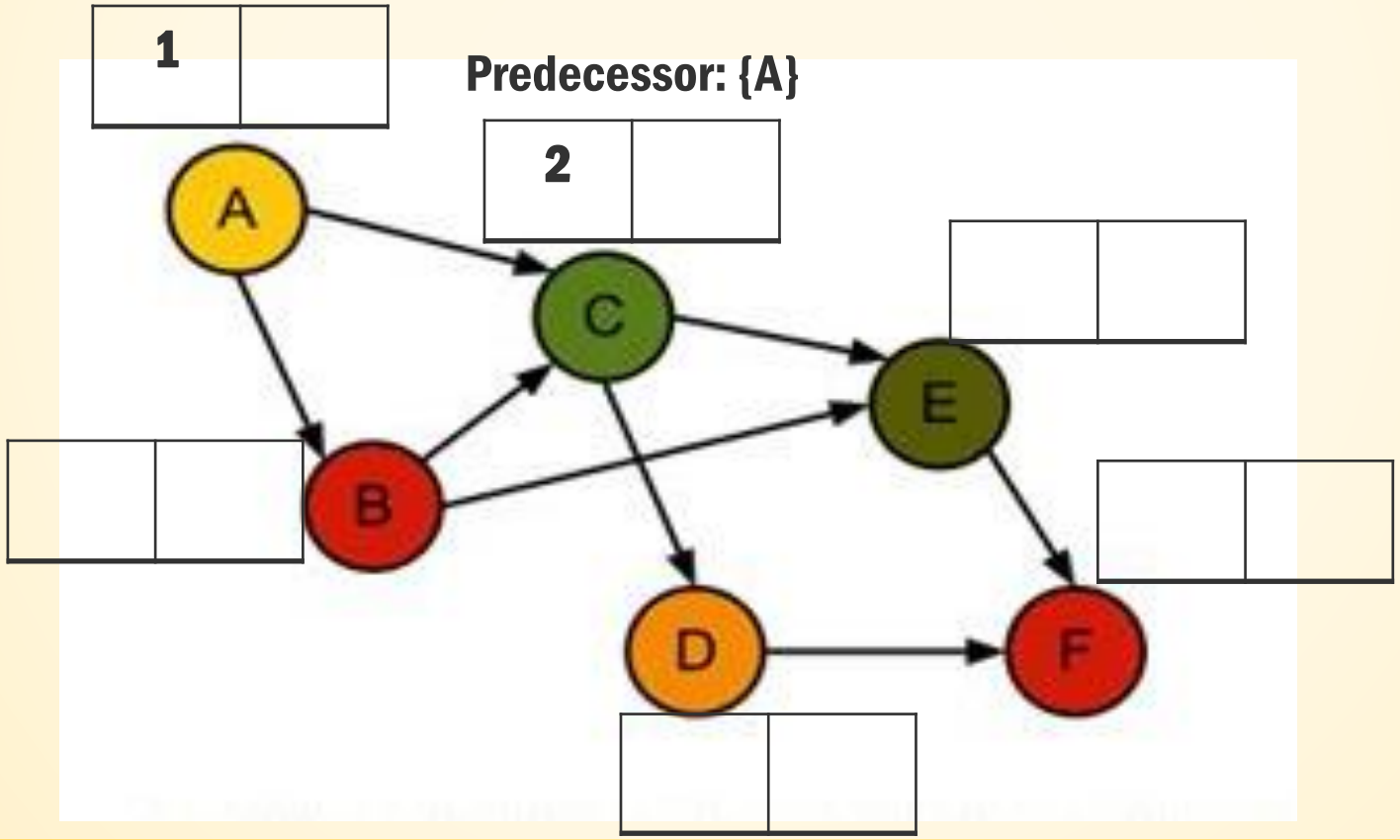
1. A是第一個透過搜尋到C的vertex，在關係上，我們把它視作C的Predecessor。
2. 在我們找到新的vertex時，會將time++，並存到新vertex的discover array中，代表新vertex在該時刻(更新的time)被找到。
3. 並依據DFS繼續從新節點持續往下找，直到沒辦法找到任何out-degree vertices為止，我們就將time++，並存入finish array中，代表這個vertex的DFS已經做完。

Q9_Ans (3/11)



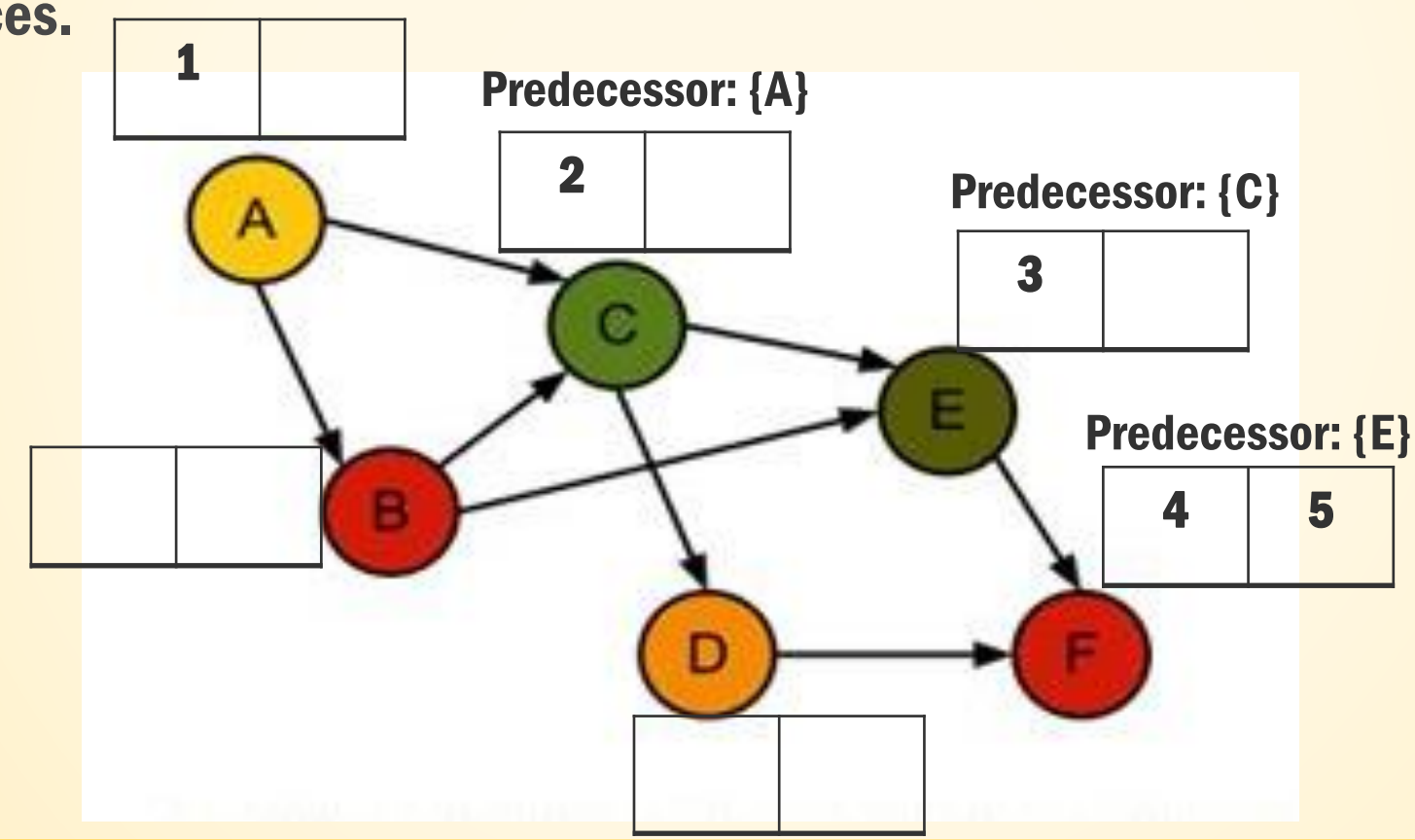
Q9_Ans (4/11)

- Found the first out-degree of vertex (C).



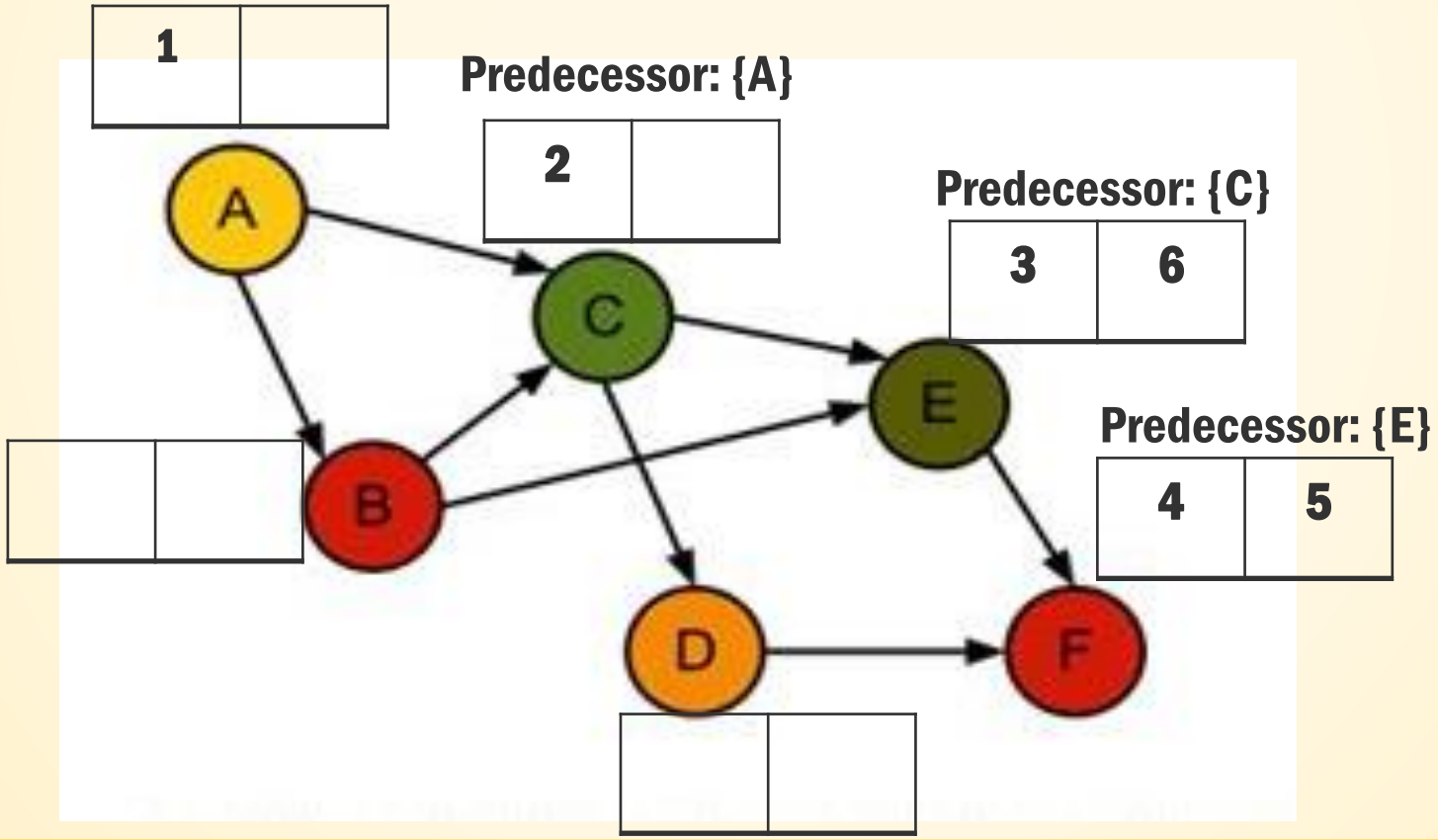
Q9_Ans (5/11)

- Repeat DFS from (C) to (F), we found (F) that would not exist any out-degree of vertices.



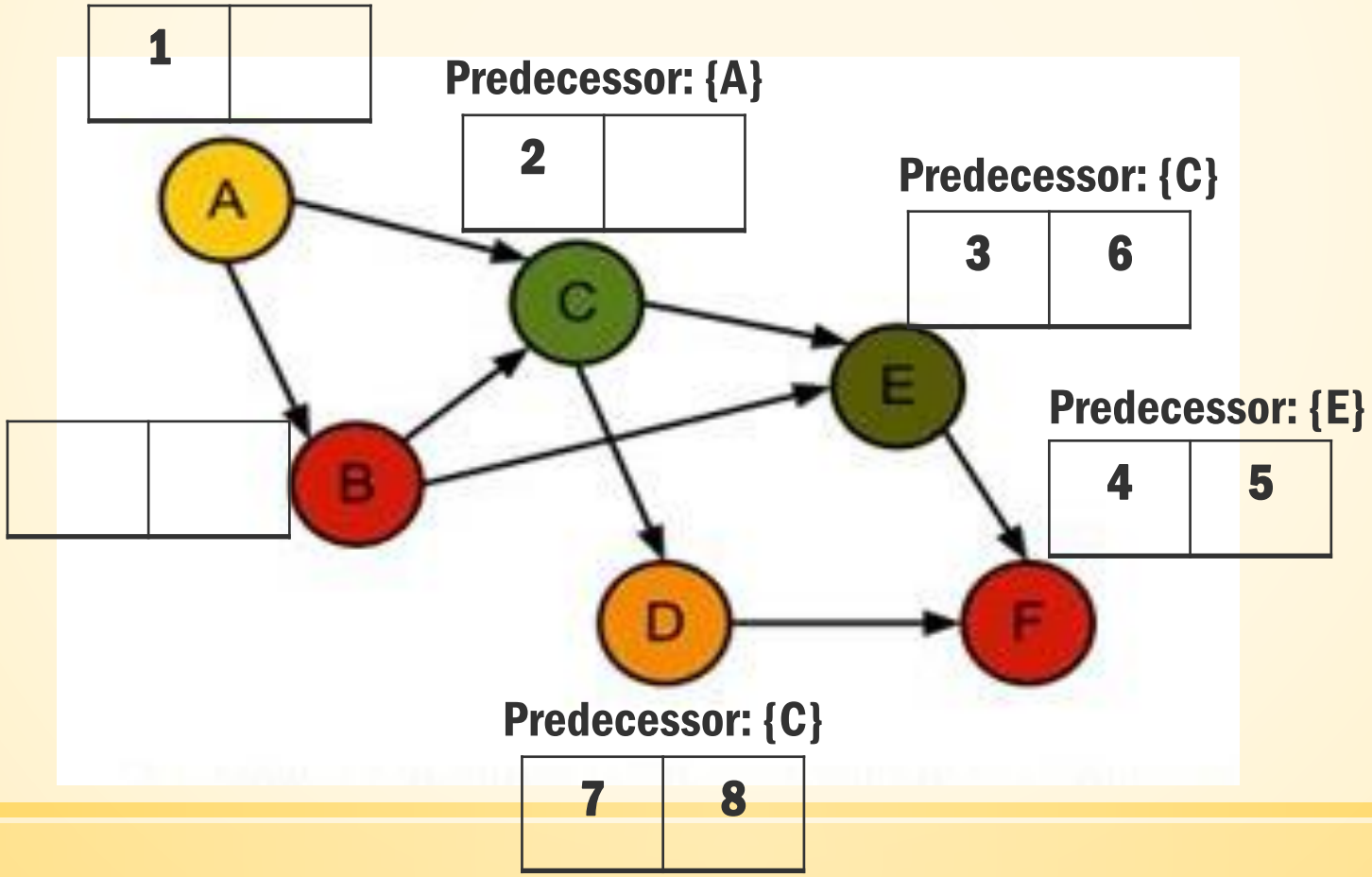
Q9_Ans (6/11)

- Finish (E).



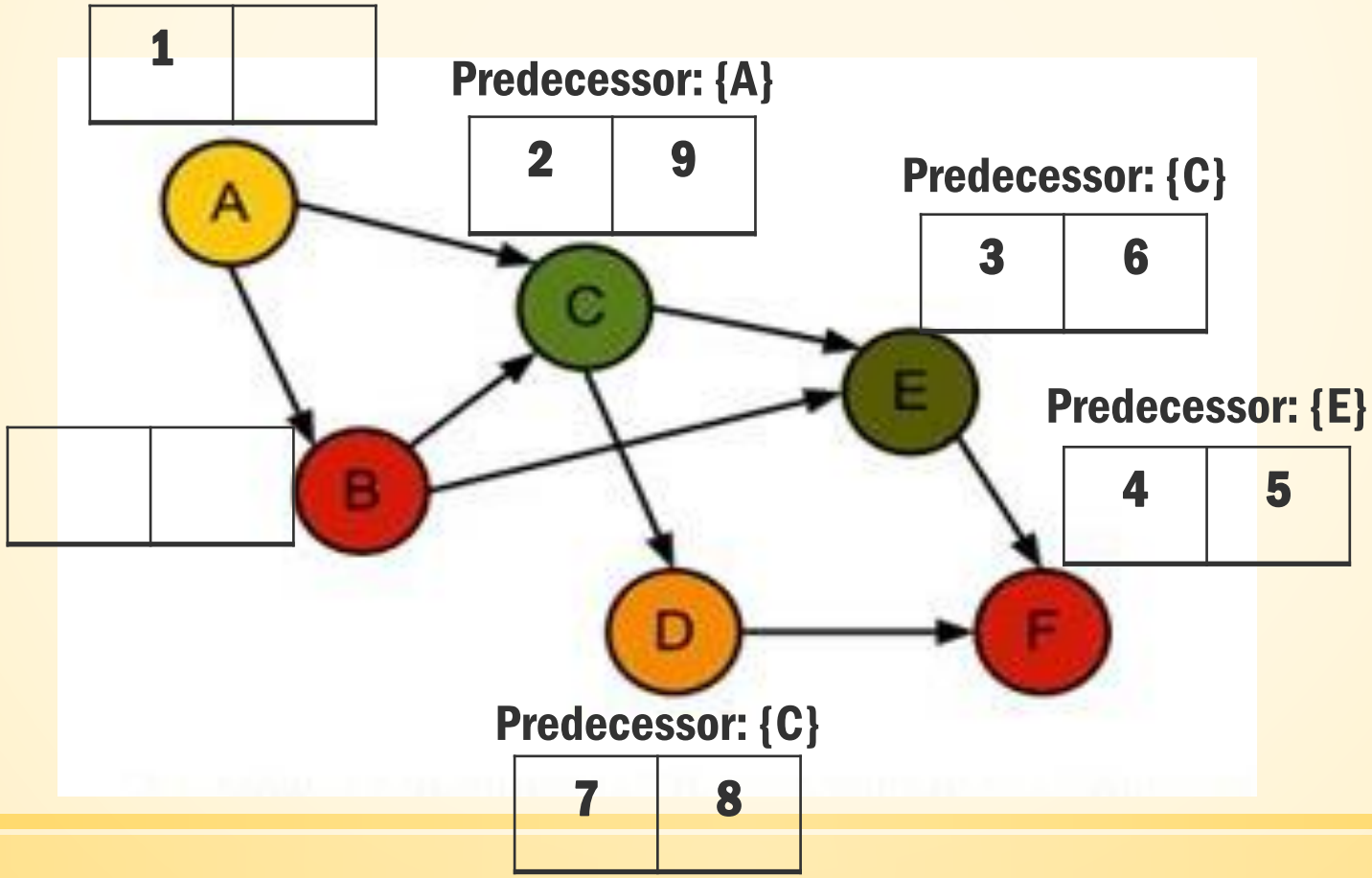
Q9_Ans (7/11)

- Vertex (C) found the first out-degree of vertex (D) in the remaining vertices.



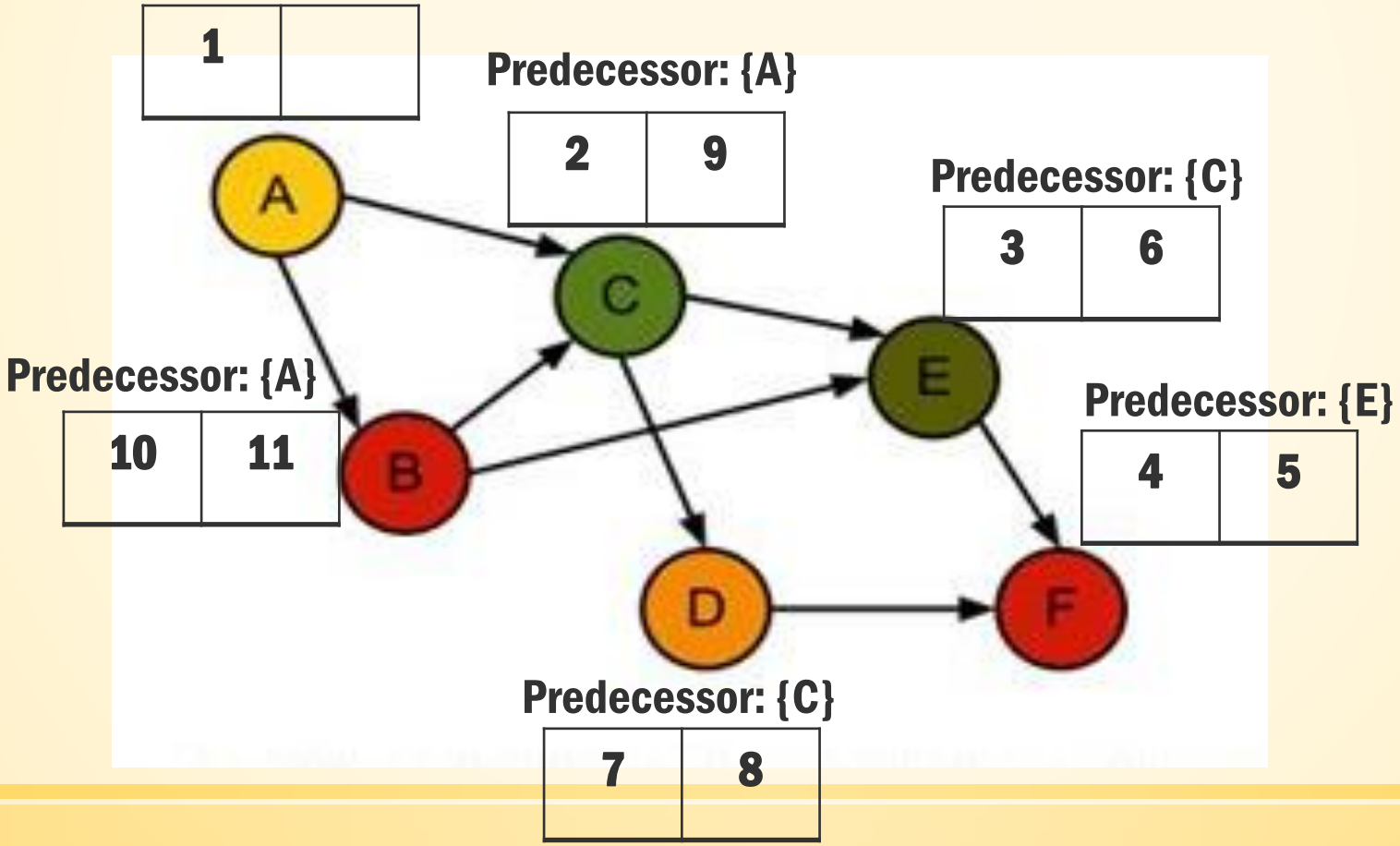
Q9_Ans (8/11)

- Finish Vertex (D).



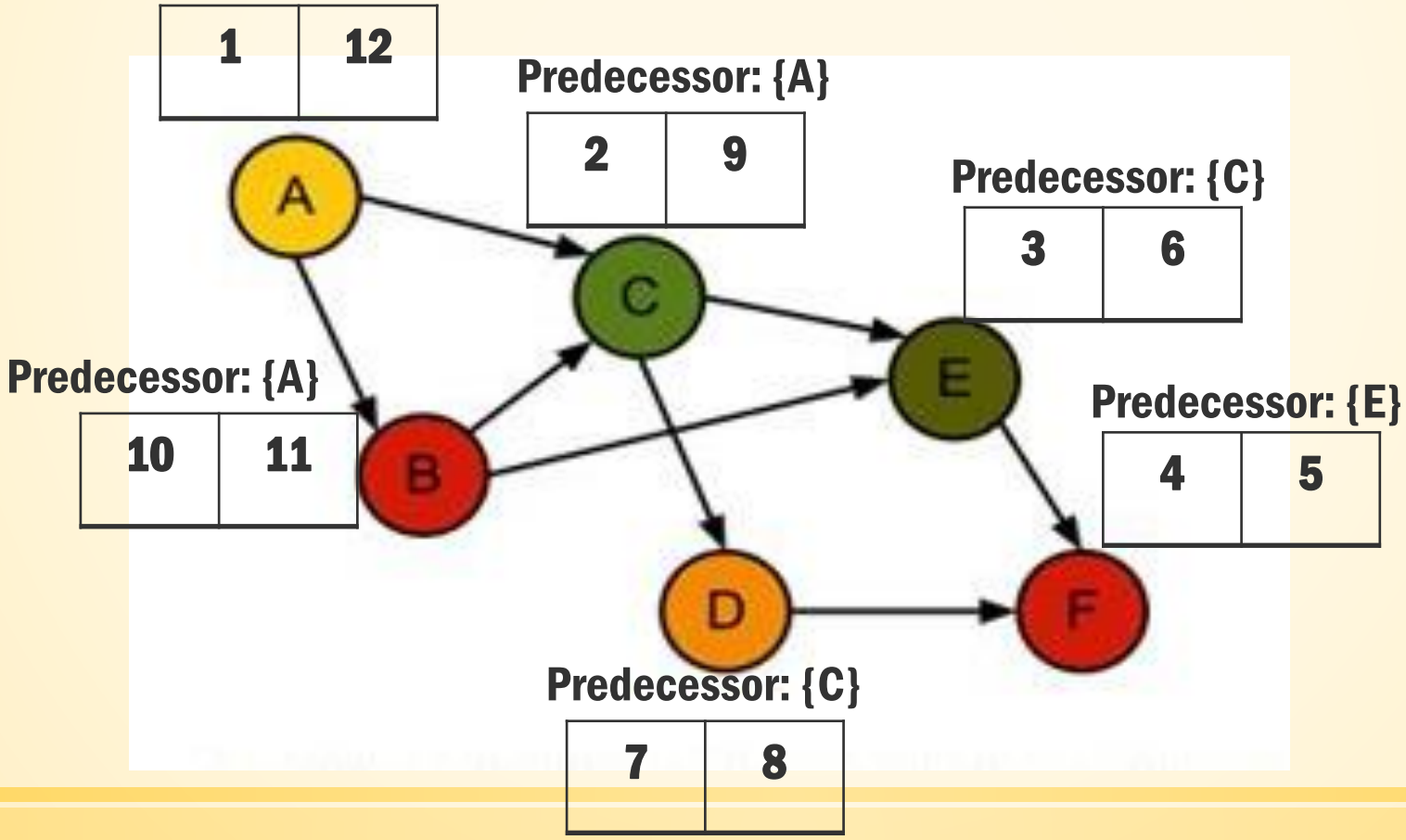
Q9_Ans (9/11)

- Finish Vertex (C).



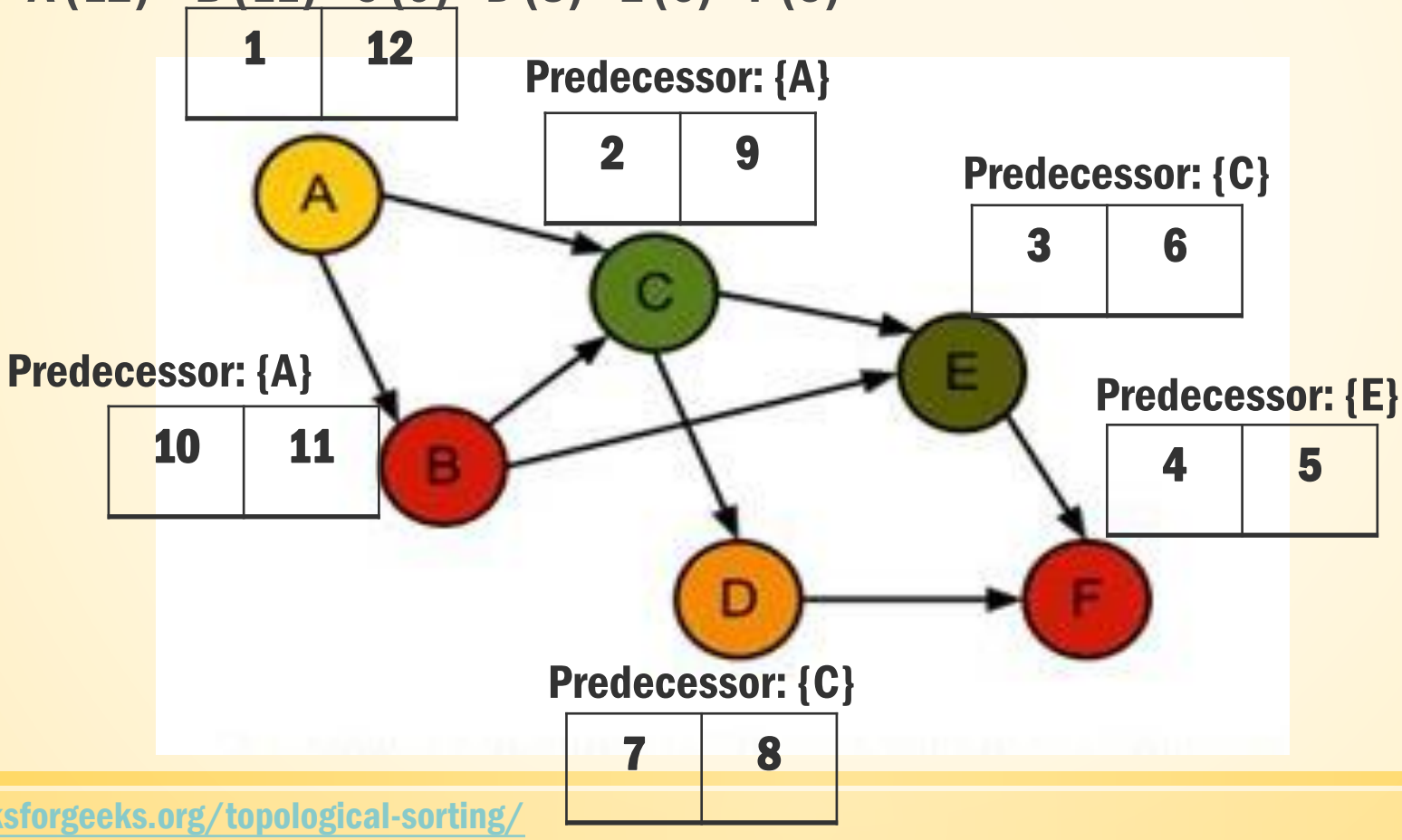
Q9_Ans (10/11)

- Vertex (A) found the first out-degree vertex (B) in the last vertices.



Q9_Ans (11/11)

- The topological order output according to the vertex of finish (from big to small):
  - A (12) > B (11) >C (9) >D (8) >E (6) >F (5)



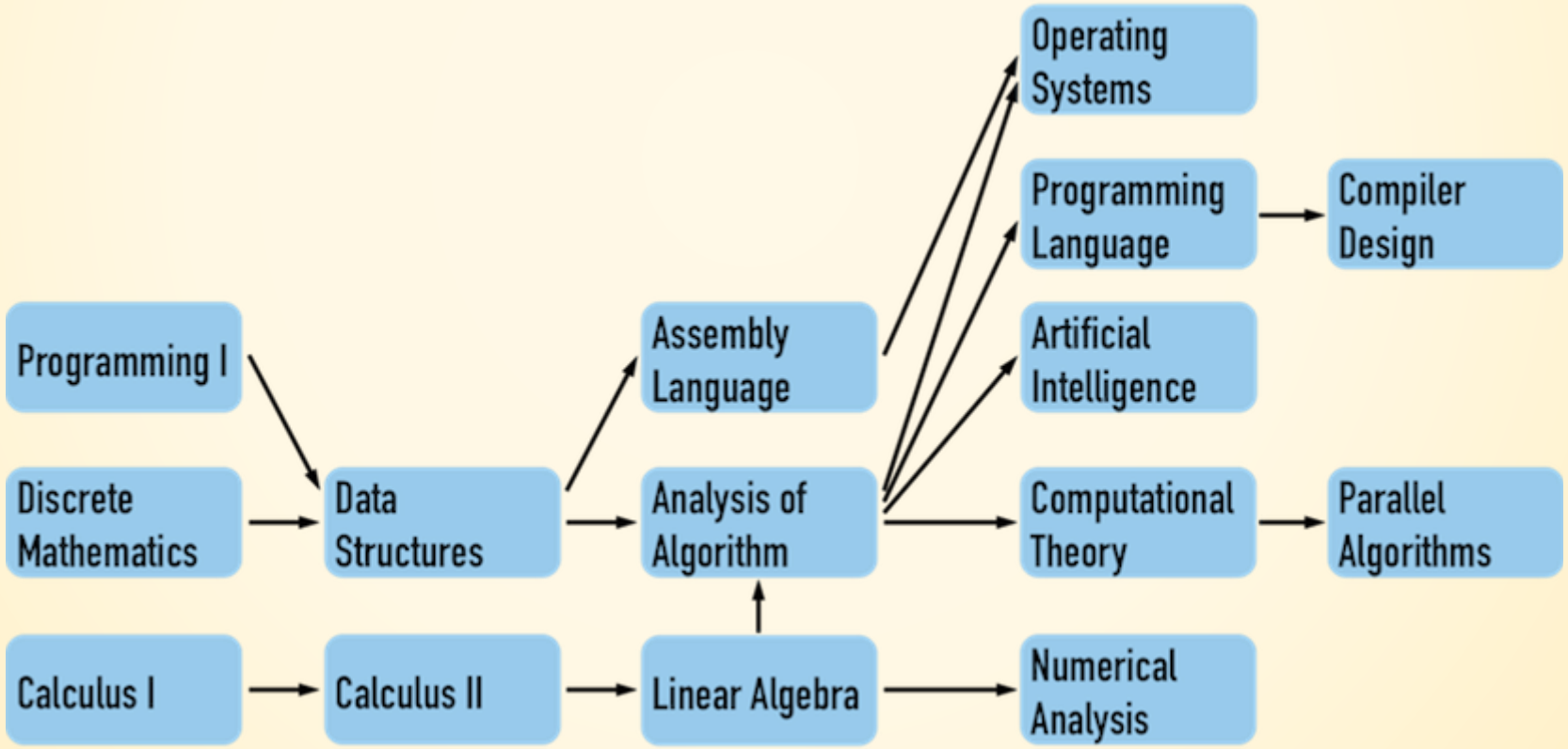
Ref: https://www.geeksforgeeks.org/topological-sorting/

Q10

- Try to come up with a practical example of applying the topological sorting or topological ordering. You may start from the meaning of nodes and edges in your example.

Q10_Ans

- Topological order can solve problems that have a "sequential relationship", and "curriculum and its prerequisites" have this relationship.



Ref: http://alrightchiu.github.io/SecondRound/graph-li-yong-dfsxun-zhao-dagde-topological-sorttuo-pu-pai-xu.html

Q10_Ans

- In symbolic representation, a vertex represents the course map of CS
- For the edge, there are many applications (for examples):
  - AoV: Edge head can be used as a prerequisite course, and then we can sort our course content (topological order) for the graph.
  - AoE: Edge can represent each course. Statistics found how much time a student needs to spend. Then we can use the graph to find the critical path and can focus on critical courses to optimize teaching content for reducing overall time.