Question 1.

a1. All noncyclic paths from A to D
A -> B -> E -> G -> F -> C -> D
A -> B -> E -> G -> F -> H -> D
A -> C -> D
A -> C -> F -> H -> D
A -> G -> F -> C -> D
A -> G -> F -> H -> D

a2. All noncyclic paths from B to H
B -> E -> G -> A -> C -> D -> H
B -> E -> G -> A -> C -> F -> H
B -> E -> G -> F -> C -> D -> H
B -> E -> G -> F -> H

a3. All noncyclic paths from E to C
E -> G -> A -> C
E -> G -> F -> C

b. Give the adjacency matrix representation of the graph.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

c. Give the adjacency list representation of the graph.
A -> B -> C -> G
B -> E
C -> D -> F
D -> H
E -> G
F -> C -> H
G -> A -> F
H -> D
d. Give the depth-first traversal of the graph (supposed we start from A).
Using Alphabetic order: A -> B -> E -> G -> F -> C -> D -> H
e. Give the breadth-first traversal of the graph (supposed we start from A).
Using Alphabetic order: A -> B -> C -> G -> E -> D -> F -> H

Question 2.

a1. All noncyclic paths from A to D
A -> B -> E -> G -> F -> C -> D
A -> B -> E -> G -> F -> H -> D
A -> C -> D
A -> C -> F -> H -> D  
A -> G -> F -> C -> D  
A -> G -> F -> H  

a2. All noncyclic paths from B to H  
B -> A -> C -> D -> H  
B -> A -> C -> F -> H  
B -> A -> G -> F -> C -> D -> H  
B -> A -> G -> F -> H  
B -> E -> G -> A -> C -> D -> H  
B -> E -> G -> A -> C -> F -> H  
B -> E -> G -> F -> C -> D -> H  
B -> E -> G -> F -> H  

a3. All noncyclic paths from E to C  
E -> B -> A -> C  
E -> B -> A -> G -> F -> C  
E -> B -> A -> G -> F -> H -> D -> C  
E -> G -> A -> C  
E -> G -> F -> C  
E -> G -> F -> H -> D -> C  

b. Give the adjacency matrix representation of the graph.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

c. Give the adjacency list representation of the graph.  
A -> B -> C -> G  
B -> A -> E  
C -> A -> D -> F  
D -> C -> H  
E -> B -> G  
F -> C -> G -> H  
G -> A -> E -> F  
H -> D -> F  

d. Give the depth-first traversal of the graph (supposed we start from A).  
Using Alphabetic order: A -> B -> E -> G -> F -> C -> D -> H  
e. Give the breadth-first traversal of the graph (supposed we start from A).  
Using Alphabetic order: A -> B -> C -> G -> E -> D -> F -> H
Question 3.

a. Give the adjacency matrix representation of the graph.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>G</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>H</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

b. Give the adjacency list representation of the graph.

A -> (B,4) -> (C,3) -> (G,1)
B -> (A,4) -> (E,3)
C -> (A,3) -> (D,8) -> (F,5)
D -> (C,8) -> (H,5)
E -> (B,3) -> (G,6)
F -> (C,5) -> (G,2) -> (H,7)
G -> (A,1) -> (E,6) -> (F,2)
H -> (D,5) -> (F,7)

b. Find the shortest path between node B and all other nodes in the above graph.

B -> A (length: 4)
B -> A -> C (length: 7)
B -> A -> C -> D (length: 15)
B -> E (length: 3)
B -> A -> G -> F (length: 7)
B -> A -> G (length: 5)
B -> A -> G -> F -> H (length: 14)

Question 4.

a.

<table>
<thead>
<tr>
<th>Step</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>Put into L</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

A -> C -> D -> B -> E -> G -> F -> H

b. Write pseudocode for the algorithm given in Appendix A.
algorithm TopologicalSorting (val G <Digraph>)

Pre
   G is acyclic

Post
   Return A list contains all vertices which are arranged by the topological order

1. L = new List
2. loop (more vertex v in G)
   1. $d[i] =$ the indegree of $v_i$
3. loop until all vertices of G are put into L
   1. Find the largest $i$ such that $d[i] = 0$ and $v_i$ has not been put into L
   2. L.addlast($v_i$)
3. loop (more vertex $v_j$ adjacent to $v_i$)
   1. if ($v_j$ has not been put in L)
      $d[j] = d[j] - 1$
4. return L
end TopologicalSorting

Question 5.
a. algorithm DepthFirstSearch (val G <Digraph>, val source <Vertex>, val dest <Vertex>)
   search for a path from source to dest using depth-first traversal and then print out the solution.

Pre
   Post
1. loop (more vertex $v$ in G)
   1. predecessor $(v) = \text{null}$
   2. unmark(v)
2. StackObj<Stack>
3. StackObj.Push(source)
4. loop (NOT StackObj.isEmpty())
   1. $w =$ StackObj.Pop() // include Top and Pop
   2. if ($w$ is dest)
      1. StackResult<Stack> // temporary stack to print the solution
      2. loop (NOT predecessor(w) is null)
         1. StackResult.Push (w)
         2. $w =$ predecessor(w)
      3. Print source
   4. loop (NOT StackResult.isEmpty())
      1. $\text{temp} =$ StackResult.Pop()
      2. Print temp
5. return
3. else if (vertex $w$ is unmarked)
   1. mark($w$)
2. loop (more vertex $x$ adjacent to $w$)
   1. StackObj.Push(x)
   2. Predecessor(x) = $w$

end DepthFirstSearch
b. algorithm BreadthFirstSearch (val G <Digraph>, val source <Vertex>, val dest <Vertex>)
search for a path from source to dest using breadth-first traversal and then print out the solution.

Pre
Post

1 loop (more vertex v in G)
    1 predecessor (v) = null
    2 unmark(v)

3 QueueObj<Queue>
3 QueueObj.EnQueue(source)
4 loop (NOT QueueObj.isEmpty())
    1 w = QueueObj.DeQueue() // include QueueFront and DeQueue
    2 if (w is dest)
        1 StackResult<Stack> // temporary stack to print the solution
        2 loop (NOT predecessor(w) is null)
            1 StackResult.Push (w)
            2 w = predecessor(w)
        3 Print source
    4 loop (NOT StackResult.isEmpty())
        1 temp = StackResult.Pop()
        2 Print temp

5 return

3 else if (vertex w is unmarked)
    1 mark(w)

2 loop (more vertex x adjacent to w)
    1 QueueObj.EnQueue(x)
    2 Predecessor(x) = w

end BreadthFirstSearch

c. DepthFirst Search:
A -> B -> E -> G -> F -> H -> J -> L -> O

BreadthFirst Search:
A -> B -> E -> F -> H -> J -> L -> O

Figure 5
d. **algorithm** simulate (val G <Digraph>, val source <Vertex>, val dest <Vertex>)

search for a path from source to dest using depth-first traversal and then print out the solution.

**Pre**

- **loop** (more vertex v in G)
  - predecessor (v) = null
  - unmark(v)
- StackObj<Stack>
- StackObj.Push(source)
- **loop** (NOT StackObj.isEmpty())
  - w = StackObj.Pop() // include Top and Pop
  - **if** (w is dest)
    - StackResult<Stack> // temporary stack to print the solution
      - **loop** (NOT predecessor(w) is null)
        - StackResult.Push (w)
        - w = predecessor(w)
      - Print source
      - **loop** (NOT StackResult.isEmpty())
        - temp = StackResult.Pop()
        - Print temp
    - **return**
  - **else if** (vertex w is unmarked)
    - mark(w)
    - print w // print the current move
    - **loop** (more vertex x adjacent to w AND x is unmarked)
      - StackObj.Push(w)
      - StackObj.Push(x)
      - Predecessor(x) = w
    - **else**
      - print “back to” + w

**end** simulate

**Question 6.**

![Image of a map with labels and connections between cities.](https://fb.com/tailieudientucntt)
b. Represent the map in an adjacency list

A -> (B, 12) -> (E, 18)
B -> (A, 12) -> (C, 3) -> (D, 3)
C -> (B, 3) -> (D, 4) -> (J, 19)
D -> (B, 3) -> (C, 4) -> (E, 7) -> (I, 21)
E -> (A, 18) -> (D, 7) -> (F, 18) -> (G, 31)
F -> (E, 18) -> (G, 12) -> (H, 3)
G -> (E, 31) -> (F, 12) -> (Q, 35)
H -> (F, 3) -> (I, 13) -> (M, 9)
I -> (D, 21) -> (H, 13) -> (J, 4) -> (L, 6)
J -> (C, 19) -> (I, 4) -> (K, 7)
K -> (J, 7) -> (L, 5) -> (O, 14)
L -> (I, 6) -> (K, 5) -> (N, 6)
M -> (H, 9) -> (N, 9) -> (Q, 24)
N -> (M, 9) -> (P, 6) -> (L, 6)
O -> (K, 14) -> (P, 6) -> (Q, 7)
P -> (N, 6) -> (O, 6) -> (Q, 8)
Q -> (G, 35) -> (M, 24) -> (O, 7) -> (P, 8)

c. Represent the map in an adjacency matrix

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
<th>N</th>
<th>O</th>
<th>P</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>7</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>8</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>7</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>0</td>
<td>6</td>
<td>9</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

https://fb.com/tailieudientucntt
d. **algorithm** ShortestPath(val source <Vertex>, val dest <Vertex>, val G <Digraph>)

1. listOfShortestPath.clear()

2. Add source to set S

3. **loop** (more vertex v in digraph) // Initiate all distances from source to v
   1. distanceNode.destination = v
   2. distanceNode.distance = weight of edge(source, v) // =∞ if source is not connected to v
   3. listOfShortestPath.Insert(distanceNode)
   4. predecessor(v) = null

4. **loop** (more vertex not in S) // Add one vertex v to S on each step.
   1. minWeight = infinity // Choose vertex v with smallest distance.
   2. **loop** (more vertex w not in S)
      1. Find the distance x from source to w in listOfShortestPath
      2. if(x < minWeight)
         1. v = w
         2. minWeight = x
         3. Add v to S.
      4. if (v is dest)
         5. StackResult<Stack> // temporary stack to print the solution
           1. **loop** (NOT predecessor(v) is null)
              1. StackResult.Push (v)
              2. v = predecessor(v)
           3. Print source
           4. **loop** (NOT StackResult.isEmpty())
              1. temp = StackResult.Pop()
              2. Print temp

5. return

5. **loop** (more vertex w not in S) // Update distance sof all w not in S
   1. Find the distance x from source to w in listOfShortestPath
   2. alt = minWeight + weight of edge from v to w
   3. if(x > alt)
      1. Update distance from source to w in listOfShortestPath to alt
      2. predecessor(w) = v

end ShortestPath
Appendix A – An implementation of topological sorting

Given a directed graph $G$ whose vertex set $V = \{v_1, v_2, \ldots, v_n\}$. The topological order list $L$ on $V$ can be found applying the following steps:

Initially, make $L$ empty
Construct a vector $deg = \{d_1, d_2, \ldots, d_n\}$ where $d_i$ is the indegree of $v_i$.
Repeat the following tasks until all of vertices in $V$ are put into $L$:
- Find the largest $i$ such that $d_i = 0$ and $v_i$ has not been put into $L$
- Put $v_i$ into $L$
- For each $v_j$ such that $v_j$ is adjacent to $v_i$ and $v_j$ has not been put in $L$, make $d_j = d_j - 1$

Example 1: Consider the graph in Figure 7, we have $V = \{0,1,2,3,4\}$, the initial $deg = \{0,4,1,1,0\}$ and $L = ()$.

At the beginning, we have $d_0 = 0$ and $d_4 = 0$, in the meantime neither 0 nor 4 have been put in $L$. Thus we choose $i = 4$ (the largest) and put 4 into $L$. Thus, $L$ becomes (4). Since 1 and 2 are adjacent to 4, we decrease the value of $d_1$ and $d_2$ accordingly. Therefore, $deg$ becomes $\{0,3,0,1,0\}$.

Similarly, in the next step we choose $i = 2$ and put $i$ into $L$. $L$ becomes (4,2) and $deg = \{0,2,0,1,0\}$. Next, we choose $i = 0$. $L$ becomes (4,2,0) and $deg = \{0,1,0,0,0\}$. Next, we choose $i = 3$. $L$ becomes (4,2,0,3) and $deg = \{0,1,0,0,0\}$. The last one to be put into $L$ is 1, and the final topological list $L$ to be found is (4,2,0,3,1).