COMP 352

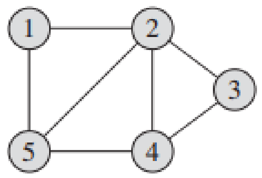
Tutorial Session 11

OUTLINE

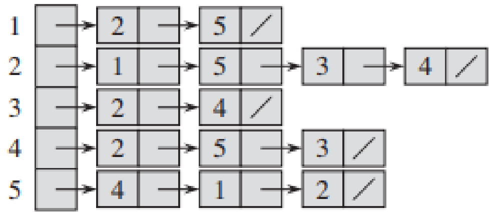
- Graph implementation
- Some graph properties
- Graph Traversal
  - BFS
  - DFS
- Short path algorithm : Dijkstra

Graph:Adjacency List

The *adjacency-list representation* of a graph G = (V, E) consists of an array *Adj* of |V| lists, one for each vertex in V .

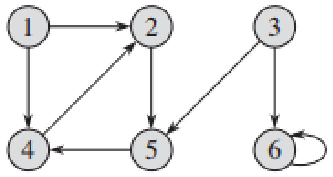


(a)

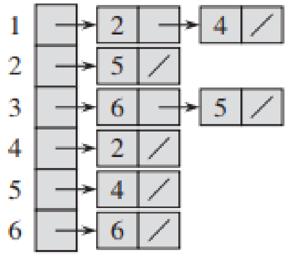


(b)

For each, the adjacency list *Adj*[u] contains all the vertices v such that there is an edge .



(a)

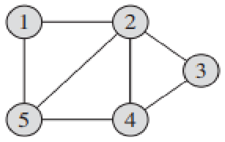


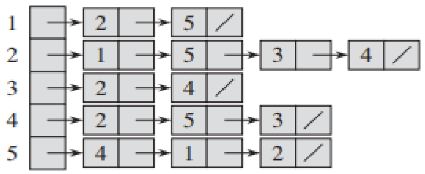
(b)

ADJACENCY MATRIX

For the adjacency-matrix representation of a graph G = (V, E). We assume that the vertices are numbered 1,2, …, |V| in some arbitrary manner.

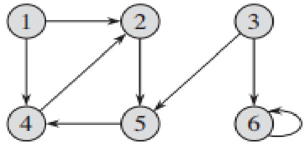
- The *adjacency matrix* of G is the n × n matrix A whose (i, j) entry, denoted by $[A]_{i,j}$, is defined by
- $[A]_{i,j}$ = 1 if nodes i and j are adjacent, 0 otherwise.

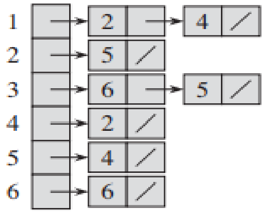

(a)


(b)

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 1 & 1 & 1 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 1 \\ 5 & 1 & 1 & 0 & 1 & 0 \end{array}$$

(c)


(a)


(b)

$$\begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 1 & 0 \\ 3 & 0 & 0 & 0 & 0 & 1 & 1 \\ 4 & 0 & 1 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 1 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}$$

(c)

PROPERTIES OF GRAPH

- **Directed graphs:** edges have a specific direction
- **Undirected graphs:** edges are two-way
- **Weighted graphs:** each edge has an associated weight
- **Sparse graphs:** graphs for which $|E|$ is much less than $|V|^2$
- **Dense graphs:** graphs for which $|E|$ is close to $|V|^2$

where $|E|$ is the number of edges and $|V|$ is the number of nodes

PROPERTIES OF GRAPH: CONT'D

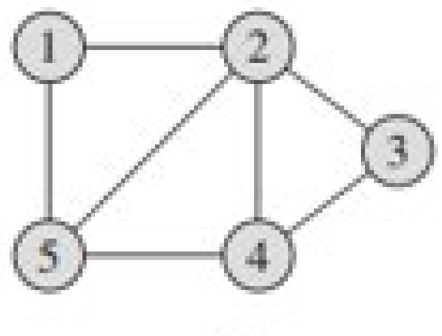
- Cycle : circular sequence of alternating vertices and edges

- Eulerian Paths/cycles: Path/cycle that visits all edges exactly once
- Hamiltonian Paths/Cycles : Path/cycle that visits all vertices exactly once (Other than first and last for cycles)

BFS TRAVERSAL

Traverse broad into the graph by visiting sibling/neighbor nodes before visiting children nodes

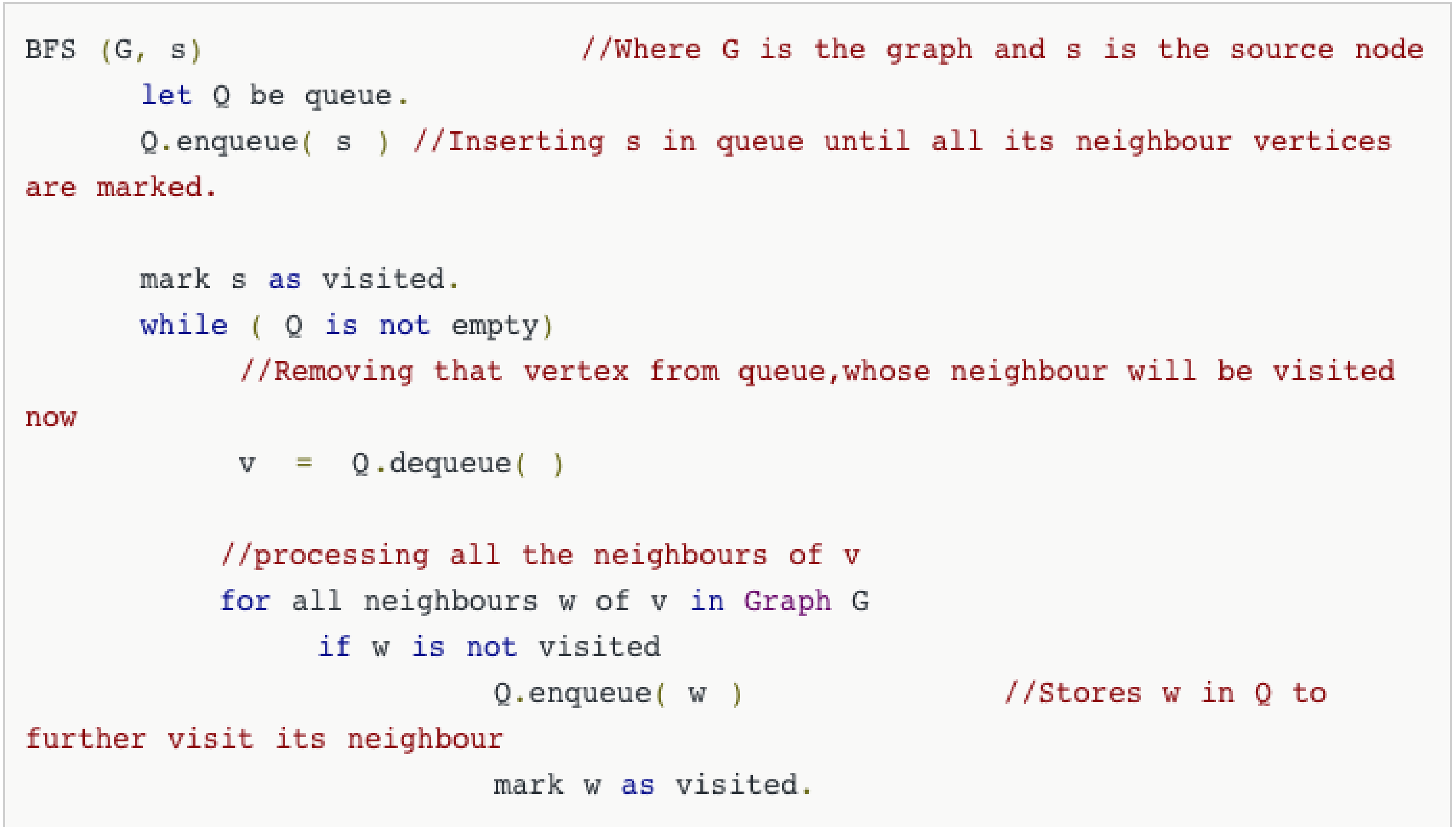
Given a graph G=(V, E) and a distinguished source vertex s, breadth-first search systematically explores the edges of G to **"discover" every vertex that is reachable from** s.



(a)

Uses queue data structure

BFS TRAVERSAL



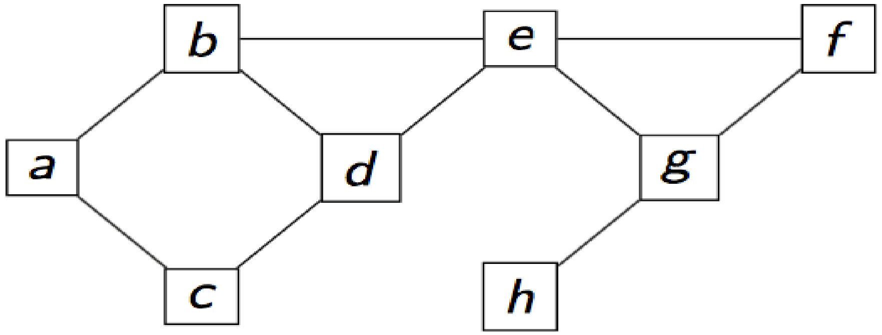
```
BFS (G, s)                   //Where G is the graph and s is the source node
      let Q be queue.
      Q.enqueue( s ) //Inserting s in queue until all its neighbour vertices
are marked.

      mark s as visited.
      while ( Q is not empty)
           //Removing that vertex from queue,whose neighbour will be visited
now
           v  =  Q.dequeue( )

          //processing all the neighbours of v
          for all neighbours w of v in Graph G
               if w is not visited
                        Q.enqueue( w )             //Stores w in Q to
further visit its neighbour
                        mark w as visited.
```

BFS TRAVERSAL

How to mark as visited?

- Use an extra array of visited and unvisited nodes
- Use an extra parameter "color" at each nodes to describe the fact it has been visited or not
- Complexity: O(|V|+|E|)

Example:



Some Applications of Breadth First Traversal

- **Shortest Path** between two nodes u and v**:** path with least number of edges.
- **Social Networking Websites:** to find people within a given distance 'k' from a person
- **Networking:** to broadcast packets such that it will reach targets in minimal time, or finding optimal neighbors.

Source: https://www.geeksforgeeks.org/applications-of-breadth-first-traversal/

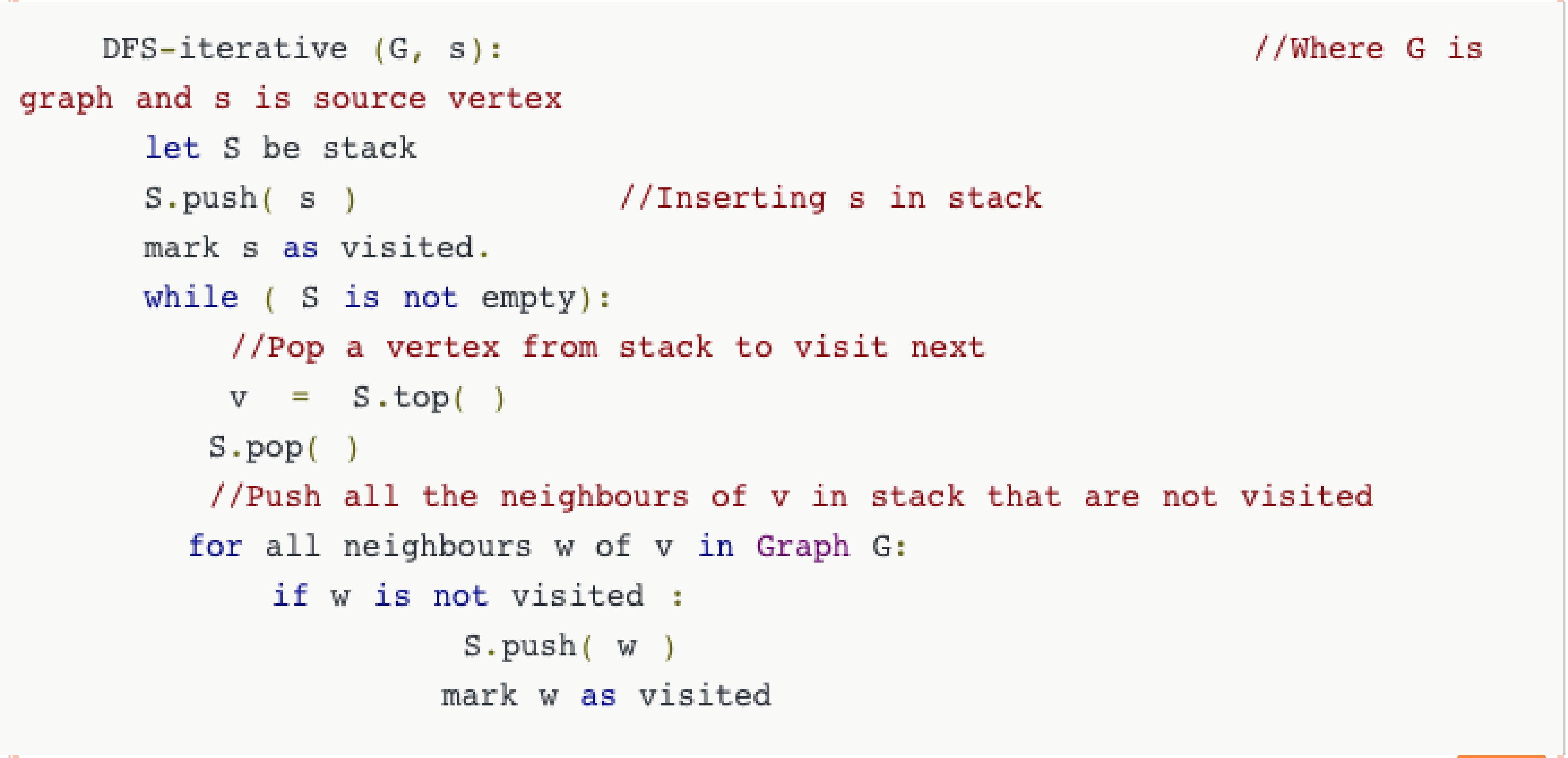
DFS TRAVERSAL

Traverse deep into the graph by visiting children nodes before visiting sibling neighbor nodes

The strategy followed by depth-first search is, as its name implies, to **search "deeper" in the graph whenever possible**. Depth-first search explores edges out of the most recently discovered vertex v that still has unexplored edges leaving it.

Use stack data structure

DFS TRAVERSAL



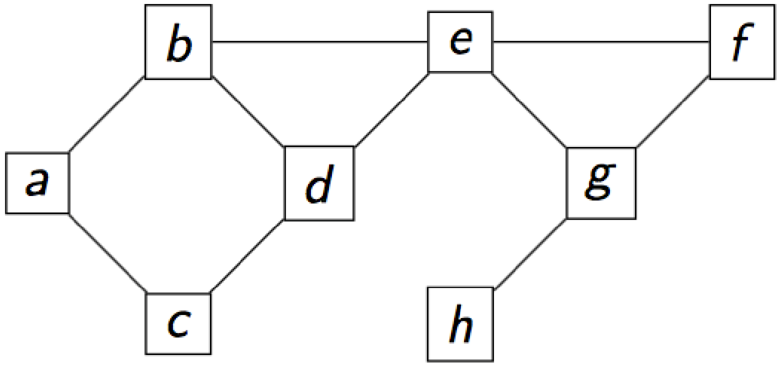
```
    DFS-iterative (G, s):                                   //Where G is
graph and s is source vertex
      let S be stack
      S.push( s )            //Inserting s in stack
      mark s as visited.
      while ( S is not empty):
          //Pop a vertex from stack to visit next
          v  =  S.top( )
         S.pop( )
         //Push all the neighbours of v in stack that are not visited
        for all neighbours w of v in Graph G:
            if w is not visited :
                     S.push( w )
                    mark w as visited
```

DFS TRAVERSAL

How to mark as visited? As previously

- Use an extra array of visited and unvisited nodes
- Use an extra parameter "color" at each nodes to describe the fact it has been visited or not
- Complexity: O(|V|+|E|)

Example:



Some Applications of depth First Traversal

- **Detecting cycle in a graph**
- **Finding Strongly Connected Components of a graph** A directed graph is called strongly connected if there is a path from each vertex in the graph to every other vertex
- **Topological sorting:** used when we want to sort a set of task based on their dependencies. The task are the vertices, and there is an edge from *x* to *y* if task *x* must be completed before job *y* can be started. The topological sort give the order in which we will execute the task. **Example :** determining the order of compilation tasks to perform in makefiles

https://www.geeksforgeeks.org/applications-of-depth-first-search/

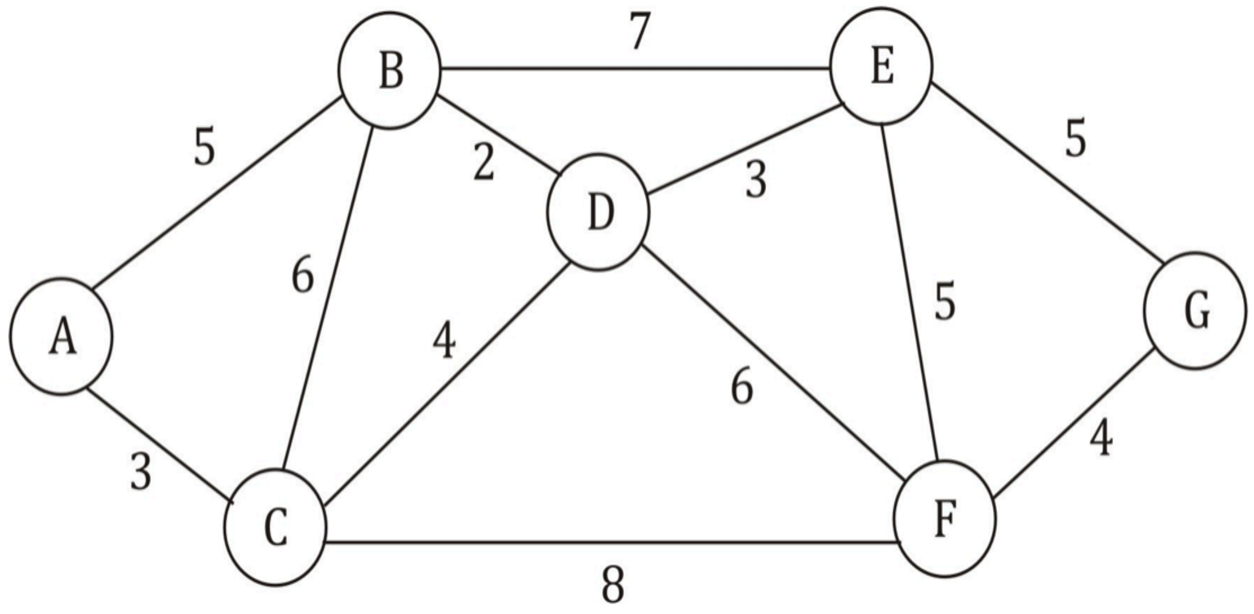
Shortest path in graph: Dijkjstra

- Dijkstra's algorithm is one of the most useful graph algorithms
- Find the shortest path is a weighted graph (directed or not)
- Example of real life problems:
  - Same principle in google map to find your path from one point to another
  - Traveling on a budget: What is the cheapest airline schedule from Seattle to city X?
  - Optimizing routing of packets on the internet
  - Shipping: Find which high ways and roads to take to minimize total delay due to traffic

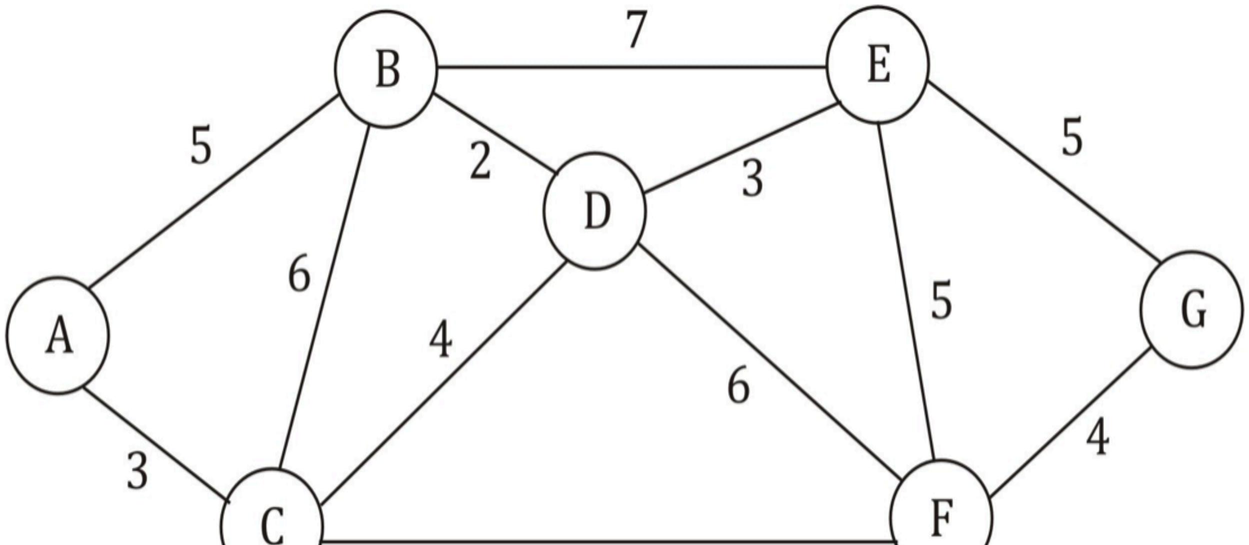
Dijkstra algorithm: Pseudo code

```
function dijkstra(G, S) :
for each vertex V in G
        distance[V] <- infinite
        previous[V] <- NULL
        If V != S, add V to Priority Queue Q
distance[S] <- 0
while Q IS NOT EMPTY
    U <- Extract MIN from Q
    for each unvisited neighbour V of U
        tempDistance <- distance[U] + edge_weight(U, V)
        if tempDistance < distance[V]
                distance[V] <- tempDistance
                previous[V] <- U
return distance[], previous[]
```

Dijkstra example



Step	Q	D(A),P(A)	D(B),P(B)	D(C),P(C)	D(D),P(D)	D(E),P(E)	D(F),P(F



MCQ 1

Adjacency matrix of all graphs are symmetric.

a) True

b) False

MCQ 2

If the origin and terminus of a walk are same, the walk is known as... ?

a) Open
b) Closed
c) Path
d) None of these

MCQ 3

The degree of any vertex of graph is …. ?

a) The number of edges incident with vertex
b) Number of vertex in a graph
c) Number of vertices adjacent to that vertex
d) Number of edges in a graph

MCQ 4

For the adjacency matrix of a directed graph the row sum is the __________ degree and the column sum is the _________ degree.

a) in, out
b) out, in
c) in, total
d) total, out

MCQ 5

Which of the following is false in the case of a spanning tree of a graph G?

a) It is tree that spans G
b) It is a subgraph of the G
c) It includes every vertex of the G
d) It can be either cyclic or acyclic

MCQ 6

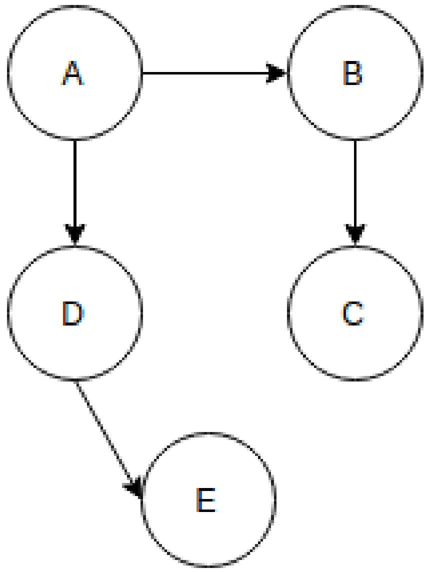
Which of the following statements is true concerning the time complexity of checking if an edge exists between two particular vertices depends on if the graph is stored using adjacency matrix?

a) Depends on the number of edges
b) Depends on the number of vertices
c) It is independent of both the number of edges and vertices
d) It depends on both the number of edges and vertices

MCQ 7

What could be possible DFS of the following graph ?

a) ABCDE
b) AEDCB
c) EDCBA
d) ADEBC


$$\begin{tikzcd}
A \arrow[r] \arrow[d] & B \arrow[d] \\
D \arrow[dr] & C \\
& E
\end{tikzcd}$$

MCQ 8

The Depth First Search traversal of a directed graph will result into?

a) Linked List
b) Tree
c) Graph with back edges
d) None of the mentioned

MCQ 9

A person wants to visit some places. He starts from a vertex and then wants to visit every vertex till it finishes from one vertex, backtracks and then explore other vertex from same vertex. What algorithm he should use?

a) Depth First Search
b) Breadth First Search
c) Post order traversal
d) None of the mentioned

MCQ 10

When the Breadth First Search of a graph is unique?

a) When the graph is a Binary Tree
b) When the graph is a Linked List
c) When the graph is a n-ary Tree
d) None of the mentioned

MCQ 11

Which of the following problems is similar to that of a Hamiltonian path problem?

a) knapsack problem
b) closest pair problem
c) travelling salesman problem
d) assignment problem

MCQ 12

A graph G is called a ….. if it is a connected acyclic graph ?

a) Cyclic graph
b) Regular graph
c) Tree
d) Not a graph

Exercise

Perform a *breadth-first search* of the following graph, where E is the starting node. In other words, show the output if we issue the call BFS(E). Provide two cases: (a) Use a counterclockwise ordering from the top (12 o'clock position); and (b) Use a clockwise ordering from the top.