Data Structure

تركيب بيانات
الفرقة الثالثة علوم إحصاء وعلوم الحاسب

By

Dr. Reda Elbarougy

د/ رضا الباروجى

Lecturer of computer sciences

In Mathematics Department

Faculty of Science

Damietta University

الاحد ٢٦-٠٤-٢٠١٧

رقم المحاضرة

م	التاريخ	الموضوع	الحالة	
١	٢٠١٧-٠٢-١٢	الفصل الأول مقدمة	تم	
٢	٢٠١٧-٠٢-١٩	الفصل الأول مقدمة	تم	
٣	٢٠١٧-٠٣-٠٥	الفصل الثانى اساسيات	تم	
٤	٢٠١٧-٠٣-١٢	الفصل الرابع المصفوفات	تم	
٥	٢٠١٧-٠٣-١٩	الفصل الرابع المصفوفات	تم	
٦	٢٠١٧-٠٣-٢٦	الفصل الخامس		
٧	٢٠١٧-٠٤-٠٢	الفصل الخامس		
٨	٢٠١٧-٠٤-٠٤	الفصل السادس	أضافى	**Stacks**
٩	٢٠١٧-٠٤-٠٩	الفصل السادس		
	٢٠١٧-٠٤-١٦	أجازة		
١٠	٢٠١٧-٠٤-١٩	لا		
١١	٢٠١٧-٠٤-٢٣	الفصل السابع		
١٢	٢٠١٧-٠٤-٣٠			
	٢٠١٧-٠٥-٠٧			

Chapter 8: Graph II

Chapter 8

- Drawbacks of sequential representation
- Linked Representation of a graph
- Traversing a Graph
    1) Breadth – First Search (DFS): preorder traversal
    2) Depth – First Search (BFS): level order traversal

Drawbacks of sequential representation

$$A = \begin{array}{c} \\ x \\ y \\ z \\ w \end{array} \begin{array}{c} \begin{array}{cccc} x & y & z & w \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

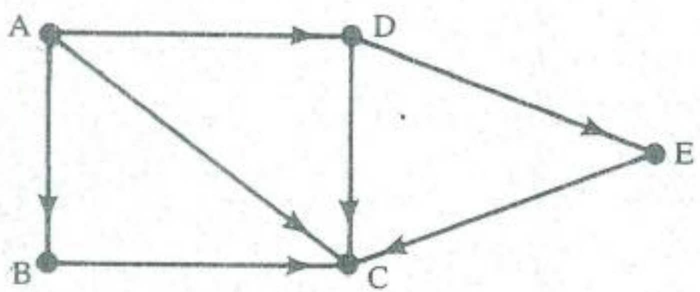
- Let G be a directed graph with m nodes .
- The sequential representation of G has major drawbacks as

1) It is difficult to insert & delete nodes in G .This is because the size of A may need to be changed & the nodes may need changed & the nodes may need to be reordered, so there may be many, many changes in the matrix A.
2) If the number of edges is O(m) or O ($m \times \log_2 \times m$), then the matrix A will be sparse (will contain many zeros) ; hence large memory space will be wasted .

Linked Representation of a graph

Graph G is also represented in memory by a linked representation also called an adjacency structure.

- Consider the following graph G (Fig a)
- The table in fig. (b) shows each node in G followed by its adjacency list, which is its list of adjacent nodes, also called its successors or neighbour's.



(a) Graph *G*.

Node	Adjacency List
A	B, C, D
B	C
C	
D	C, E
E	C

(b) Adjacency lists of *G*.

Fig. 8-7

Linked Representation of a graph

- The linked representation will contain two lists (or files), a node list NODE & an edge list EDGE, as follows:

a) Node list:- Each element in the list Node will correspond to a node in G , & it will be record of the form :
   - NODE will be the name or key value of the node
   - NEXT will be pointer to the first element in the adjacency list of the node, which is maintained in the list EDGE.
   - Shaded area indicated other information in the record such as indegree, outdegree of the node.

NODE	NEXT	ADJ	

Linked Representation of a graph

b) Edge list:- Each element in the list EDGE will correspond to an Edge of G & will be a record of the form: Where Field:

- DEST will point to the location in the list NODE of the destination or terminal node of the edge.
- LINK will link together the edge with the same initial node, that is, the nodes in the same adjacency list.
- Shaded area indicated the other information like weight or label of edge

Linked Representation of a graph

Following Fig . shows the schematic diagram of a linked representation of graph G in Fig(8.7 a).

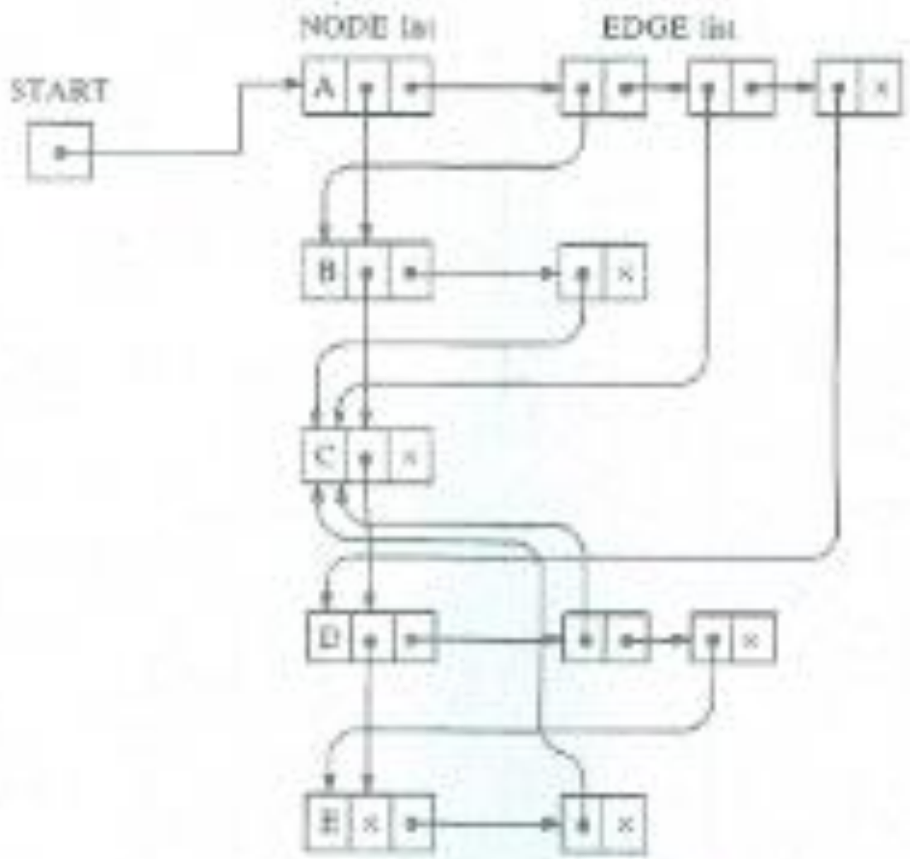


Fig. 8-8

Linked Representation of a graph

Following Fig. shows the memory representation.

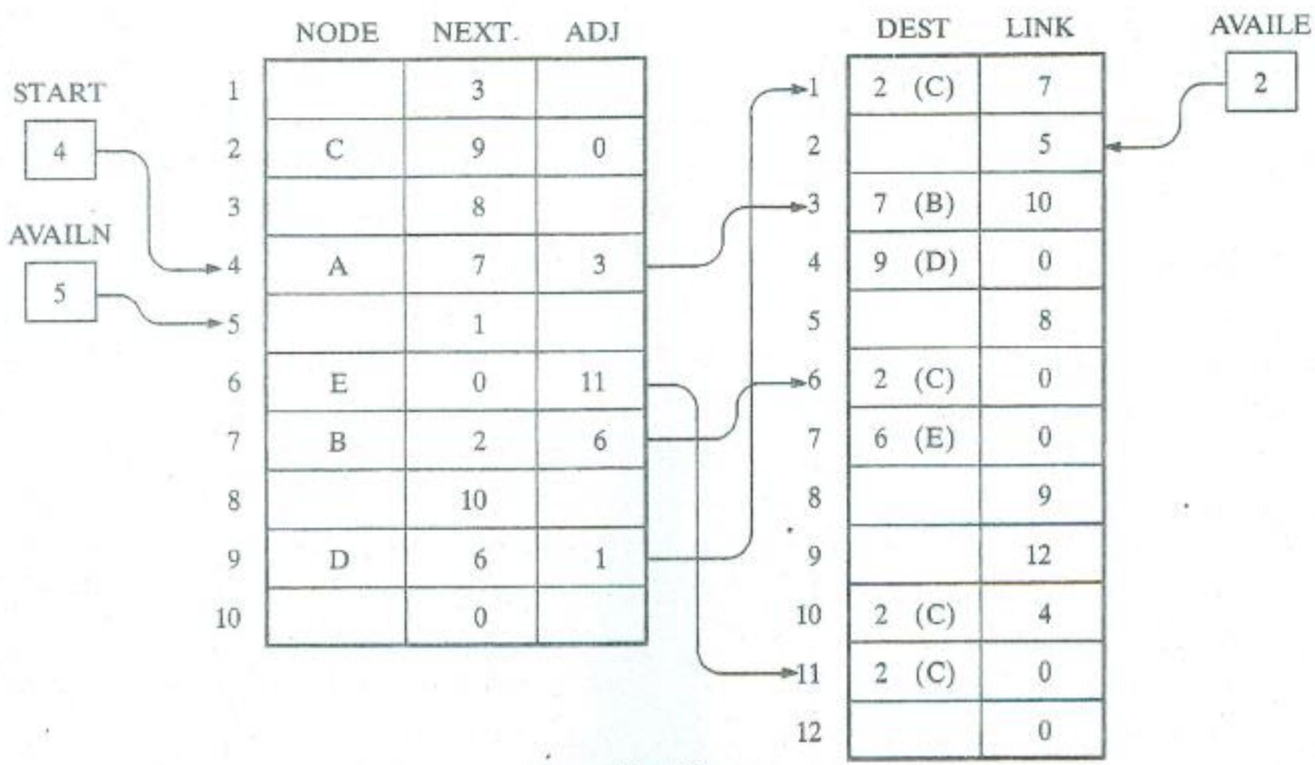


Fig. 8-9

Example 8.5

Suppose friendly airways has nine daily flights ,as follows :

103 Atlanta to Houston	301 Denver to Rene
106 Houston to Atlanta	305 Chicago to Miami
201 Boston to Chicago	308 Miami to Boston
203 Boston to Denver	402 Reno to Chicago
204 Denver to Boston	

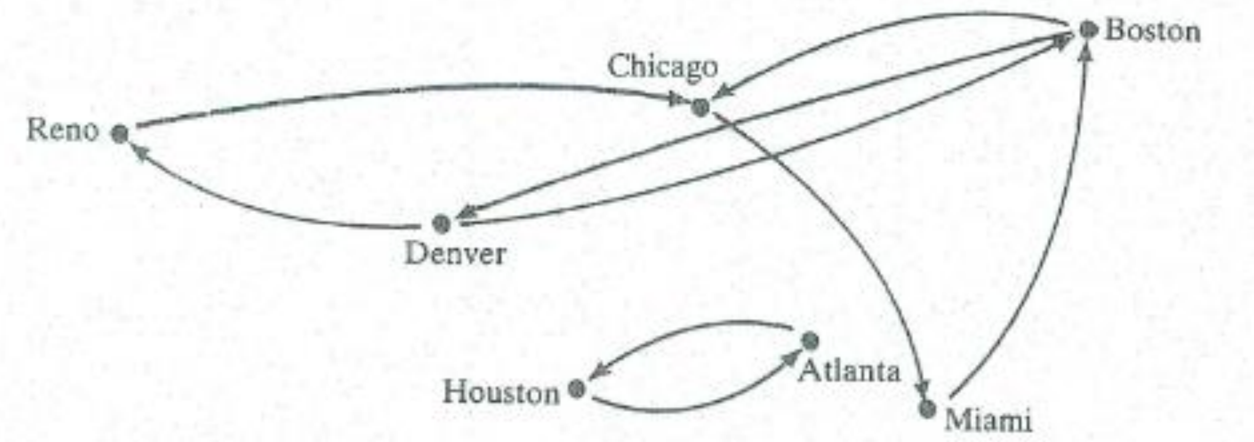


Fig. 8-10

Example 8.5

Following Fig. shows the graph appear in memory using the linked representation.

NODE list

	CITY	NEXT	ADJ
1		0	
2	Atlanta	12	1
3	Chicago	11	7
4	Houston	7	2
5		6	
6		8	
7	Miami	10	8
8		9	
9		1	
10	Reno	0	9
11	Denver	4	5
12	Boston	3	3

START = 2, AVAILN = 5

EDGE list

	NUMBER	ORIG	DEST	LINK
1	103	2	4	0
2	106	4	2	0
3	201	12	3	4
4	203	12	11	0
5	204	11	12	6
6	301	11	10	0
7	305	3	7	0
8	308	7	12	0
9	402	10	3	0
10				11
11				12
12				0

AVAILE = 10

Fig. 8-11

Traversing a Graph

- There are two methods for traversing a graph.
  1) Breadth – First Search (DFS): preorder traversal
  2) Depth – First Search (BFS): level order traversal
- The BFS uses a queue as an auxiliary structure to hold nodes for future processing & the DFS uses stacks.
- During the execution of algorithm, each node N of G will be in one of three states, called the status of N as follows.

Traversing a Graph

- STATUS =1 : (Ready state)

  The initial state of the node N.
- STATUS =2 : (Waiting state)

  The node N is on the queue or stack , waiting to be processed.
- STATUS =3: (Processed State) The node N has been processed.

1) Breadth – First Search

The general idea behind a breadth first search beginning as a staring node A is as follows.

- First we check the starting node A.
- Then we check all the neighbours of A.
- Then we check all the neighbours of the neighbours of A and so on.
- In this way we need to keep track of the neighbours of a node, and we need to guarantee that no node is processed more than once.
- This is accomplished by using queue to hold nodes that are waiting to be processed. And using a field STATUS which tells us the current status of any node.

1) Breadth – First Search

Algorithm BFS:- This algorithm executes a breadth- first search on a graph G beginning at a starting node A.

1.Initialize all nodes to the ready state (STATUS =1).

2.Put the starting node A in QUEUE & change its status to the waiting state (STATUS = 2).

3.Repeat steps 4 & 5 until QUEUE is empty:

4.Remove the front node N of QUEUE.

Process N & change the status of N to the processed state (STATUS =3) .

5.Add to the rear of QUEUE all the neighbors of N that are in the ready state (STATUS = 1), and change their status to the waiting state (STATUS = 2) .

[End of Step 3 Loop .]

6.Exit.

Example 8.7

Consider the following graph G in following Fig. Suppose G represents the daily flights between cities of some airline and suppose we want to fly from city A to city J with minimum number of stops. In other words, we want the minimum path P from A to J. (Where each edge has length 1).

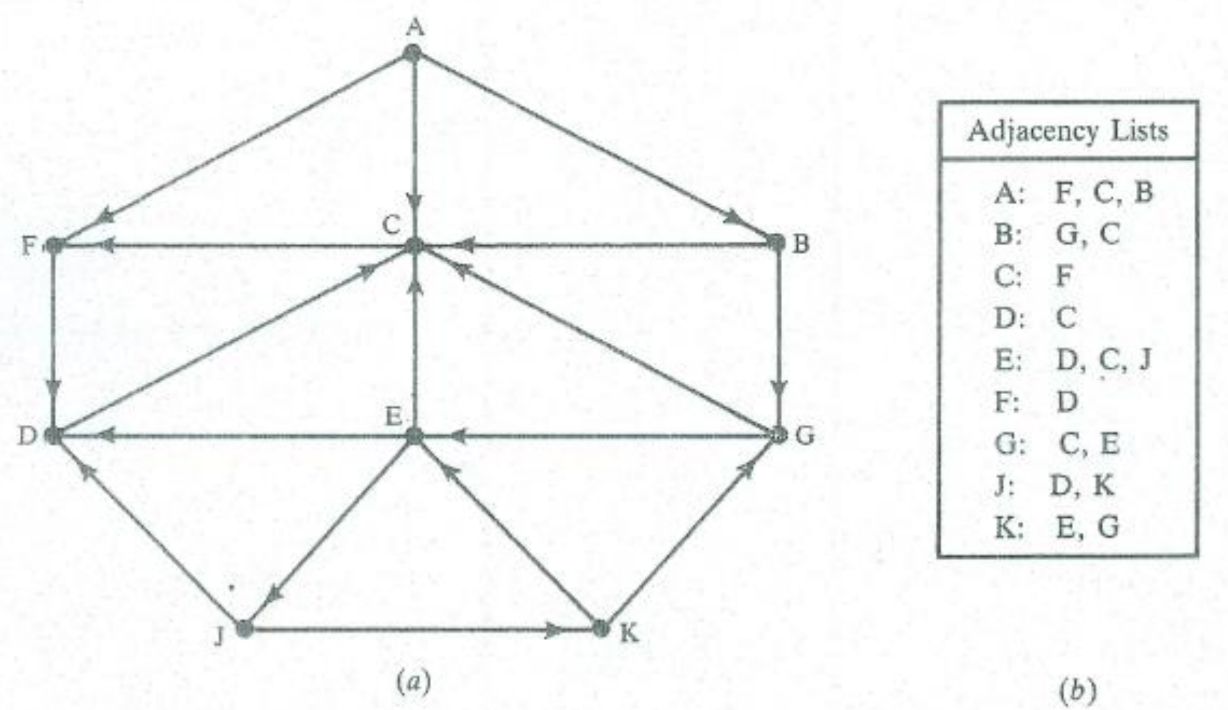


Fig. 8-14

Example 8.7

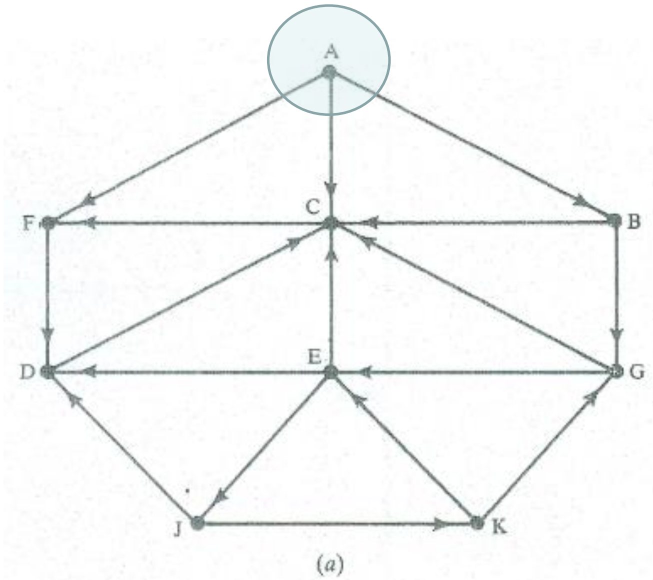
- The minimum path P can be found by using a Breadth-First-search beginning at city A & ending when J is encountered.
- During the execution of the search, we will also keep track of the origin of each edge by using an array ORIG together with the array QUEUE.
- The steps of search is as follows:

Example 8.7

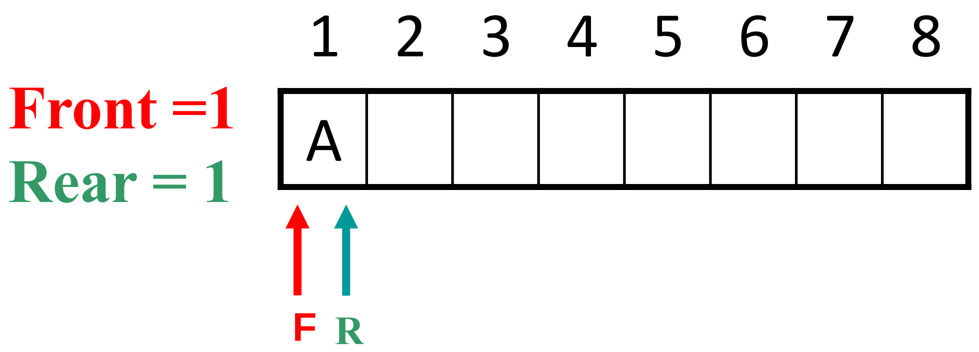
a) Initially, add **A** to QUEUE & add NULL to ORIG as follows:

FRONT =1 QUEUE : A

REAR = 1 ORIG : ϕ



(a)

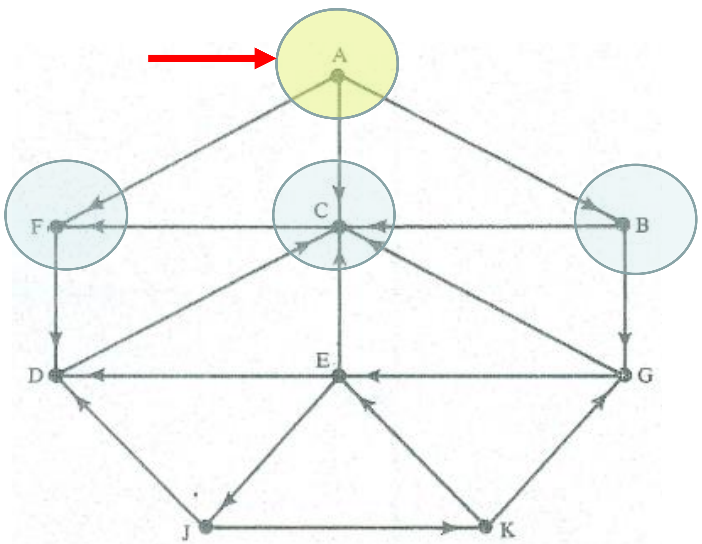


Example 8.7

b) Remove the front element A from QUEUE by setting FRONT := FRONT + 1, & add to QUEUE the neighbors of A as follows.

FRONT = 2 QUEUE : A, F, C, B

REAR = 4 ORIG : ϕ, A, A, A



(a)

Front = 2

Rear = 4

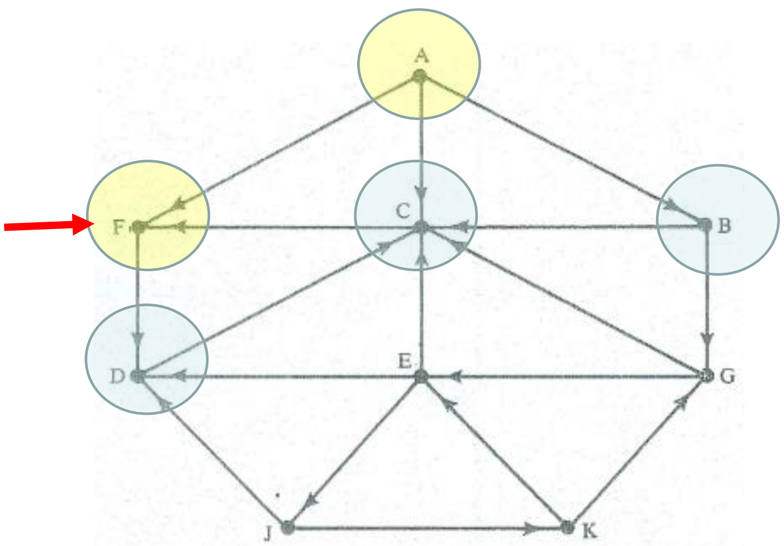
1	2	3	4	5	6	7	8
A	F	C	B				
	F		R				

Example 8.7

c) Remove the front element F from QUEUE by setting FRONT := FRONT + 1, & add to QUEUE the neighbors of F as follows.

FRONT = 3   QUEUE : A, F, C, B, D

REAR = 5   ORIG : ϕ, A, A, A, F



(a)

Front = 3
Rear = 5

1	2	3	4	5	6	7	8
A	F	C	B	D			

F   **R**

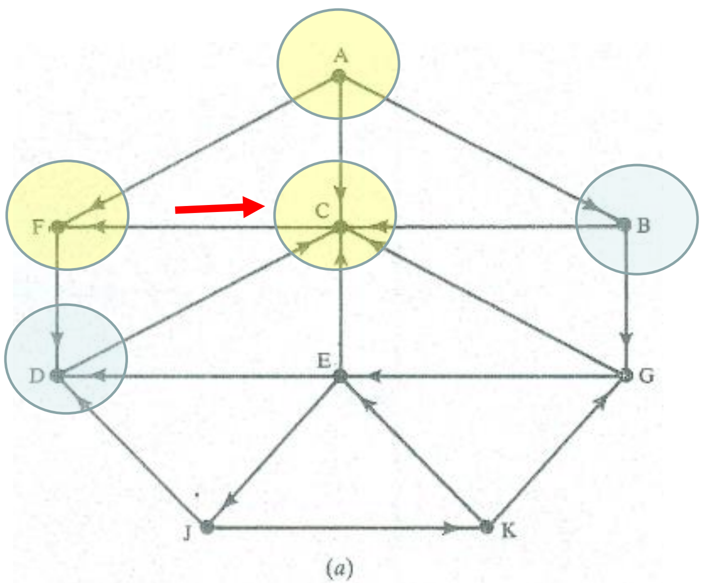
Example 8.7

d) Remove the front element C from QUEUE , & add to QUEUE the neighbors of C as follows.

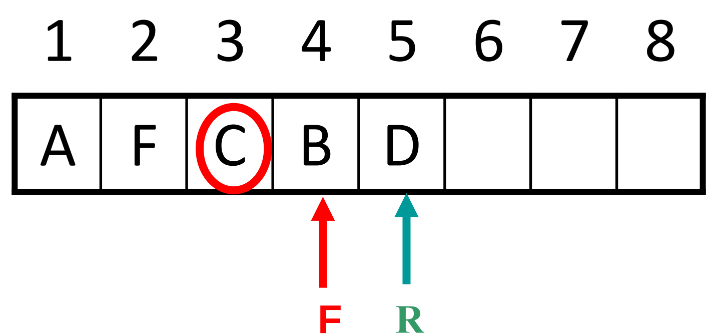
FRONT = 4   QUEUE : A, F, C, B, D

REAR = 5   ORIG : ϕ, A, A, A, F

Note that the neighbors of C i.e. F is not added to QUEUE, since F is not in the ready state (because F is already been added to QUEUE)



Front = 4
Rear = 5



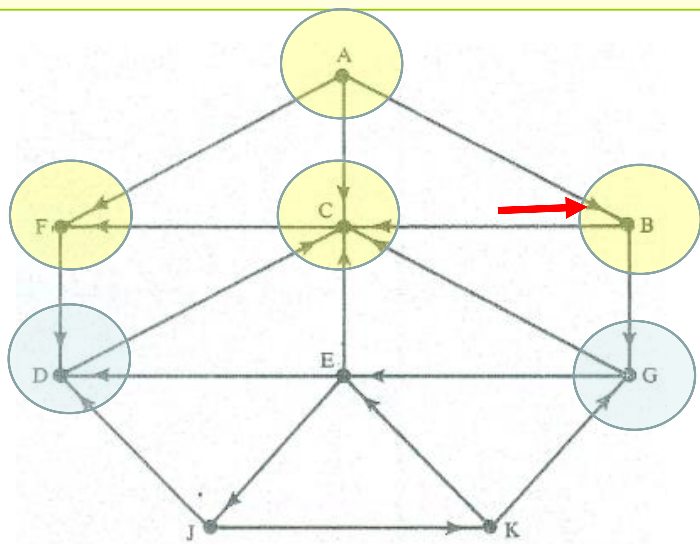
Example 8.7

e) Remove the front element B from QUEUE , & add to QUEUE the neighbors of B as follows.

FRONT = 5   QUEUE : A, F, C, B, D, G

REAR = 6   ORIG : ϕ, A, A, A, F, B

Note that only G is added to QUEUE, since the other neighbor, C is not in the ready state. (i.e. it is already added to QUEUE.)



(a)

Front = 5

Rear = 6

1	2	3	4	5	6	7	8
A	F	C	B	D	G		

F

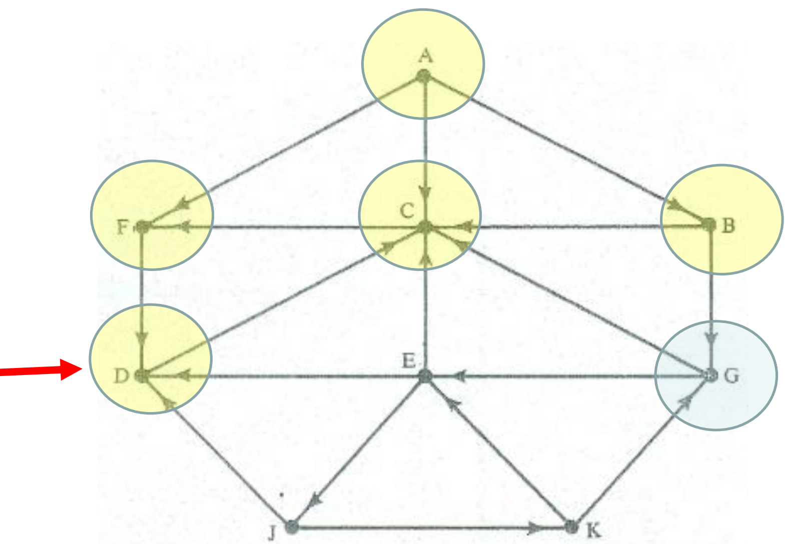
R

Example 8.7

f) Remove the front element D from QUEUE , & add to QUEUE the neighbors of D as follows.

FRONT = 6 QUEUE :A, F, C, B, D, G

REAR = 6 ORIG :ϕ, A, A, A, F, B



(a)

Front = 6

Rear = 6

1	2	3	4	5	6	7	8
A	F	C	B	D	G		

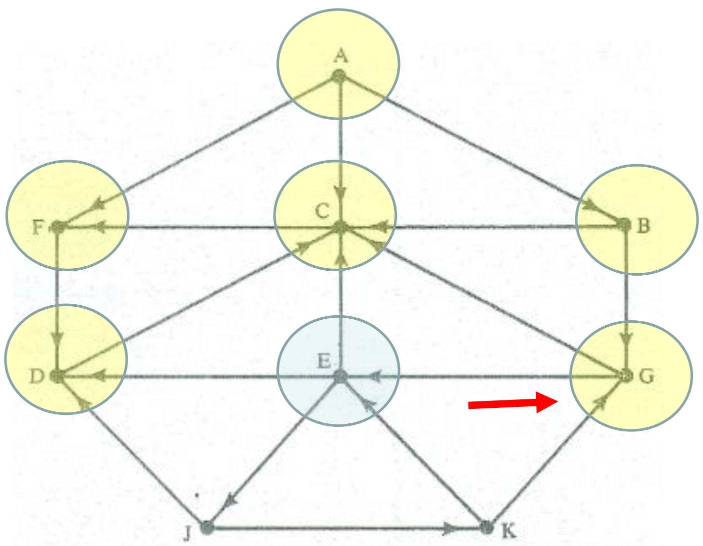
F R

Example 8.7

g) Remove the front element G from QUEUE , & add to QUEUE the neighbors of G as follows.

FRONT = 7 QUEUE : A, F, C, B, D, G, E

REAR = 7 ORIG : ϕ, A, A, A, F, B, G



(a)

Front = 7

Rear = 7

1	2	3	4	5	6	7	8
A	F	C	B	D	G	E	

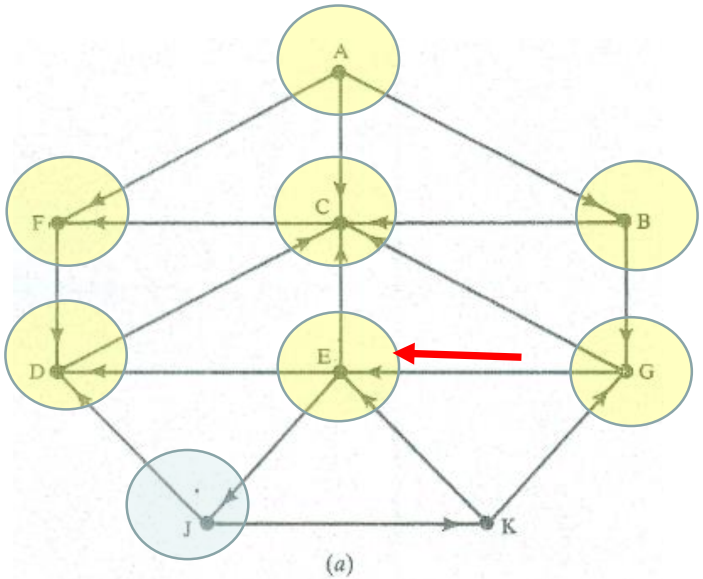
F R

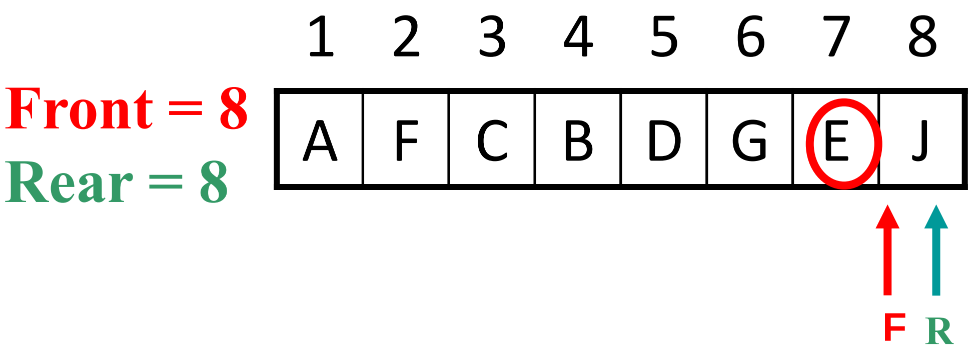
Example 8.7

h) Remove the front element E from QUEUE , & add to QUEUE the neighbors of E as follows.

FRONT = 8 QUEUE : A, F, C, B, D, G, E, J

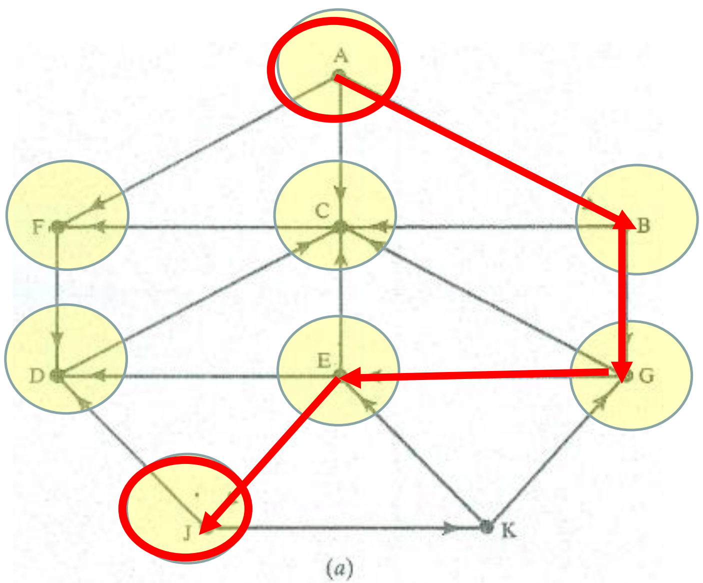
REAR = 8 ORIG : ϕ, A, A, A, F, B, G, E





Example 8.7

- We stop as soon as J is added to QUEUE, since J is our final destination.
- We now backtrack from J, using the array ORIG to final the path P.
- Thus We obtain **J ← E ← G ← B ← A** Is the required path P.



2) Depth – First Search

The idea behind a depth-first search beginning at a starting node A is as follows.

- First we examine the starting node A.
- Then we examine each node N along a path P which begins at A; that is, we process a neighbor of A, and so on.
- After coming to "Dead End" that is, to the end of the path p, we backtrack on P until we can continue along another, path P^1 and so on. (This algorithm is similarly to inorder traversal of a binary tree.)
- This algorithm is similar to BFS except here we use **stack** instead of the **queue**. Also, a field STATUS is used to tell us the current status of a node.

2) Depth – First Search

Algorithm DFS :- This algorithm executes a depth-first search on a graph G beginning at a starting node A .

1.Initialize all nodes to the ready state (STATUS =1).

2.Push the starting node A onto STACK & change its status to the waiting state (STATUS = 2).

3.Repeat steps 4 and 5 until STACK is empty :

4. Pop the top node N of STACK.

   Process N and change its status to the processed state (STATUS = 3).

5. PUSH onto STACK all neighbors of N that are still in the ready state (STATUS = 1), and change their status to the waiting state (STATUS = 2) .

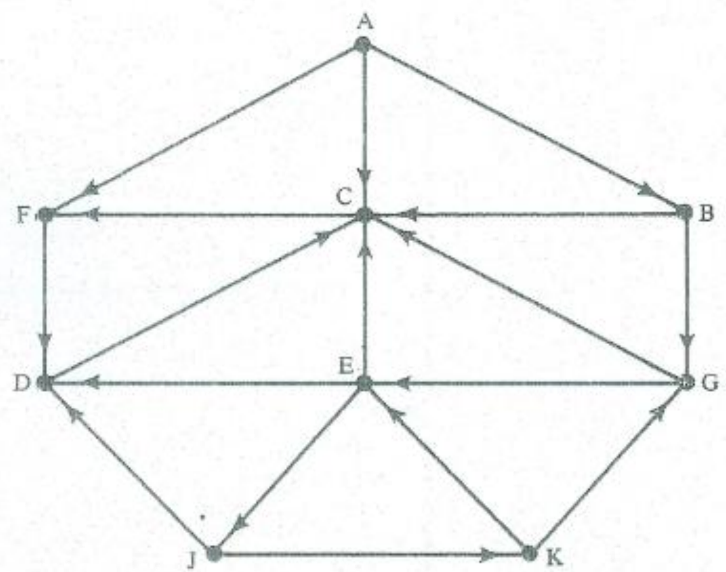
   [End of Step 3 Loop .]

6. Exit

Example 8.8

Consider the following graph G in the following figure. Suppose we want to **find and print** all the nodes reachable from the node J (Including J itself).

One way to do this is to use a depth-first search of G starting at the node J.



(a)

Adjacency Lists	
A:	F, C, B
B:	G, C
C:	F
D:	C
E:	D, C, J
F:	D
G:	C, E
J:	D, K
K:	E, G

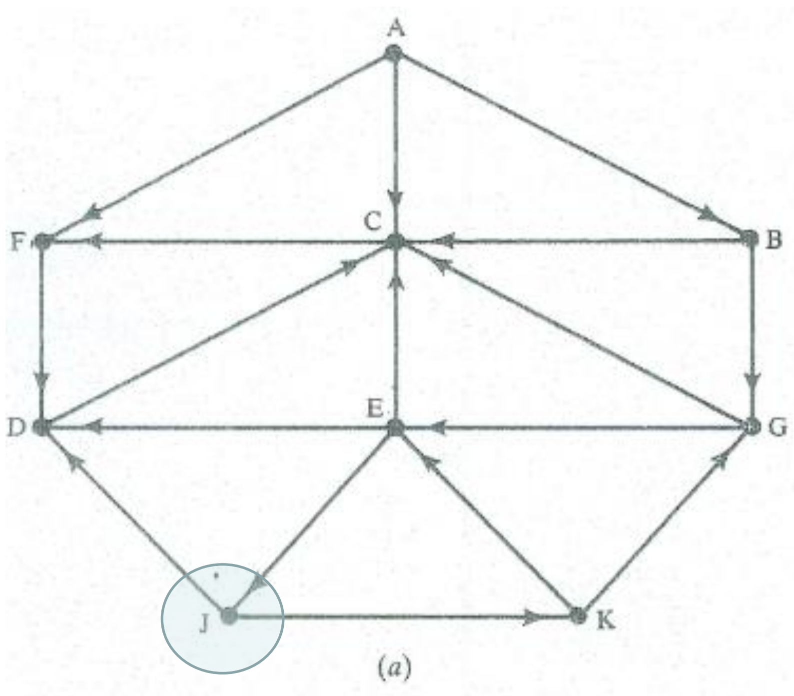
(b)

Fig. 8-14

The steps of DFS is as follows :

a) Initially, Push J onto the stack as follows :

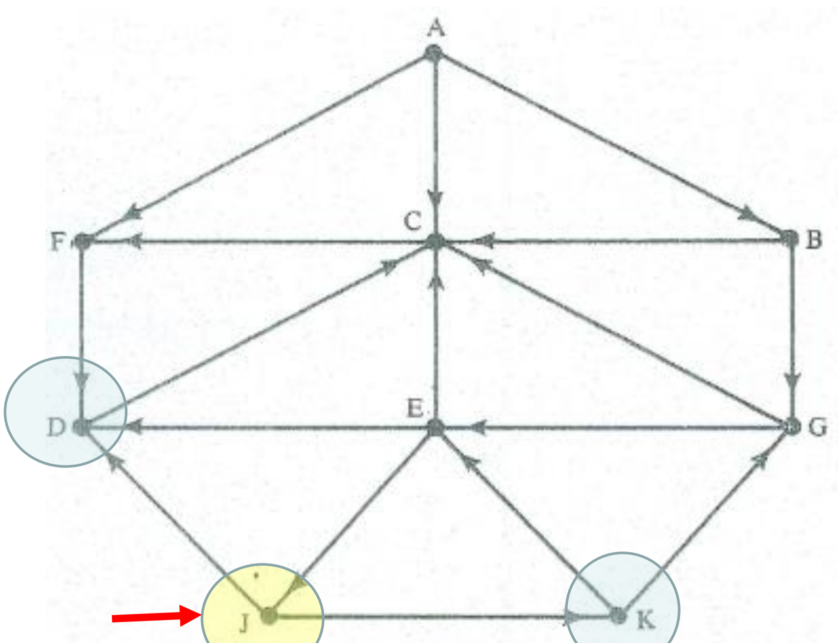
STACK : J



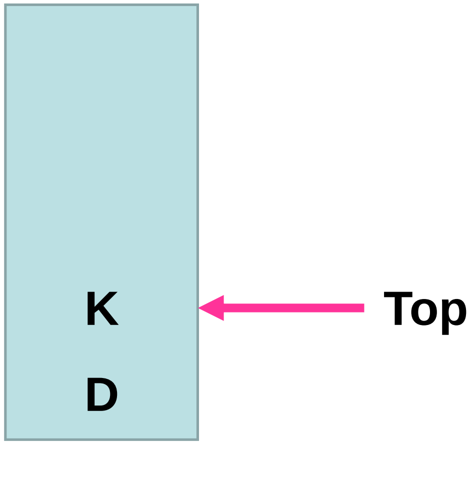
The steps of DFS is as follows :

b) Pop and print the top element J , & then Push onto stack all the neighbors of J (Those that are in the ready state) as follows:

Print J **STACK : D, K**



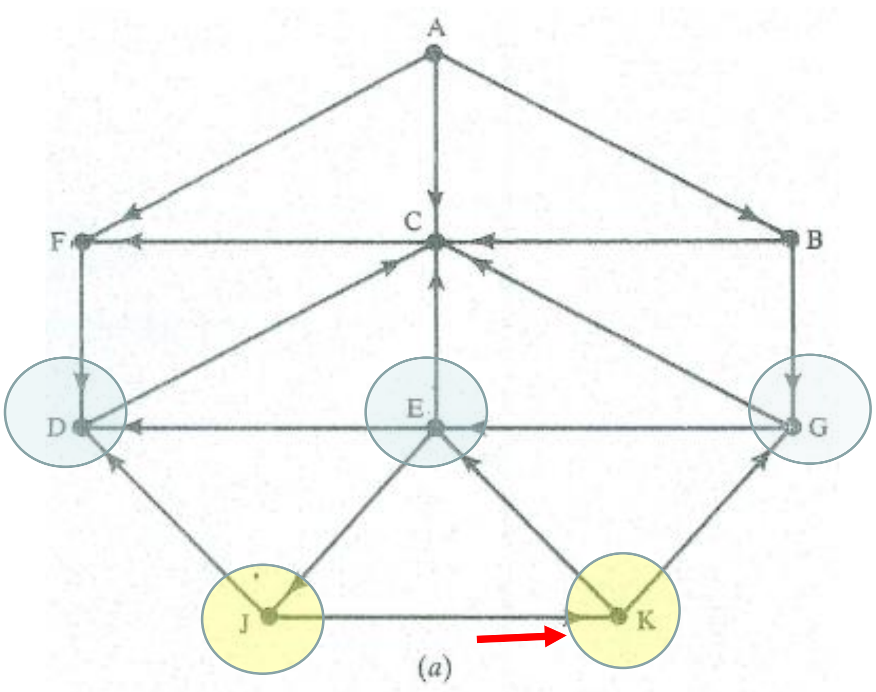
(a)

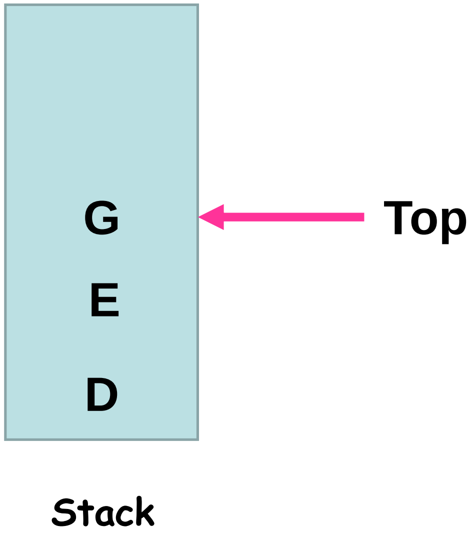


The steps of DFS is as follows :

c) Pop & Print the top element K , & Push onto the stack all the neighbors of K (Those that are in the ready state) as follows:

Print K   STACK : D , E , G

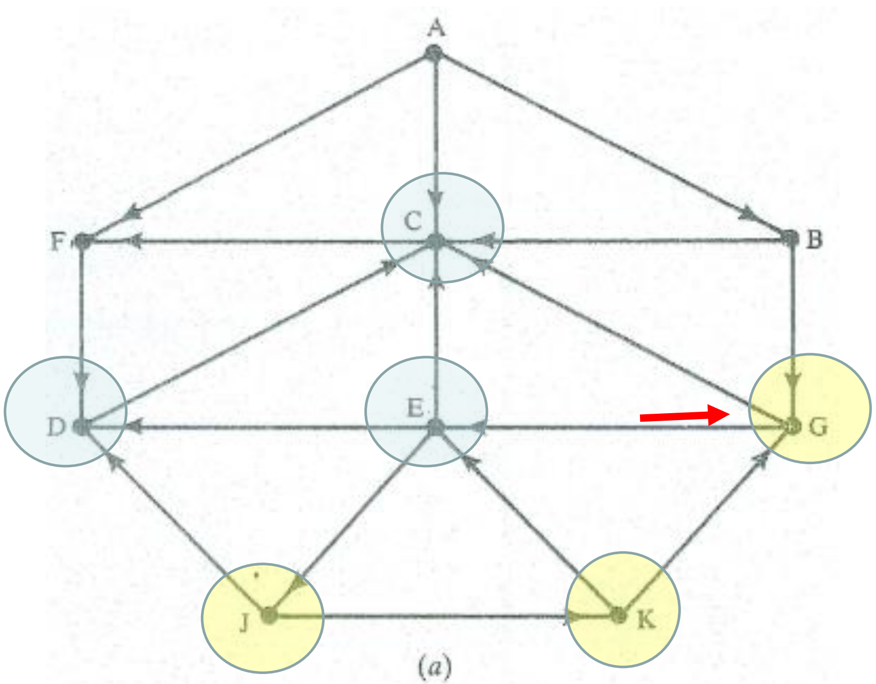


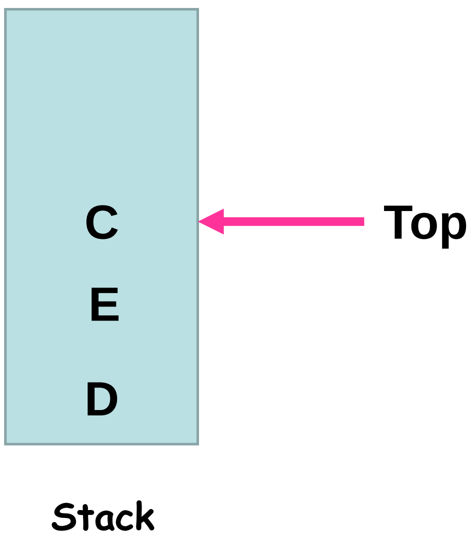


The steps of DFS is as follows :

d) Pop & print the top element G , & then push onto the stack all the neighbors of G (Those in the ready state)

Print G STACK : D , E , C



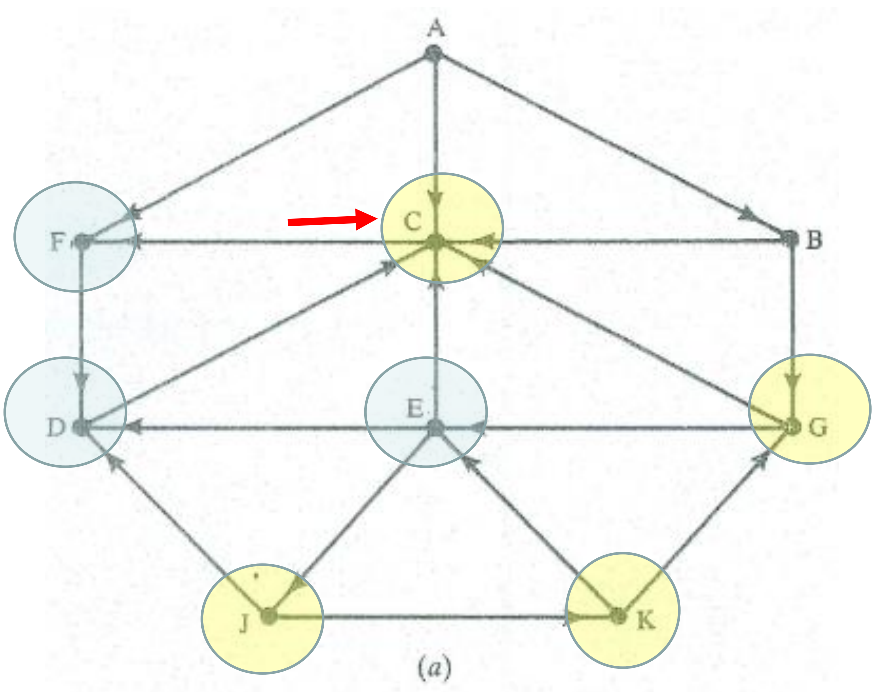


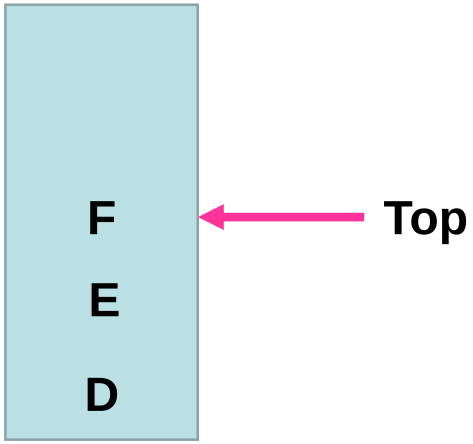
Note that only C is pushed onto the stack, since the other neighbour, E is not in the ready state (because E has already been pushed onto stack)

The steps of DFS is as follows :

e) Pop & print the top element C, & then push onto the stack all the neighbors of C (Those in ready state) as follows :

Print C STACK : D , E , F



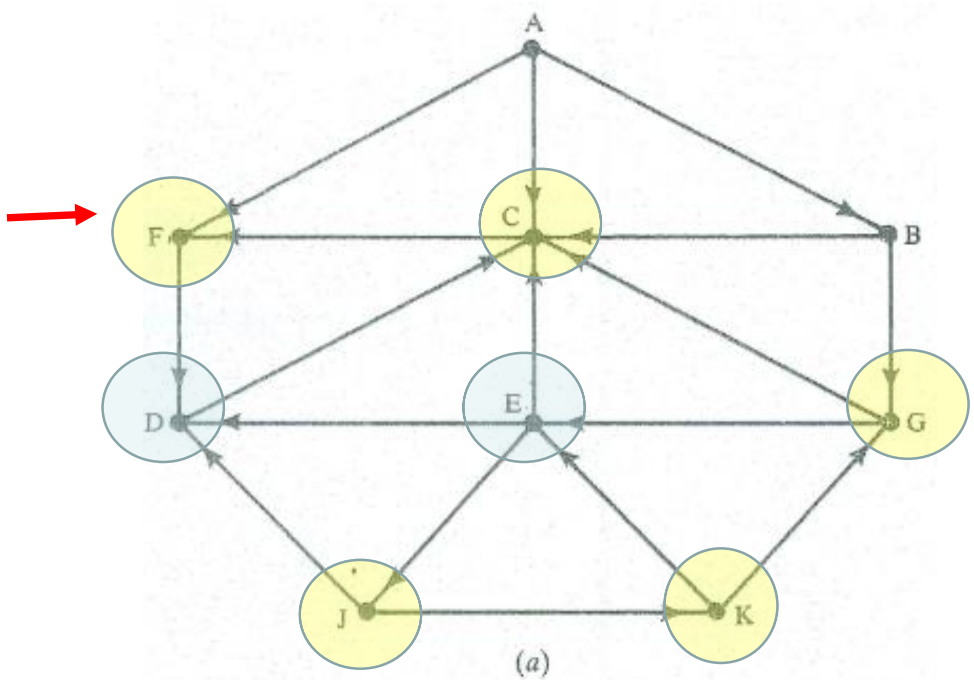


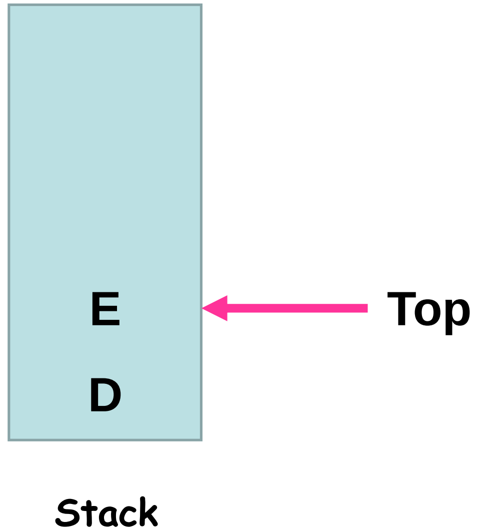
Stack

The steps of DFS is as follows :

f) Pop & print the top element F , & then Push onto stack all the neighbors of F(Those that are in the ready state) as follows :

Print F STACK : D, E



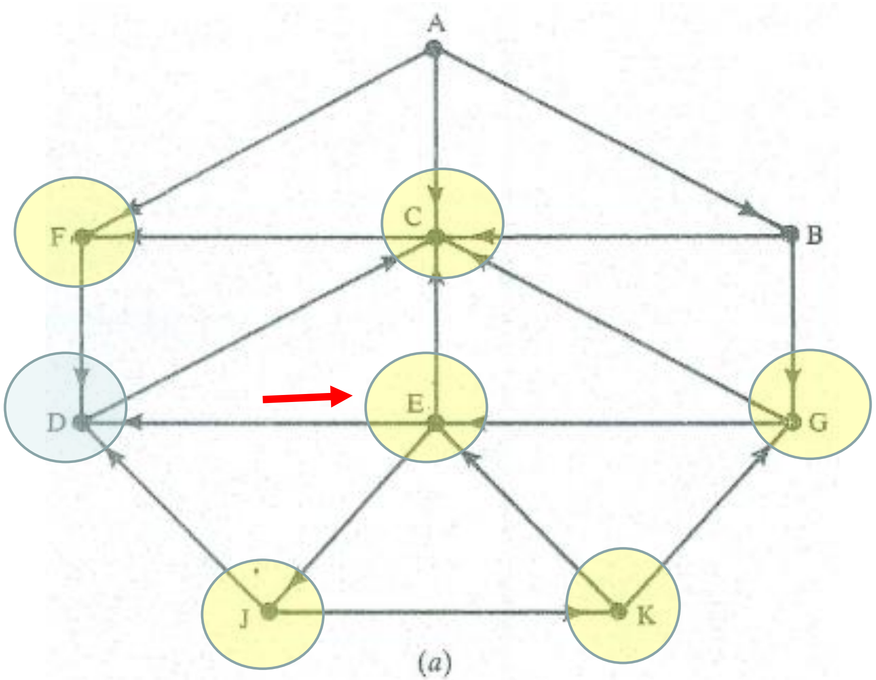


Note that only D of F is not pushed onto the stack, since D is not in the ready state (because D has already been pushed onto stack)

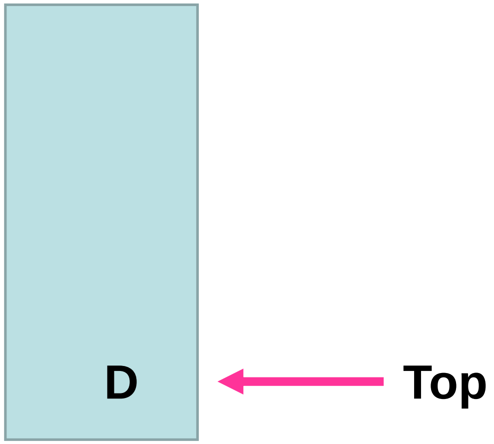
The steps of DFS is as follows :

g) Pop & Print the top element E , & Push onto the stack all the neighbors of E (Those in the ready state) as follows :

Print E STACK : D



(a)



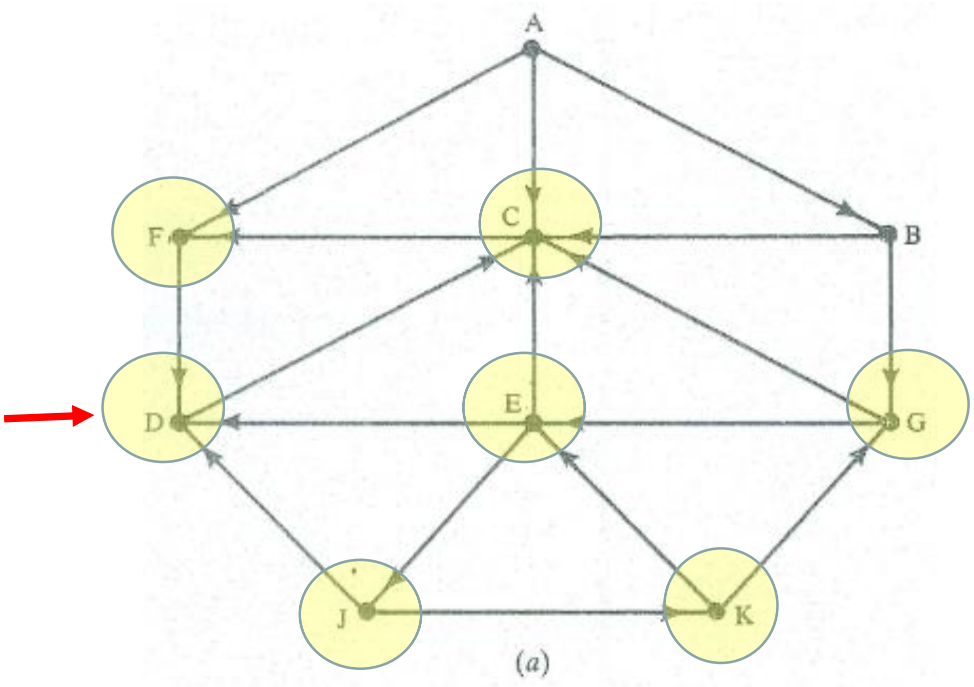
Stack

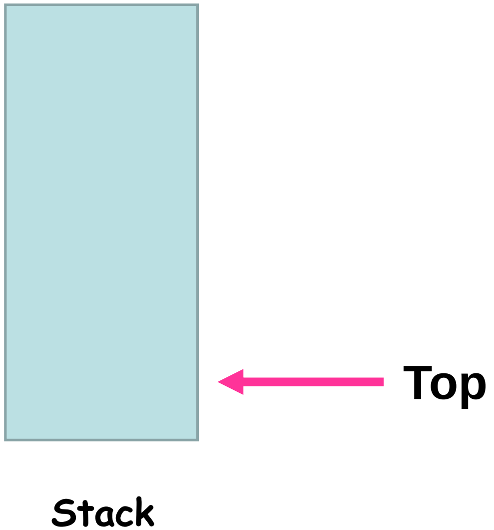
(Note that none of the three neighbours of E is in the ready state)

The steps of DFS is as follows :

h) Pop & print the top element D , & push onto the stack all the neighbors of D (Those in the ready state) as follows

Print D STACK :





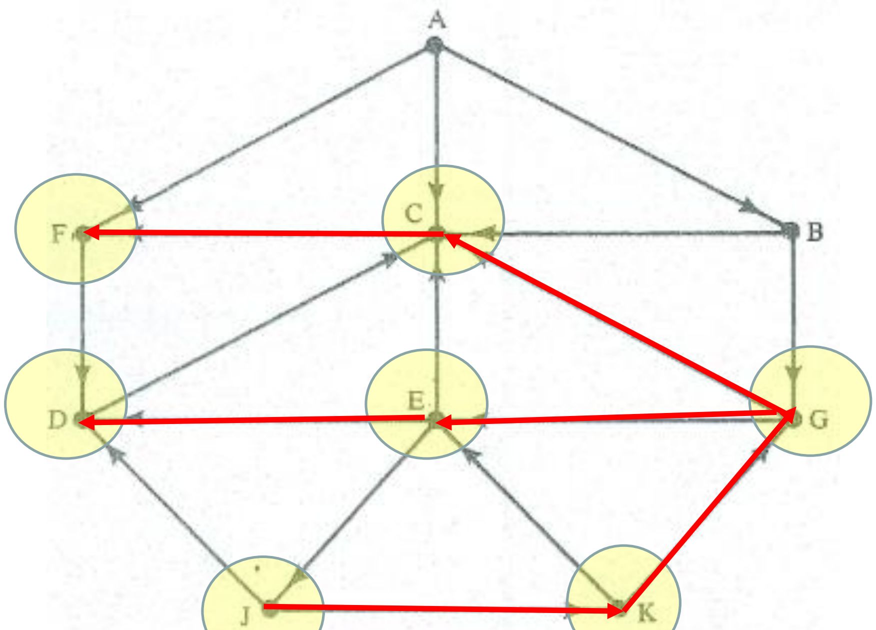
The steps of DFS is as follows :

The stack is now empty , so the depth-first search of G starting at J is now complete.

Accordingly , the nodes which were printed

J , K , G , C , F , E , D

Are the nodes which are reachable from J.



(a)

تم الإنتهاء من المحاضرة