Fall - 2022 (Data Structure)

5 a) # Write a case where using arrays provides advantages instead of linked lists.

Ans:

(I) Random Access: Arrays allow for direct access to elements using their index. Accessing an element in array takes O(1) time complexity where linked list require traversing from head node to reach specific element cause O(n)

(II) Memory Efficency: Array is more memory efficient it doesn't require extra memory for storing pointers to the next node.

(III) Cache Locality: accessing element in array is faster due to reduced cache misses.

(IV) Arrays are simpler. They have fixed size making it easy to allocate memory manage

(b) # Given a linked list and 2 integer m and k write Pseudo code with

(i) First insert a node with the item value m at the beginning.

(ii) then delete last k node from L.L

Algo;

Function insert delete (heads, m, k)

(i) create a new node with value m.

(ii) Set the next pointer of the new node to the current head of the L.L

(iii) Update the head with new node.

(iv) count = 0

(v) If the count is less than k continue to next node

(vi) If count is k, set the next pointer Previous node to none.

(vii) increament count

Simulation:

(1) initial — 1, 3, 5, 7

m = 0, k = 2

2) insert m = 0 at begin. : 0, 1, 3, 5, 7

3) Delete ~~last~~ last k node. : 0, 1, 3, ~~5, 7~~

6/ # Write cases where using Linked list provides advantages instead of array.

Ans:

(i) Dynamic size: L.L can easily grow or shrink during runtime. they do not require predefine size like array.

(ii) Efficent insertion Deletion: insertion and deletion require O(1) time complexity in L.L where array require O(n).

(iii) No waste memory: L.L only allocate memory for elements when needed. array often have unused allocated memory

4) Ease of implementation: When dealing with operations that require frequent resizing LL is better than Dynamic Array.

5) inserting, deleting at the beginning or end of a linked list can be done in O(1) time, which is more efficient, compared to array where this required shifting all the elements.

B) # 2 integers m and k. write code,

(1) First insert a node with the item value m at the end of the list,

(1) Then delete k nodes from L.L.

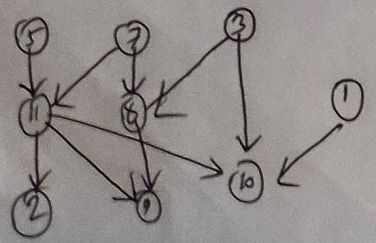
ii) Queue type of Data structure will be appropriate for implementing this traversal algorithm. Because BFS uses a Queue to maintain the order of exploration, When you visit a vertex you enqueue its adjacent unvisited vertices. The vertices are explored in the order they were enqueued, ensuring that closer vertices are visited, before farther ones. 3

① Queue follows first in first out Principle. The friends who were discovered first will be informed first.

8) The starts from a vertex and then wants to visit every vertex ~~from the~~ till it has no unvisited adjacent vertex. After that it goes back and explores other vertex from the same vertex. What is the most suitable graph traversing algorithm should be used.

(i) According to your answer show the traversing steps on the following ~~algorithm that~~ graph Fig 2 where you have the freedom to choose any vertex as the starting vertex

(ii) What type of Data structure (stack/Queue) will be appropriate for implementing this traversal algorithm



Ans: The most suitable graph traversing is Depth first search (DFS).

DFS explore as far as possible along each branch before backtracking. In this case, DFS would allow the person to go as deep as possible from the starting vertex before coming back and exploring other vertices from the starting

Part

Graph traversal steps

(1) Chose a starting vertex.

(2) exploring as far as possible along each branch before backtracking.

(3) mark visited vertices to avoid revisiting

ii) The suitable graph is DFS. So the type of Data structure will be used is stack.

(I) Stack is used to keep track of the vertices to be visited. also can use recursion implicitly or stack explicitly.

(II) When reach a vertex, push it to stack then explore as deep as possible recursively visiting adjacent unvisited vertices.

(III) When there are no more unvisited adjacent vertices, you back track by popping vertex from the stack to explore other branches.

Stack is appropriate because it allows for backtrack efficiently.

Elements are added to the stack when exploring new vertices and removed from stack during backtracking ensuring DFS explores path to the deepest

Fall-22 – Data Structure

1) a) The worst time complexity to delete an array element is O(n), justify this statement.

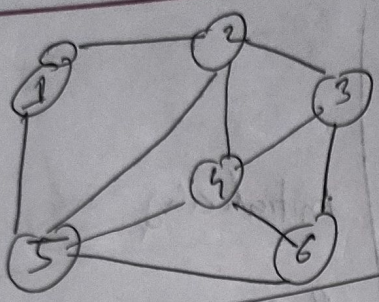
Sol^n The statement is justified because in the worst case scenario when you want to delete an element from an array you might have to shift all the subsequent element to fill the gap left by deleted element. The operation takes linear time proportional to the number of elements in the array (n) leading time complexity O(n)

(4a) Why recursion important in Data structure, write some example of it;

o/ Recursion is important for solving problems with recursive structure. it follows the divide and conquer strategy, recursive code is often more readable and decent.

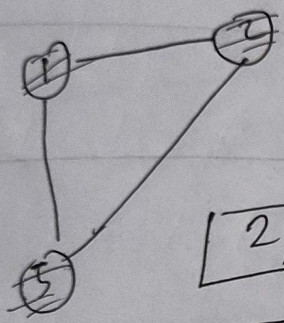
```
int factorial (int n)
    int n = 5;
    if (n == 1)
        return 1
    else
        return n * fact (n-1)
```

```
fibo (int a)
int fib = fibo(n-1) + fibo(n-2)
else if (n == 2)
    fib = 1
else fib = 0
return fib;
```



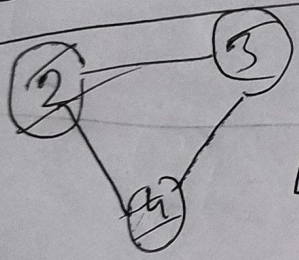
Initializing.

Result = 1



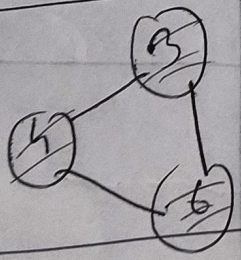
Result = 1, 2, 5

2	5



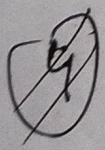
Result 1, 2, 5, 3, 4

3	4



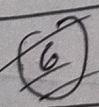
Result : 1, 2, 5, 3, 4, 6

4	6



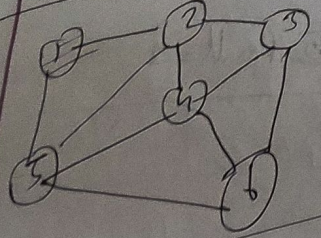
6

Result : 1, 2, 5, 3, 4, 6



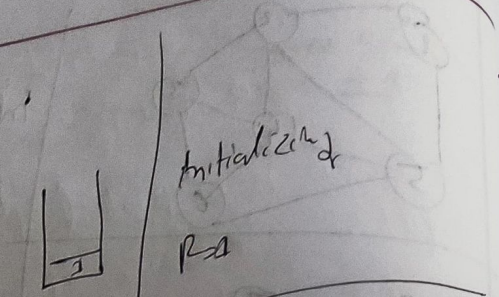
Result : 1, 2, 5, 3, 4, 6

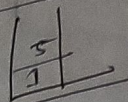
DFS

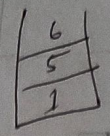


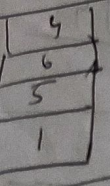
Initialization

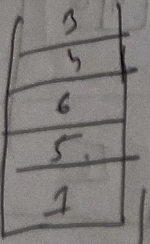
R={}



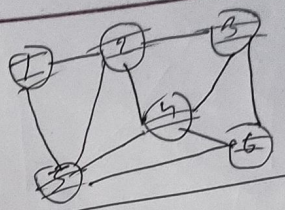


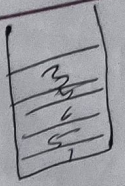


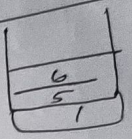




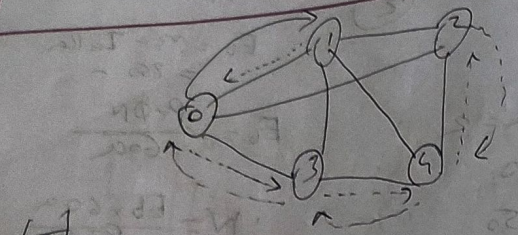
R; 1, 5, 6, 4, 3, 2

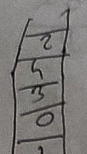




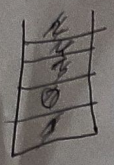


DFS



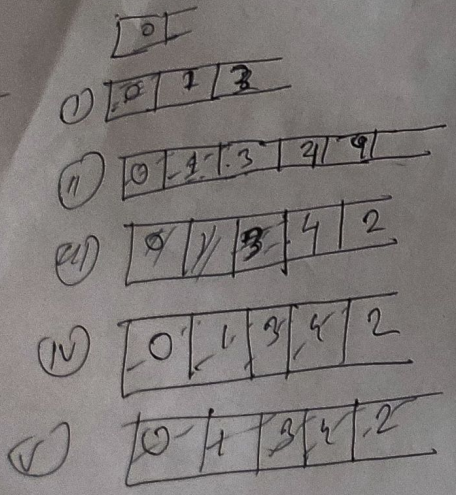


result: 1, 0, 3, 4, 2



result : 1, 0, 3, 4, 2

BFS



R = 0

R = 0, 1, 3

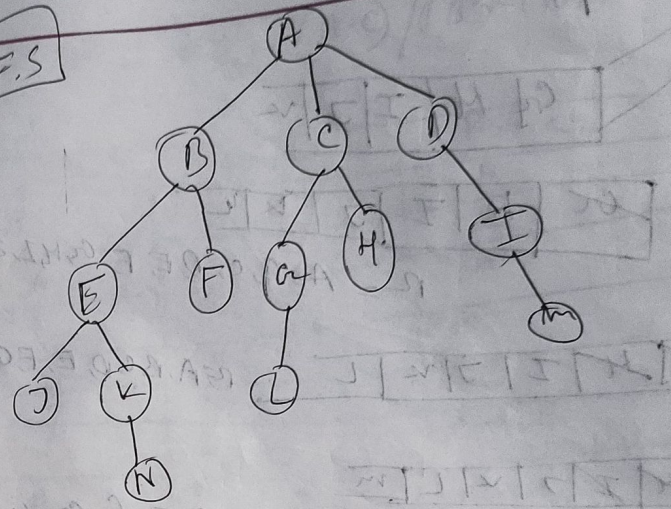
R = 0, 1, 3, 4

R = 0, 1, 3, 4, 2

R = 0, 1, 3, 4, 2

R = 0, 1, 3, 4, 2

BFS



(I) A

R = A

(II) A B C D

R = A, B, C, D

(III) A B C D E F

R = A, B, C, D, E, F

(IV) A B C D E F G H

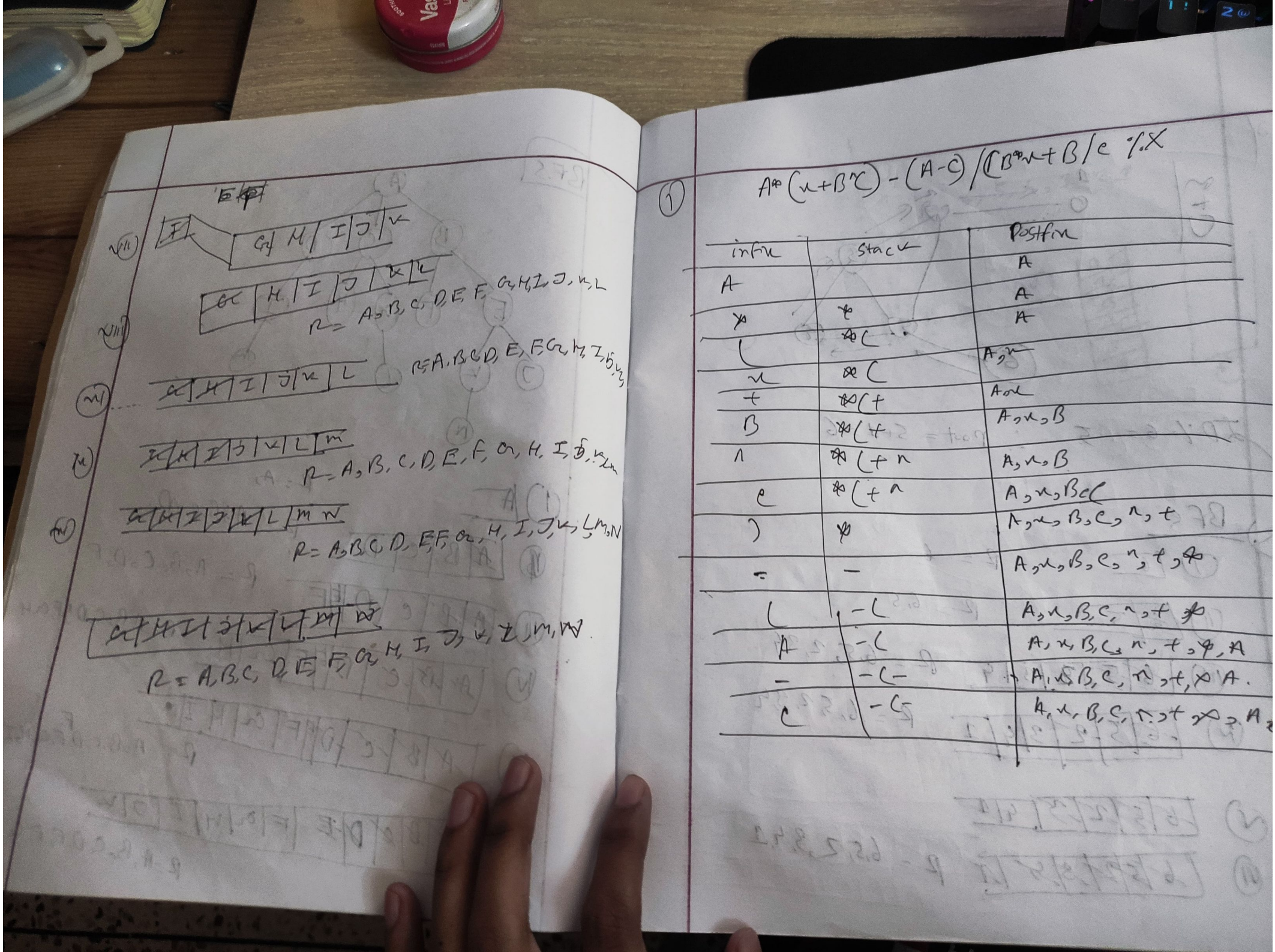
R = A, B, C, D, E, F, G, H

(V) A B C D E F G H I

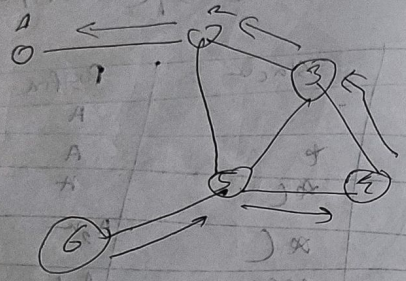
R = A, B, C, D, E, F, G, H, I

(VI) A B C D E F G H I J K

R = A, B, C, D, E, F, G, H, I, J, K



Q7



ID % 6 = 5 ∴ root = 5+1 = 6

BFS

(I) 6 R = 6

(II) 6 5 R = 6, 5

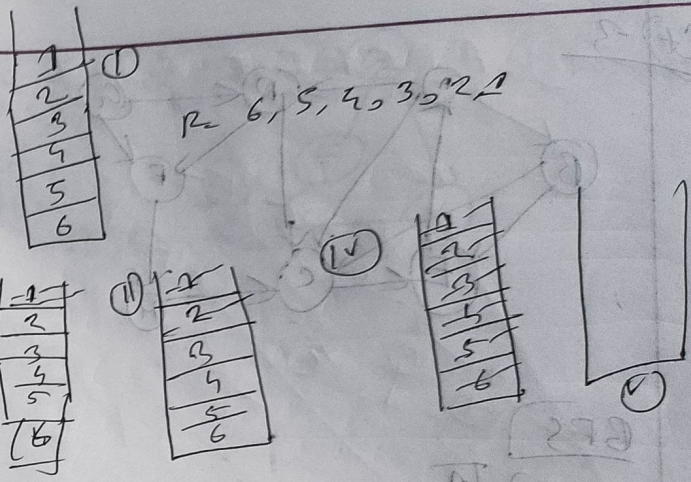
(III) 6 5 2 3 4 R = 6, 5, 2, 3, 4

(IV) 6 5 2 3 4 1 R = 6, 5, 2, 3, 4

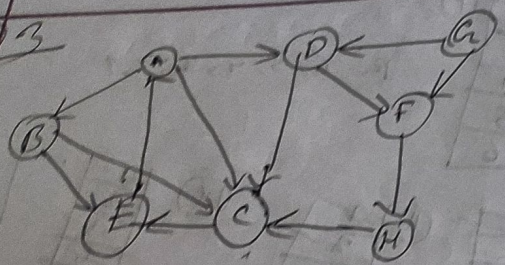
(V) 6 5 2 3 4 1

(VI) 6 5 2 3 4 1 R = 6, 5, 2, 3, 4, 1

DFS



CH 3



BFS

(I) A

(II) A | B | D | C

(III) A | B | D | E

(IV) A | B | D | E | C | F

(V) A | B | D | E | C | F | H

(VI) A | B | D | E | C | F | H | G

(VII)

(I) A | B | D | C | E

(II) A | B | D | C | E | F

(III) A | B | D | C | E | F | H

(IV) A | B | C | D | E | F

(V) A | B | C | D | E | F | G | H

DFS