Design and Analysis of Algorithms

Assignment - 3

Sharad Gumber
CST (Core)
Roll no :- 12

Q-1 What is difference between DFS and BFS. Please write the applications of both the algorithms.

Ans

DFS	BFS
(a) DFS stands for Depth First Search.	(a) BFS stands for Breadth First Search.
(b) DFS uses stack data structure.	(b) BFS uses Queue data structure for finding the shortest path.
(c) In DFS, we might traverse through more edges to reach a destination node from a source.	(c) BFS can be used to find single source shortest path in an unweighted graph, because in BFS we reach a node with minimum number of edges from a source node.
(d) In DFS we use a recursive algorithm that uses the idea of backtracking	(d) In BFS there is no concept of backtracking

Sharad

(e) DFS is more suitable when there are solutions away from source	(e) BFS is more suitable for searching vertices which are closer to the given source.
(f) DFS requires less memory.	(f) BFS requires more memory.

* Applications of BFS :- (i) GPS Navigation Systems.
(ii) Broadcasting in Network
(iii) Peer to Peer Networks.

* Applications of DFS :- (i) Topological sorting
(ii) Solving puzzles with only one solution
(iii) Detecting cycle in a graph.

Sharad

Q.2 Which Data Structures are used to implement BFS and DFS and why ?

Ans BFS does the search for nodes level-by-level, i.e. it searches the nodes with respect to their distance from the root. For this, we can see that BFS requires to visit the child nodes in order their parents were discovered. Whenever we visit a node, we insert all the nodes into our data structure. If we use a queue data structure, it is guaranteed that, we pop the nodes in order their parents were discovered.

DFS is depth first search, so we have to traverse a whole branch of tree then you can traverse the adjacent nodes. So for keep tracking on the current node it requires last in first out approach which was implemented by stack, after it reaches the depth of a node then all the nodes will be popped out of stack. Next it searches for adjacent nodes which are not visited yet.

Sharad

Q.3 What do you mean by sparse and dense graphs? which representation of graph is better for sparse and dense graphs?

Ans Sparse graph :- Sparse graph is a graph in which the number of edges is close to the minimal number of edges. Sparse graph can be disconnected graph. As the name indicates sparse graph are sparsely connected.

Dense graph :- A dense graph is a graph in which the number of edges is close to the maximal number of edges, or in other words if every pair of vertices is connected by one edge.

- If graph is sparse, we should store it as a list of edges.
- If the graph is dense, we should store it as an adjacency matrix.

Q.4 How can you detect a cycle in a graph using BFS and DFS?

Ans By following the below steps, we can find a cycle in BFS:-

(1) Compute number of incoming edges for each vertex present in the graph and initialize the count of visited nodes as 0.

(2) Pick all the the vertices with in-degree as 0 and add them into a queue.

(3) Perform Dequeue, then:-
- Increment count of visited nodes by 1.
- Decrease in degree by 1.
- If in-degree of a neighbouring nodes is reduced to zero enqueue.

(4) Repeat step 3 until the queue is empty.

(5) If count of visited nodes is not equal to the number of nodes in the graph.

Shared

Q.5 what do you mean by disjoint set data structure? Explain 3 operations along with examples, which can be performed on disjoint sets.

Ans The disjoint set data structure is also known as union-find data structure and merge-find set. It is a data structure that contains a collection of disjoint or non-overlapping sets. The disjoint set means that when the set is partitioned into the disjoint subsets.

Three operations performed on disjoint sets are:-

(1) Making new sets:-

```
function Makeset (x) is
    if x is not already in the
        x.parent = x
        x.size = 1
        x.rank = 0
    end if
end function
```

(2) Finding set representation :-

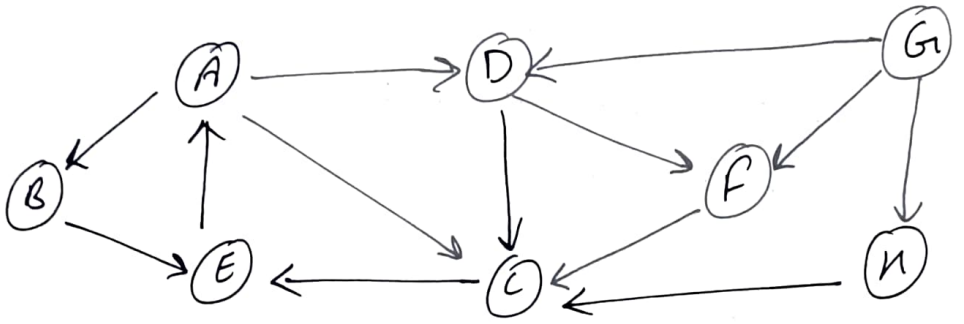
```
function find (x) is
    if x.parent ≠ x then
        x.parent = find (x.parent)
        return x.parent
    else
        return x
    end if
end function
```

(3) Merge two sets :-

```
function union (x, y) is
    x = find (x)
    y = find (y)
    if x = y then
        return
    end if
    if x.size < y.size then
        (x, y) = (y, x)
    end if
    y.parent = x
    x.size = x.size + y.size
end function
```

Sharad

Q-6 Run BFS and DFS on graph shown on right side.



Sol

BFS :-

Node	B	E	C	A	D	F
Parent		B	B	E	A	D

Unvisited nodes - G and H

Path = B → E → A → D → F

DFS :-

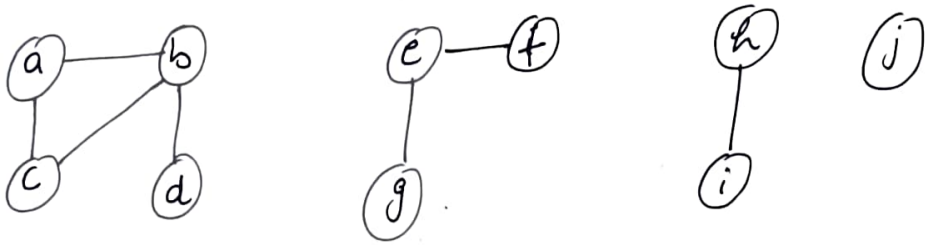
node processed :- B B C E A D F

Stack B CE EE AA DE FE E

Path :- B → C → E → A → D → F

Q-7 Find out the number of connected components and vertices in each component using disjoint set data structure.

Ans



$V = \{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g\}, \{h\}, \{i\}, \{j\}$

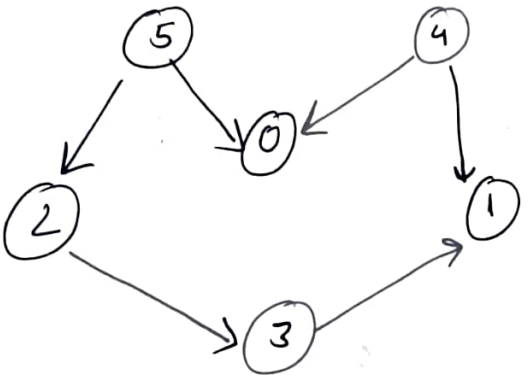
$E = \{a,b\}, \{a,c\}, \{c,b\}, \{b,d\}, \{e,f\}, \{e,g\}, \{h,i\}, \{j\}$

(a,b)	$\{a,b\}$ $\{c\}$ $\{d\}$ $\{e\}$ $\{f\}$ $\{g\}$ $\{h\}$ $\{i\}$ $\{j\}$
(a,c)	$\{a,b,c\}$ $\{d\}$ $\{e\}$ $\{f\}$ $\{g\}$ $\{h\}$ $\{i\}$ $\{j\}$
(b,c)	$\{a,b,c\}$ $\{d\}$ $\{e\}$ $\{f\}$ $\{g\}$ $\{h\}$ $\{i\}$ $\{j\}$
(b,d)	$\{a,b,c,d\}$ $\{e\}$ $\{f\}$ $\{g\}$ $\{h\}$ $\{i\}$ $\{j\}$
(e,f)	$\{a,b,c,d\}$ $\{e,f\}$ $\{g\}$ $\{h\}$ $\{i\}$ $\{j\}$
(e,g)	$\{a,b,c,d\}$ $\{e,f,g\}$ $\{h\}$ $\{i\}$ $\{j\}$
(h,i)	$\{a,b,c,d\}$ $\{e,f,g\}$ $\{h,i\}$ $\{j\}$

No. of connected components = 3.

Shared

Q.8 Apply topological sorting and DFS on graph ~~with~~ having vertices from 0 to 5.



Ans Adjacent List 0 →
1 →
2 → 3
3 → 1
4 → 0, 1
5 → 2, 0

Visited :-

false	false	false	false	false	false
0	1	2	3	4	5

Stack (empty).

Step 1 :- Topological Sort (0) visited [0] = true
list is empty no more recursion call

stack | 0 |

Step 2 :- Topological Sort (1), visited [1] = true
list is empty no more recursion call

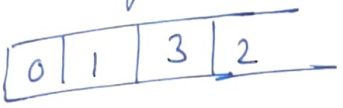
stack | 0 | 1 |

Step 3 :- Topological sort (2), visited [2] = true

↓

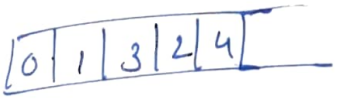
Topological Sort (3), visited [3] = true

'1' is already visited, no more recursion call stack



Step 4 :- Topological sort (4), visited [4] = true

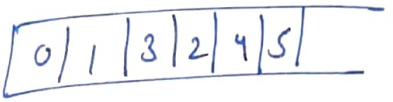
'0', '1' are already visited no more recursion call stack



Step 5 :- Topological sort (5), visited [5] = true

↓

'2', '0' are already visited. no more recursion call stack



Step 6 :- Print all the elements of stak from top to bottom.

5, 4, 2, 3, 1, 0

Sharad

Q.9 Heap data structure can be used to implement priority queue. Name few graph algorithms where you need to use priority queue and why?

Ans Yes, we can use heaps to implement the priority queue. It will take O(log N) time to insert and delete each element in the priority queue. Heaps are great for implementing a priority queue because of the largest and smallest element at the root of the tree for a max-heap and a min-heap respectively. We use a max-heap for a max-priority queue and a min-heap for a min-priority queue.

Few graph algorithms :-

(a) Dijkstra's :- when the graph is stored in the form of adjacency list or matrix, priority queue can be used to extract minimum efficiently when implementing Dijkstra's algorithm.

(b) Prism's algorithm :- It is used to implement Prism's algorithm to store keys of nodes and extract minimum key node at every step

(c) Heap sort :- Heap sort is typically implemented using Heap which is an implementation of Priority Queue.

Anand

Q-10 What is difference between Max and Min heap?

Ans

Max- heap	Min- heap
(a) In a Max-heap the key present at the root node must be greater than or equal to among the keys present at all of its children.	(a) In a Min-heap the key present at the root node must be less than or equal to among the keys present at all of its children.
(b) In a Max-heap the maximum key element present at the root.	(b) In a Min-heap the minimum key element present at the root.
(c) A Max-heap uses the descending priority.	(c) A min-heap uses the ascending priority.
(d) In the construction of a Max-heap, the largest element has priority.	(d) In the construction of a Min-heap, the smallest element has priority.
(e) In a Max-heap, the largest element is the first to be popped from the heap.	(e) In a Min-heap, the smallest element is the first to be popped from the heap.

Shared