Name - Akanksha Dubey
Section - C
Roll No. - 42
University Roll No. - 2016600

Tutorial - 5

1. What is difference between DFS and BFS. Write an application of both the algorithm.

BFS	DFS
BFS stands for breadth first search.	DFS stands for Depth first search
• BFS uses Queue to find the shortest path.	• DFS uses stack to find the shortest path.
• BFS is better when target is closer to source	DFS is better when target is far from sources.
• As BFS considers all neighbour so it is not suitable for decision tree used in puzzel games.	• DFS is more suitable for decision tree. As with one decision, we need to traverse further to argument the decision. If we reach the conclusion we won.
• BFS is slower than DFS	DFS is faster than BFS
Time complexity = O(V+E) V = Vertices E = Edge.	Time complexity = O(V+E)
Application. • Used in GPS navigations to find neighboring places. • Use to find the path. • Use to find all the neighbour nodes.	Application • Use to Detect cycle in a graph. • Use to find the Path from source to destination. • Used to perform topological Sorting

2. Which Data structure is used to implement BFS and DFS

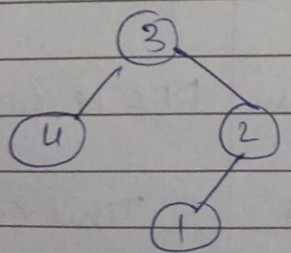
Data structures used in case of BFS and DFS are queue and stack respectively

BFS requires queue data structure because it traverse a graph in breadth word motion & uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

→ DFS requires stack data structure because it traverse a graph in depth word motion & uses a stack to remember to get to the next vertex to start a search, when a dead end occurs in iteration.

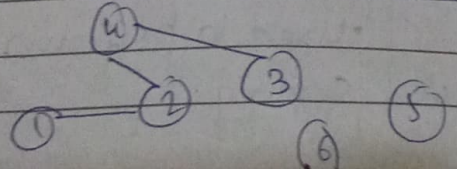
3. Dense Graph

is a graph in which no. of edge is close to the maximal number of edges.



Sparse Graph

is a graph in which number of edges is close to minimal number of edges.



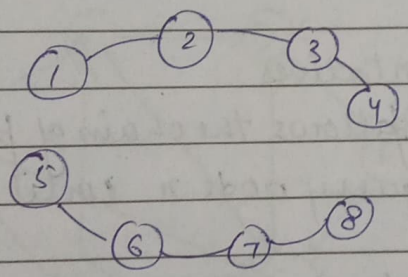
It is ideal to use sparse graphs by adjancency list and dense graph by an adjacency matrix.

4. ~~Detect cycle using BFS~~

5. Disjoint set

- 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 disjoint subsets.

Ex:-

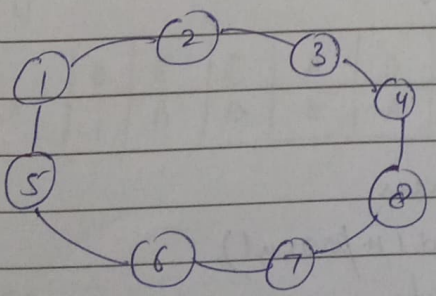


$S1 = \{1, 2, 3, 4\}$

$S2 = \{5, 6, 7, 8\}$

Union of these two sets are.

$S1 \cup S2 = \{1, 2, 3, 4, 5, 6, 7, 8\} = S3$



Disjoint set supports three operations.

① Making new sets.

The mask set operation adds a new element into a new set containing only the new element, and the new set is added to the data structure.

If the data structure is instead of viewed as a partition of a set, then the MakeSet operation enlarges the set by adding the new element and it extends the existing partition by putting the new element into a new subset containing only the new element.

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

Finding Set Representatives

The find operation follows the chain of parent pointers from a specified query node x until it reaches a root element.

This root element represents the set to which x belong and may be x itself. Find returns the root element it reaches.

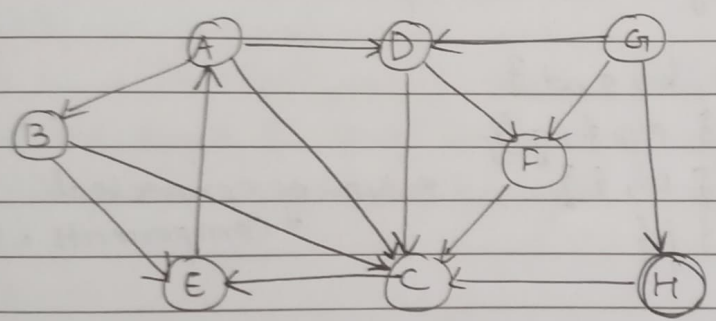
```
function find(x) is
  if x.parent ≠ x then
    x.parent = find(x.parent)
    return x.parent
  else
    return x
  end if
end function.
```

6.

Merge two sets

The operation union (x, y) replace the set containing x and the set containing y with their union.

Ex: function Union (x, y) is
    x = find (x)
    y = find (y)
    if x = y then
        return
    end if
    y.parent = x
    x.size = x.size + y.size.
end function.



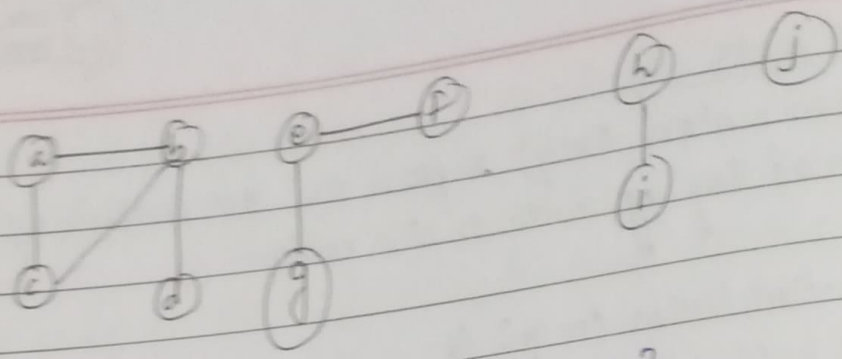
using BFS

Node	B	E	C	A	D	F
Parent	-1	B	B	E	A	D

DFS

B	E	A	D	F	C

7.



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

How make find and union operation will be performed.

→ First each vertex will be parent of itself.

→ If an edge exists between vertices, then we perform find operation & check if their parents same & perform union operation on them.

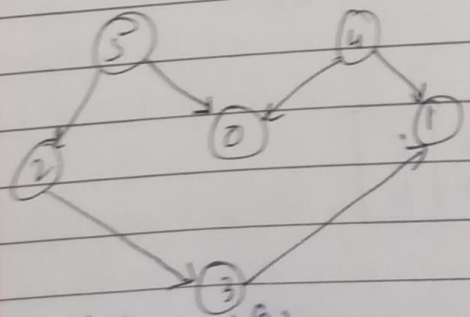
$S_1 = \{a, b, c, d\}$

$S_2 = \{e, f, g\}$

$S_3 = \{h, i\}$ * No. of connected components = 3

$S_4 = \{j\}$

(8)



topo sort(0)

As we cannot visit any other node push 0

topo sort(1)

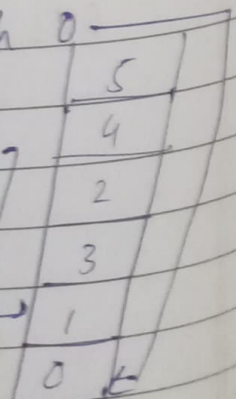
we cannot visit any other node push 1

toposort(2) toposort(3)

first push 3 then 2

topo Sort(4)

push 4 to stack



toposort(5)
push 5 to stack as other are visited
∴ 5 4 2 3 1 0

(a) We can use heaps to implement the priority queue. It will take O(log N) time to insert & delete each element in priority queue.

Based on heap structure, priority queue has also two types Max Priority & Min-Priority

Some algorithms we need to use Priority queue

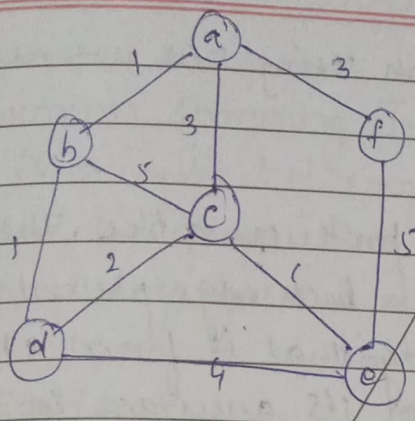
(i) Dijkstra's
It is used in calculating shortest path in the algorithms when the graph is stored in form of adjacency list a matrix priority queue can be used to extract efficient.

(ii) Prisms
It is used to implement Prism algo to store key of nodes & extact minimum key node at every step.

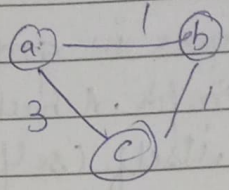
(iii) Data Compression
It is used in Huffman coding which is used to compress data.

10	Min Heap	Max Heap
	In a min heap the key present at root must be less than or equal to among the key present at all of its children.	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.
•	The minimum key element is present at the root.	The maximum key element is present at root.
•	Uses ascending priority.	• Uses descending priority.
•	Here the smallest element has a priority.	• Here the largest element has a priority.
	• The smallest element is to be popped from heap.	• The largest element is to be popped from heap.

(4)

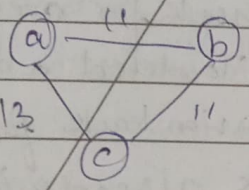


If we add the weight of the graph by 10, YES the shortest path can change then, consider.



Shortest path
a → b → c

If we add 10:



Shortest path
a → c

There is no change in the

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

For Detecting cycle in a graph using BFS we need to use Kahn's algorithm for Topological Sorting

1) Compute in-degree (no. of incoming edge) for each of vertex present in graph & initialize count of visited node as 0.
2) Pick all vertices with in-degree as 0 and add them in a queue.
3) Remove a vertex from queue and then
   - increment count of visited node by 1.
   - Decrease in-degree by 1 for all the neighbouring
   - If in-degree of neighbouring nodes is reduced to 0 then add queue
4) Repeat (3) until queue is empty.
5) If count of visited node is not equal to no. of nodes in graph has cycle otherwise not.

For Detecting cycle in graph using DFS we need to do following:

DFS for a connected graph produces a tree. There is cycle in graphs if there is a back edge present in the graph. A back edge is an edge that is from a node to itself (self-loop) or one of its ancestors in the tree produced by DFS. For a disconnected graph get DFS forest as output. To detect cycle check for a cycle in individual trees by checking back edges. To detect a back edge, keep track of vertices currently in recursion track for DFS traversal. If a vertex is reached that is already in recursion stack then there is a cycle.