Sakshi Rajvanshi.
01
CST SPL-2.

Tutorial - 05

Ques1. what is the difference between DFS and BFS. write the applications of both the algorithms.

Ans.

DFS	BFS
1. DFS stands for Depth first search.	1. BFS stands for Breadth first search.
2. It uses stack data structure.	2. It uses queue data structure for finding the shortest path.
3. In DFS, we might traverse through more edges to reach a destination vertex from a source.	3. We reach a vertex with minimum no. of edges from a source vertex.
4. DFS is more suitable when there are solutions away from source.	4. BFS is more suitable for searching vertices which are ~~given~~ closer to the given source.
5. Here, children are visited before the siblings.	5. Here, siblings are visited before the children.
6. DFS algorithm is a recursive algorithm that uses the idea of backtracking.	6. In BFS there is no concept of backtracking.
7. DFS requires less memory.	7. BFS requires more memory.

Applications of DFS:

DFS is used in various applications such as acyclic graph and topological order etc.

Applications of BFS:

BFS is used in various application such as bipartite graph, and shortest path etc.

Ques 2. Which Data structures are used to implement BFS and DFS and why?

Ans. DFS algorithm traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any ~~it~~ iteration.

BFS Algorithm uses a queue and a graph. The algorithm makes sure that every node is visited not more than once. It explores every node once and put that node in queue and then it takes out nodes from the queue and explores its neighbours.

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

Ans. In dense graph, every pair of vertices is connected by one edge. The sparse graph is completely opposite. If a graph has only a few edges (the number of edges is close to the maximum number of edges), then it is a sparse graph.

There is no strict distinction between the sparse and the dense graphs.

If the graph is sparse, we should store it as a list of edges.

Alternatively, if the graph is dense, we should store it as an adjacency matrix.

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

Ans. Steps involved in detecting cycle in a directed graph using BFS.

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

Step-2: Pick all the vertices with in-degree as 0 and add them into a queue (Enqueue operation)

Step-3: Remove a vertex from the queue (Dequeue operation) and then

1. Increment count of visited nodes by 1.
2. Decrease in-degree by 1 for all its neighbouring nodes.
3. If in-degree of a neighbouring node is reduced to zero, then add it to the queue.

Step 4: Repeat step 3 until the queue is empty.

Step 5: If count of visited nodes is not equal to the number of nodes in the graph has cycle, otherwise not.

using DFS

DFS can be used to detect a cycle in a Graph. DFS for a connected graph produces a tree. There is a cycle in a graph only if there is a back edge present in a graph.

For a disconnected graph, get the 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 the recursion stack of function for DFS traversal. If a vertex is reached that is already in the recursion stack, then there is cycle in the tree.

Ques 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.

Disjoint - set data structures support three operations :

1) Making a new set containing a new element

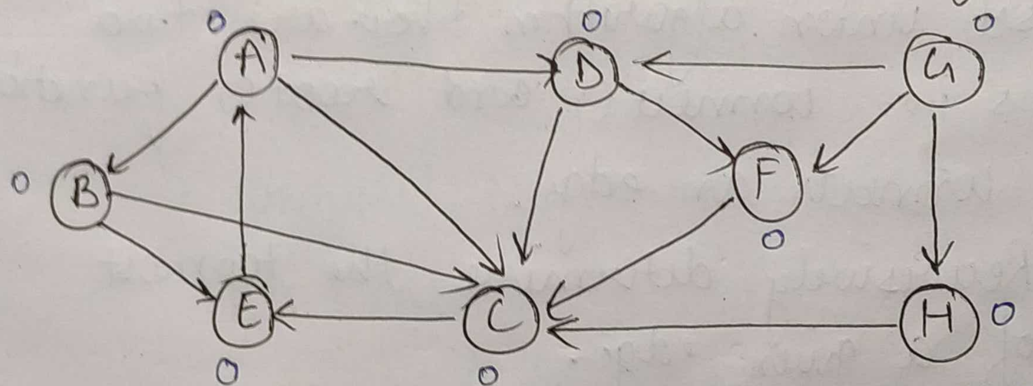
2) finding the representative of the set containing a ~~new~~ given element.

3) Merging two sets.

eg. S1 = {1, 2, 3, 4}
S2 = {5, 6, 7, 8}

S1 ∪ S2 = {1, 2, 3, 4, 5, 6, 7, 8}

Ques 6. Run BFS and DFS on the following graph.



Let 'A' be the source node and 'f' be the goal node.

1. For BFS

queue	visited
A	
{B, C, D}	{A}
{D, C, E}	{A, B}
{C, E, F}	{A, B, D}
{E, F}	{A, B, D, C}
{F}	{A, B, D, C, E}
	{A, B, D, C, E, F}

for DFS   Stack

visited	A	B	D	C	E	F
Stack	~~B~~ ~~D~~ ~~C~~	~~E~~ ~~D~~ ~~C~~	~~F~~ ~~C~~	~~E~~ F	~~F~~	

→ {A, B, D, C, E, F}

Ques7. find the number of connected components and vertices in each component using disjoint set data structure.

In disjoint set union algorithm there are two main functions i.e. connect() and root() function.

connect (): connects an edge.

Root (): Recursively determine the topmost parent of a given edge.

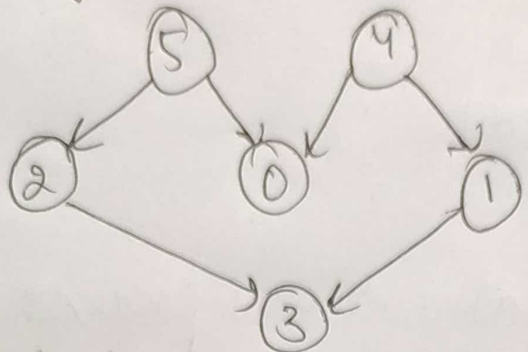
for each edge {a, b} check if a is connected to b or not, if found to be false connect them by appending their top parent.

After completing the above step for every edge, print the total no. of the distinct top-most parents for each vertex.

Pseudo code

```
int parent max;
  int root (int a)
{
    if (a == parent [a])
       return a;
    return parent [a] = root [parent [a]];
}
void connect (int a, int b)
{   a = root (a);
    b = root (b);
    if (a != b)
       parent [b] = a;
}
void connected components (int n)
{    set <int> s;
   for (i=0; i<n; i++)
   {   s.insert (root (parent [i]));
   }
  cout << s.size () << '\n';
}
void printanswer (int N, vector <vector <int>> edges
{  for (int i=0; i<=N; i++)
        parent [i] = i;
   for (int i=0; i< edge.size (); i++)
      connect (edges [i] [0], edges [i] [1]);
   connected components (N);
}
```

Ques 8. Apply topological sorting and DFS on graph having vertices from 0 to 5.



```
class graph {
   int v;
   list <int> * adj;
   void topologicalsortutil ( int v, bool visited [], stack
                                  <int> & stack);

publi :
     graph (int v);
     void addEdge (int v, int w);
     void topological sort () ;
  };
 waph :: waph (int v)
  {
     this → v = v;
     adj = new list <int>[v];
  }
 void graph :: add edg (int v, int w)
  {
     adj [v]. push_ back (w);
  }
void graph :: topological sort util (int v, boolvisited[],
                            stack <int> & stack]
```

```
{
  visited [v] = true;
   list <int>:: iterator i;
 for (i= adj [v]. begin (); i != adj [v].end();++i)
     if (!visited [*i]
        topological sort util (*i, visited , stack);
   stack. push (v);
}
void graph :: topological sort ()
{
  stack <int> stack;
  bool * visited = new bool [v];
  for (int i=0; i<v ; i++)
   visited [i] = false;
 for (int i=0; i<v; i++)
   if (visited [i]== false)
       topological sort util (i, visited, stack);
 while (stack.empty() == false)
  { cout << stack. top() << " " ;
    stack.pop();
  }
}
```

9) Heap data structure Can be used to implement priority queue? Name two graph algorithms where you need to use Priority Queue & why?

Sol:- Yes, Heapdatastructure can be used to implement priority queue

Heap data structure provides an efficient implementation of Priority Queue.

Few Graph algorithms where priority Queue is used

→ Dijkstra's Algorithm when the graph is stored in the adjacency matrix & list, priority Queue can be used to extract minimum efficiently when implementing Dijkstra's algorithm

→ Prims Algorithm To store keys of node & extract minimum key node at every step.

⇒ A* search algorithm A* search algorithm find the shortest Path between two vertices of a weighted graph.

The priority Queue is used to keep track of unexplored routes, the one for which a lower bound on the total path length is smallest is given highest priority.

10) what is the difference between Min heap & Max heap.

Sol

Min heap	Max heap
1. In min heap the key present at root node must be less than or equal to among the keys present at all of its children	1. In 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.
2. In min heap the minimum element is present at the root	2. In max heap the maximum element is present at root.
3. min heap uses the ascending priority	3. Max heap uses the descending priority

Min heap	Max heap
4. In the construction of min heap, the smallest element has priority	4. In the construction of Max heap the largest element has priority.
5. The smallest element is the first to be popped from the heap.	5. The largest element is the first to be popped from the heap.