Course: Introduction to Graph Algorithms

Professor: C Pandu Rangan

Department: Computer Science and Engineering

Institute: IISc

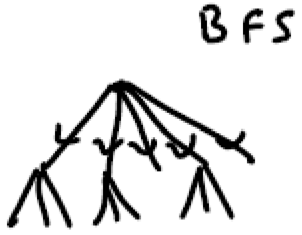
Week: 08

Lecture 37 BFS

Namaskara, in this session we are going to discuss about another way of exploring the graph and this is called breadth first search, BFS. So, we explored by going from one vertex to its neighbor and from that neighbor to another neighbor of the vertex and so on. So, we had a chain being built and grown during the search process. So, you start from there go to its neighbor from there go to its another neighbor and then this is a DFS. that is the reason why it is called the depth first search you go deep you build a chain until you cannot proceed any further okay this is visualized as starting from a route and going deeper and deeper down that is why the word depth first search. In breadth first search you start from a vertex but first explore all the neighbors okay.

So it is going to be like this. The order in which we are going to explore is account for all the unvisited neighbors and then proceed to the next level. Therefore, BFS is taking place in a natural way level by level. okay the search tree is built level by level, the search tree in DFS is built by chain going downwards okay.

You have a neighbor and here it is level by level, first you would pick up this then this kind of. So in BFS the search tree is built level by level starting from root of course.



So how do we ensure that we are building level by level? we would like to finish certain things earlier and then take up certain things later. In that case we have to put the task in a queue, whatever you want to do earlier you put it first in the queue. And whatever you want to do later you put it in the rear of the queue and queue is processed by things that are in the front of the queue and thereby an order is imposed. So the all the things in a particular level should be queued up so that when the return comes all of them are getting expanded.

okay so the breadth first search can be implemented in a natural way by using a data structure called Q. So we will have a Q of vertices right, in Q the vertices that are to be expanded we say expanded because all its neighbors are going to be considered. Here one neighbor is considered and then you went depth but here I am going to consider all neighbors and that is the reason why the action at a vertex is known as expanding the vertex, this is extending the chain, this is expanding a vertex you go to the next level and define the obviously you want to visit only the nodes that are not visited. So when you are expanding you would expand only to the nodes that are unvisited if it is visited those visited neighbors are ignored and those edges become non-tree edges.

As usual tree edges are defined only from, so tree edges they go from visited to unvisited or non-visited nodes. They form non-tree edge okay. So the computations and the way in which we would do everything is very similar to depth first search instead of stack which we have used in our iterative code we will use a queue and the bookkeeping and other things are more or less same, right. So here is the pseudo code okay you start breadth was such of a graph from a source vertex s. Now for each u in V minus s u dot color equal to white because it is not yet visited not exposed so u dot color is white.

u dot d is infinity, this is a parameter which we will maintain which is a distance from the source that is number of hops or number of edges let me write down in the previous slide, for every vertex u you maintain u dot color which will be white gray or black the same meaning white is not explored, grey means currently under consideration it is going to be it is still under processing black means the processing with respect to that node is done everything all neighbors are expanded and everything is considered. And then we have an attribute u dot d which is the so called distance from the root. That is number of edges from s to u in BFS tree okay. So, this you can also interpret as a level number with root at level 0. So this is level 0, all these will be at level 1 all these will be at level 2 and so on level by level you would proceed okay.

The level number the distance from the route number of hops how many edges you jump over to reach u starting from source or starting from the route how many edges you jump over. Okay, that is why number of hops distance all these terms are used that is u dot d if you know it is a finite number if you do not know it or if u is not reachable it will be defined as infinity and for the tree edges we have this u dot parent this is the u dot p is nil. This is the step 1

```
        BFS ($G$, $s$)
① For each $u \in V - \{s\}$
          $u$ . Color = white .
          $u$ . $d$ = $\infty$ .
          $u$ . $p$ = NIL .
```

and step 2 you have to initialize for s, s dot color equal to gray because we are now considering this. and s dot d equal to 0 the number of edges you would use to go from s to s is 0, and s dot parent in the tree is this does not have. So you have initialized for all vertices

```
②        $s$ . Color = Gray
          $s$ . $d$ = 0
          $s$ . $p$ = NIL
```

and then Enqueue Q s, this means insert s in Q include or insert s in Q.

```
③       Enqueue ($Q$, $s$) { insert $s$ in $Q$ } .
```

This is the initial setup so you have a Q you have so what you are going to do is you, while Q is not empty let u be a vertex that is removed okay Dequeue Q. Dequeue means delete, Enqueue means insert the special names are used because adding is done at the rear, deleting is done at the front that is how the queues work. You can imagine the way in which a queue elements are handled in a real life. If there is a box office and there is a queue the person in the front gets the ticket, he moves away then the next one that is how the queue operates. When a new person wants to buy a ticket he joins at the rear, end of the queue. Therefore, insertion or enqueuing is done at the rear, dequeuing or deleting is done in the front. This is a well-known queue data structure. So, this is how the queue operations are done for each v belonging adjacency of u for each v in adjacency of u. If v dot color equal to white if this is white then v dot color equal to gray, and v dot d equal to u dot d plus 1 and v dot p equal to u.

```
while ($Q \neq \phi$).
    $u$ = Dequeue ($Q$)
    For each $v \in Adj(u)$
        if ($v$.color == white)
            $v$.color = GRAY
            $v.d = u.d + 1$
            $v.p = u$.
```

So, you are expanding u and when you expand u, you get a neighbor and if it is white that is if it is not processed you are going to include in the tree. How do you say that you are including in the tree? By setting this, this is defining the tree edge, the tree edge parent of this is u, this is v, u is parent of v. u is a parent of v this defines the tree edge. So, naturally if u is at level let us say 10, v will be at the next level 11.

So, the level number or the distance is incremented by 1 and that is what, so this defines level number, level number or the distance from the route and that is it. Then after that Enqueue Q put insert v in the queue. So once all neighbors are done with a it has become black u dot color equal to black okay. u dot color equal to black and then the job is done.

So, all these things are in while loop and if this clutter is to be removed, I would write like this okay this is a tree edge u to v and u equal to parent of v, u is parent of v in BFS tree, so I would erase this, so that the code looks neat. This is a level number, so this comment I can remove, now the code looks cleaner. and this marks the end of BFS tree algorithm okay. Again, we can show that all these things are done in linear time.

You can have the version of this in directed graph and undirected graph and so on. We can have more detailed discussions on this but then we are not going to discuss them in this course. I just introduced a BFS as a simple alternate for a depth first search of course BFS is used to solve various graph algorithms like matching and so on and BFS like computation is done in other graph algorithms and applications. And in this course we will not focus on the details related to them, so this is just a cursory exposition on it couple of examples in the assignment will give you some working experience with this spanning tree, okay but we will focus much less on this topic, alright. So, if you want details related to this and if you want to explore further I will give you the references and you may follow them this session and this point. Thank you.