Day 1

Done	☑
Topic	Detect cycle in an undirected graph using BFS Dijkstra Algorithm
Languages	Java
Difficulty	⭐⭐⭐⭐
Date Started	@March 1, 2023 11:00 PM
Date Completed	@March 2, 2023 3:20 AM
Related to Progress (Days)	Your Progress

What I Learned Today

Code Implementation Of Dijstra's Algorithm

Detect cycle in an undirected graph using BFS

Quick Links

YouTube

Documentation

Dijkstra Algo

Documentation

Key Concepts

Dijkstra's Algorithm

Step1: All nodes should be marked as unvisited.

Step2: All the nodes must be initialized with the "infinite" (a big number) distance. The starting node must be initialized with zero.

Step3: Mark starting node as the current node.

Step4: From the current node, analyze all of its neighbors that are not visited yet, and compute their distances by adding the weight of the edge, which establishes the connection between the current node and neighbor node to the current distance of the current node.

Step5: Now, compare the recently computed distance with the distance allotted to the neighboring node, and treat it as the current distance of the neighboring node,

Step6: After that, the surrounding neighbors of the current node, which has not been visited, are considered, and the current nodes are marked as visited.

Step7: When the ending node is marked as visited, then the algorithm has done its job; otherwise,

Step8: Pick the unvisited node which has been allotted the minimum distance and treat it as the new current node. After that, start again from step4.

Detect cycle in an undirected graph using BFS

so today we will be solving the problem detect a cycle in an undied graph but this time we will be solving this using the bfs algorithm yes in the previous video we did learn about how to solve this problem using the dfs algorithm but over here we are going to use the other traversal technique that is known as the breath first search travis technique so just in case if you do not know about this technique i have already made a video on this technique so go back and watch that video and then you can come back and watch it again so you're given

00:30

this graph now it's not two graphs right it's a graph with two different components so if i ask you does this graph contain a cycle so you will be saying yes because even if this component doesn't contain the cycle this component does contain a cycle if any of the components in a graph does contain a cycle i can say that there is a cycle right so you just need to detect a cycle so as we know that every graph will be stored in an adjacency list so at first let's draw the adjacency list so the adjacency list of the graph will

01:09

look something like this right so now if you have seen the bfs traversal video what is the first step that we do we always consider a visiting array so let's draw a visiting array of size 11. this will be the visiting array and everything will be marked initially as unvisited which means zero right what's the next step that we always do in the bfs traversal we know that there can be multiple components right so what we do is we always run a for loop so that it calls a dfs for every unvisited node so the for loop will run from 1 to

01:45

n where n is the total number of nodes over here n will be 11 so we know that if this node is not visited right if this node is not visited it means the node i is not visited i will call the bfs traversal now over here now over here what will be the bfs traversal doing it will be checking if there is a cycle so i can say this time i'll call a function cycle bfs or cycle bfs check something like that with the node i which will simply check if there is a cycle or not if there is a cycle i can say i can return a true now again

02:23

why this because if any of the node is unvisited i'm going to call it traversal that means i'm i'm traversing for a new component and if in that component i do find a cycle that means the entire graph is a cycle so i can simply return so that's that's how we are going to do we're going to run a bfs for every component and if we find a cycle in any component we simply return it to so this will be our logic it's just that we have to decide what modifications we have to do in the bfs traversal so that

02:57

we can detect a cycle using the bfs so the modifications in the bfs have to be discussed so let's understand the thought process what is the bfs traversal it's breadth first traversal so at first you travels here then you traverse here then you try to traverse here and here simultaneously adjacent nodes then you traverse here and here right after that when you traverse here right after this it will try to traverse to its adjacent node that is 7 and since it tries to traverse to its adjacent node 7 and it finds that that has already been

03:41

traversed which means there is a cycle now had there not been a cycle that means this node this edge would not have been there and 8 would never have found its adjacent node to have been already visited so this will be our thought process if any of the adjacent node has been visited previously then i can say that there is a cycle right for this eight the next node was seven that was visited previously so i could say that there is a cycle so let's see how we will be doing this in the dfs travis so at first

04:20

the value of i will be one so what we do is we take the value of is one and we check if one is unvisited so we see that one is unvisited so that means we are going to call a cycle bfs right so we call a cycle bfs with the value one so let's call the cycle bfs check or whatever your function name is with the node value one right how does bfs works it initially takes a q right any it initially takes a cube now this time we will modify the cube in the normal bfs traversal we did just put the node in it over here we're gonna put the node

04:59

as well as the previous or the parent node i'll tell you the significance of the parent node as we move forward in discussing the modifications in our bfs travis so initially i can say over here the starting node will be 1 and i can say 1 doesn't have any previous node it doesn't has right so i can put a minus 1 saying that the starting node will never have a previous node because this is the point where i start traversing so i will surely not have anything on my back so i'm pointing it by -1 so this will be

05:35

initialization it will take a q which will have a node parent and you'll insert one comma minus one and at the same time please make sure that one is marked as visited and you'll have this visited array after this you're gonna run a while loop right which traverses for all the nodes so the first node that you get is one and the parent node or the previous node is minus one and let's check out what are the adjacent node for the node one so the adjacent node is two so you check out is 2 visited no 2 is not visited so

06:09

obviously it's not a cycle because not previously visited so what you do is you take this 2 and put it into our queue now when you put it into the queue what will be the previous node now you have travis from 1 now you're trying to go to 2 right so that means the previous node will be 1 because you came from 1 so the previous node will be 1 so you put this one into your queue so once you have done this please make sure you mark 2 as visited also so does one have any further adjacent nodes i cannot see so

06:43

the traversal for node 1 is over next time again you check which is the topmost value at your node so at your q so the value that you get is 2 what's the previous node 1 so now if you check out what are the adjacent nodes of 2 so the adjacent node of 2 is 1 so you check out is that visited yes it is visited but can you say that it is a cycle no why because this one which is visited is actually the previous note so from the position you came that that is what it is giving as an adjacent node so that's that's not a

07:25

cycle that's just the previous node that is why it's visited so i will ignore this next i check out four okay is that visited no let's mark it as visited and let's put that into a queue four comma what will be the previous node two because four came from 2 do we have any further adjacent nodes for 2 no so the traversal for 2 is also completed right next what you'll do you'll have the node value 4.

07:55

so let's take that out so when you take the node value 4 out you have the previous node as 2 let's check out the adjacent nodes of 4 so the adjacent node of 4 is 2 i see is that visited yes it's visited but will you call it as a cycle no because that's a previous node and obviously if you came from a previous node it's bound to be visited so you came to four so the previous node is two that's an adjacent node so it's visited because you came from it so i can ignore this two and i'll not put it into our queue so i can say there are no further

08:30

nodes for four so the traversal for four is complete so once you complete the traversal for four you see that the q becomes empty that means you have done the bfs traversal for the node starting with one that means this entire component has been visited but did you find a cycle no so this function is going to return a false right and whenever it returns or falls it gets a false it means this component did not have a cycle so we are going to move across to the next component to check for a cycle so the i will become 2 is 2

09:06

visited yes because this component was already visited next time i will become 3 is 3 visited no so 3 is not visited i go to inside that and call the cycle bfs function so the cycle bfs function will now call three right what a what were your steps in the bfs traversal you always declared a queue and since you have modified the bfs this time the bfs will contain node and pattern so initially you have the node 3 and since this is the starting point i can say that the initial parent will be minus 1 and what is the other thing that

09:43

you did before starting on with the traversal please make sure you mark 3s visit it so you have done that right now what will you do now now the traversal starts so the traversal starts with the value node three and the previous as minus one so let's check out the adjacent nodes of three so the adjacent nodes of three is five so you basically check if five is unvisited yes that's unvisited so let's visit five and make sure you put that into the queue so five comma three because three will be the node from where did

10:20

five come so three is visited now you have made sure that five is also visited so the traversal for three is over because three doesn't have any further adjacent nodes now it's time to take the next guy out that's five so let's take five out so if you take five out the previous node will be three so let's check out for the adjacent nodes of five so the adjacent nodes of five is 3 but can you say that it's a cycle no because 3 is again an adjacent node which is actually previous so it's

10:53

bound to be visited so next node you get is n so you check out 10 visited and you see that 10 is not visited so what you do is you mark it as visited and you take that 10 and you put that into your queue data structure so if you put that into your q data structure it will be 10 comma 5 right what about the next adjacent node it's 6 so do you see 6 as visited no so please make sure you mark it as visited and you put it into your queue so 6 comma 5 does go into your queue right so put that into your queue what's your next step obviously your

11:33

next step will be to check if there are any further adjusted notes and you see that there are no further adjusted notes so the traversal for the node 5 is over next time you get this node 10 so let's check so let's take out the node 10. so in the previous traversal 6 and 10 where visited right so the previous node for 10 is 5 so let's check out the adjacent nodes for 10.

11:59

so the adjacent nodes for 10 is 5 now you see that 5 is visited what will you call it as a cycle no because 5 is the previous node itself so the previous node is bound to be visited so there is no need to put it into the queue what's the next adjacent node 9 is 9 visited no so please make sure you mark it as visited and you take 9 and the previous node will be 10 for 9 right so you put it into your queue so you visit 9 also are there any further adjusted nodes for 10 no so you just complete the travis for 10 now you get the next guy that is 6.

12:37

so let's write six so you have a six and the parent of the previous node for six is five right so let's check out the adjacent nodes for six so for six the adjacent nodes are five again 5 is the parent node that is why 5 is visited so it's not a cycle it's the previous node so it's not a cycle next adjacent node for 6 that is 7 is 7 visited no so you please make sure you mark it as a visited and you put it into your queue data structure so 7 comma 6 is what you put in the queue so 7 is also visited

13:15

so are there any further adjacent nodes for 6 because 7 is also done no so i can say the traversal for 6 is over now let's check out what's the next note so the next node is basically 9 where the previous is or the parent is 10. so let's check out the adjacent nodes for 9. the first adjacent node for 9 is 10 10 is already visited but 10 is the previous of the parent node so you cannot call this as a cycle so you will not take this node into consideration let's check out the next answer note it's eight so this guy eight

13:52

is actually not visited so please mark it as visited and you please make sure that eight is being taken right so you take eight and the previous node for this 8 will be 9 so that means this 8 came across this guy this 8 came across this path understand it came across this path right so are there any further adjacent nodes for 9 no so the traversal for nine is over next is the most important step so you see that the next value is seven six so please make sure you write this so node of seven and the previous is six

14:31

so for seven what is the adjacent node a one is six that's very obvious because seven did come up from six that is why the previous or the parent node is six so i can say that's an adjacent so that's why i cannot say that it is a cycle but the next adjacent node for 7 is 8 and you see that 8 is already visited but 8 is not the previous node so if 7 had a traversal from this side how did 8 got marked as visited that means someone would have traveled from this side right and it would have marked it visit it now since you're traversing

15:13

from this side and you came across to it that means there is a cycle there is a cycle that is what bfs will tell you because you traverse breath wise in this direction someone else did travel breath breathwise in this direction and ultimately both of you did reach this same guy eight which means there is a cycle in your component in this component which ultimately means there is a cycle in your entire graph so you can say right over here the moment you see that this 8 is not equivalent to the parent and that's marked as visited

15:48

you can say there is a cycle and it can directly return it true saying that there is a cycle so this guy gets a true and this returns a true which means there is a cycle in your graph so the stuff over here is very simple if in any of the component you get a cycle it means in the entire graph you can say that there is a

cycle so that is what i have done over here so the thought process is very simple we try to maintain a previous so that if we are following a path and in the next we get a node which has been visited by someone

16:23

else that means there is a cycle right so let's talk about the time complexity of this algorithm i can say since the bfs traversal takes big o of n because there are n nodes through travis so i can say that the time complexity will be big o of n and what about the space complexity i can call the space complexity as big o of n in general terms because i'm using a visited array i'm using a q but it's near about n so i can call the near about time complexity and the space complexity to be big o of n if

16:58

we use the bfs traversal now you must be thinking if both the algorithms have the same complexity which one to use again it's completely on you you can use any one of the algorithms to detect a cycle either the bfs or the dfs now it's time to discuss the c plus plus as well as the java code for this cycle detection using bfs so let's discuss the c plus plus solution to the algorithm so you know how to store a graph in an adjacency list and let's assume the number of nodes is n so i'm basically

17:29

calling a function a cycle with the total number of nodes as well as the graph that is stored in the adjacency list and telling the function to tell me if there is a cycle in an entire graph so this function will initially declare a visited array which will be marked as 0 what's the next step i know in any of the components if there is a cycle that means the graph has a cycle so i run a for loop and i will be calling check for cycle for every unvisited node the stuff that i did explain while explaining the algorithm right so

18:04

basically call the check for cycle or the parameters that you will pass you will pass the starting node for which the bfs will be called along with that make sure you pass on the variables like the number of nodes the graph adjacency list and the visited array so if this bfs call returns a true that means the component has a cycle so you're going to return it to right so i write a boolean check for cycle and this does take the starting node for which the bfs will start along with this it takes the number of nodes this is not required but

18:39

i've just passed it on and the adjacency list as well as the visited array now what i do is i declare a queue right of pair which stores the node as well as the previous or the parent whatever you want to call it now i make sure that the node where i am starting is marked as visited and i'm also inserting the initial starting node with the previous as minus one so these were the initializations that i did in the modified bfs right after that you know you're gonna till your q is not empty or if you find

19:12

a cycle so what i do is i take the top most element in the queue because that is what we will be traversing for so you get the node as well as the previous right after that your traverse for all its adjacent nodes so let's travis for all its adjacent node so what do you check is if that's not if that adjacent node has not been visited yet that's a very simple task your market is visited right and you put it into your queue simple as that but what if that's visited what if this f is false that

19:42

means the adjacent node has been visited and if the adjacent node is not among the previous one what does that mean someone else would have been visited someone else would have visited that if your

adjacent node is visited and that's not equivalent to your parent or previous it means there is a cycle so your return are true so you can return it through the moment you find it and the bfs will be over but if you complete this traversal as you saw in the example for the first component the queue will become empty in

20:18

that case the component doesn't have a cycle so you're going to return a false so this is how your modified vfs will look like and it will give you an answer if there is a cycle in a component or not so this will be your entire code if this returns true at any moment it means there is a cycle if it would have checked for every node and it did not find a cycle it will lastly come across to the line number 45 and it will return a false so this will be the code for c plus plus i'll be leaving the code in the description

20:48

below right so do check it out so let's discuss the java solution to the logic that we did discuss so since we know that a graph can be stored in an adjacency list and you know the number of notes is v so i have written a function s cycle which is going to take your graph in forms of adjacency list and it will take the number of notes and it this function will check if there is a cycle in the entire graph and it's going to return a true if there is and if there is no cycle it is going to return a false so

21:21

initially what i do is i basically create a visited array and make sure everyone is marked as unvisited which means false now the next step i need to check for every component because i know if i'm checking for a graph in that case if any of the component contains a cycle that's going to return a true so as already discussed i run a loop from 1 to the number of nodes that is v and if the node is unvisited i basically call the dfs i basically call the bfs the modified vfs the check bfs the check for cycle bfs whatever you want to name it

21:54

you can and i just pass in the starting node along with the graph and the visited array so this is my function which takes the adjacency list takes the starting node for which the bfs is called and takes the visited array now what i have done is as yes as you did see that the queue was containing two elements so i've created a class of node which contains the first and second basically first will be the node and the second you can say as the parent right so it does contain that so you know what is the initial step the

22:27

starting node has to be inserted into the queue and the parent of that will be minus 1 because that's the starting node so you insert s comma minus 1 and whatever is the first step that's marking it as visited so please make sure the initialization steps are completed or putting the starting node into the queue and marking it as visited now what was the next step you kept on traversing for every node till the queue didn't become empty so let's do it so what you do is you get the first guy that's the node and the parent

22:58

right you just get the first guy from the queue once you've got it please make sure you remove it because that will not be required for your next iteration so once you've removed travels for all its adjacent node adjacence dot get off node will give you the list of the adjacent nodes and you can for each loop in that so what i get is i t as the adjacent node so if this node has not been travised the task was very simple if the node was not traversed we always mark this as visited and we did put that into the queue now how will you put that

23:32

into the queue the node will be i t and the previous node from where it came will be node so please make sure you put them into the queue but what if that adjacent node is visited if visited id is true it means this adjacent node has been visited there can be two options either it's the previous node then means no cycle but what if it's not the previous node and visited it means it would have been visited by someone else which apparently gives you an indication that there is a cycle so i can simply return a true which says me

24:06

that there is a cycle someone else has marked that visited so i return true but what if this never gets executed like in the example you saw the first component went on and ultimately the cube become empty after all the nodes were traversed so in that case that this while loop will get over and you can return a false saying that this component did not give me any cycles so basically if any of the times this returns a true it will ultimately return a true but if the entire for loop is executed and it did not find a cycle for any component

Code Snippets

```
//Dijkstra's Algorithm
import java.util.ArrayList;
import java.util.PriorityQueue;

public class DijkstraAlgo {
    static class Edge {
        int src;
        int dest;
        int wt;
        public Edge(int s, int d, int w){
            this.src=s;
            this.dest=d;
            this.wt=w;
        }
    }
    static void createGraph(ArrayList<Edge> graph[]){
        for (int i=0; i< graph.length; i++){
            graph[i]=new ArrayList<>();
        }
        graph[0].add(new Edge(0,1,2));
        graph[0].add(new Edge(0,2,4));

        graph[1].add(new Edge(1,3,7));
        graph[1].add(new Edge(1,2,1));

        graph[2].add(new Edge(2,4,3));

        graph[3].add(new Edge(3,5,1));

        graph[4].add(new Edge(4,3,2));
        graph[4].add(new Edge(4,5,5));
    }
    static class Pair implements Comparable<Pair>{
        int n;
        int path;
        public Pair(int n, int path){
            this.n=n;
            this.path=path;
        }
        @Override
        public int compareTo(Pair p2){
            return this.path - p2.path; //path based sorting for pairs
        }
```

```
    }
    public static void dijkstra(ArrayList<Edge> graph[], int src){ // O(V+ElogV) -> Priority Queue
        int dist[] = new int[graph.length];
        for (int i=0; i< graph.length; i++){ //dis[i] -> src to i
            if (i != src){
                dist[i] = Integer.MAX_VALUE; //+infinity for neighbour of src
            }
        }
        boolean[] vis = new boolean[graph.length];
        PriorityQueue<Pair> pq = new PriorityQueue<>();
        pq.add(new Pair(src,0));
        //loop
        while (!pq.isEmpty()){
            Pair curr = pq.remove();
            if (!vis[curr.n]){
                vis[curr.n] = true;
                //neighbours ke liye check karlo
                for (int i=0; i< graph[curr.n].size(); i++){
                    Edge e = graph[curr.n].get(i);
                    int u = e.src;
                    int v = e.dest;
                    int wt = e.wt;

                    if (dist[u]+wt < dist[v]){
                        dist[v] = dist[u]+wt;
                        pq.add(new Pair(v,dist[v]));
                    }
                }
            }
        }
        for (int i =0; i< dist.length; i++){
            System.out.print(dist[i]+" ");
        }
        System.out.println();
    }
    public static void main(String[] args) {
        int V=6;
        ArrayList<Edge> []graph = new ArrayList[V];
        createGraph(graph);
        int src = 0;
        dijkstra(graph, src);
    }
}
```

```
//Detect cycle in an undirected graph using BFS
import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Queue;

public class GraphPractice {
    static void addEdge(ArrayList<Integer> graph[], int u, int v) {
        graph[u].add(v);
        graph[v].add(u);
    }

    public static boolean detectCycleUtilBFS(ArrayList<Integer>[] graph, int curr, boolean[] visited, int V) {
        int parent[] = new int[V];

        Queue<Integer> q = new LinkedList<>();
        visited[curr] = true;
        q.add(curr);
        while (!q.isEmpty()) {
            int u = q.poll();
            for (int i = 0; i < graph[u].size(); i++) {
                int v = graph[u].get(i);
                if (!visited[v]) {
                    visited[v] = true;
                    q.add(v);
```

```
                    parent[v] = u;
                } else if (parent[u] != v) {
                    return true;
                }
            }
        }
        return false;
    }

    public static boolean detectCycleBFS(ArrayList<Integer>[] graph) {
        boolean[] visited = new boolean[graph.length];
        Arrays.fill(visited, false); // we are marking false in all
        for (int i = 0; i < graph.length; i++) {
            if (!visited[i]) {
                if (detectCycleUtilBFS(graph, i, visited, graph.length)) {
                    return true;
                    //Cycle exists in one of the parts
                }
            }
        }
        return false;
    }

    public static void main(String[] args) {
        int V = 4;
        ArrayList<Integer> graph[] = new ArrayList[V];
        for (int i = 0; i < 4; i++) graph[i] = new ArrayList<Integer>();
        addEdge(graph, 0, 1);
        addEdge(graph, 1, 2);
        addEdge(graph, 2, 0);
        addEdge(graph, 2, 3);

        System.out.println(detectCycleBFS(graph));
    }
}
```

Challenges Experienced

Resource Links

Dijkstra Algorithm Java - Javatpoint

Dijkstra Algorithm Java with java tutorial, features, history, variables, object, programs, operators, oops concept, array, string, map, math, methods, examples etc.

https://www.javatpoint.com/dijkstra-algorithm-java

https://www.youtube.com/watch?v=A8ko93TyOns