



IIT Madras
ONLINE DEGREE

Mathematics for Data Science 1
Professor Madhavan Mukund
Chennai Mathematical Institute
Lecture: 62
Breadth-first search

(Refer Slide Time: 00:14)

Breadth First Search

Madhavan Mukund
<https://www.cmi.ac.in/~madhavan>

Mathematics for Data Science 1
Week 10

Reachability in a graph

- Mark source vertex as reachable
- Systematically mark neighbours of marked vertices
- Stop when target becomes marked

Graph structure (vertices v_0 to v_9):

```
graph TD; v0 --> v1; v0 --> v2; v1 --> v3; v2 --> v4; v3 --> v6; v4 --> v5; v5 --> v7; v5 --> v8; v7 --> v9; v8 --> v9;
```

So, we have been looking at this question of reachability in a graph. So, we said that to find whether a vertex is reachable for target vertex is reachable from a source vertex, we systematically explore the graph, we mark the source vertex, and then we go to its neighbours mark its neighbours and keep doing this systematically until the target becomes marked.

(Refer Slide Time: 00:36)

Reachability in a graph

- Mark source vertex as reachable
- Systematically mark neighbours of marked vertices
- Stop when target becomes marked
- Choose an appropriate representation
 - Adjacency matrix
 - Adjacency list

	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	0	1	0	0	0
4	1	0	0	1	0	0	0	1	0	0
5	0	0	0	1	0	0	0	1	0	0
6	0	0	0	0	0	1	0	0	0	0
7	0	0	0	0	1	0	0	0	1	0
8	0	0	0	0	0	1	0	0	0	1
9	0	0	0	0	0	0	0	0	1	0

	0	1	2	3	4
0	{1,4}				
1	{2}				
2	{0}				
3	{4,6}				
4	{0,7}				

	5	6	7	8	9
5	{3,7}				
6	{5}				
7	{8}				
8	{5,9}				
9	{8}				

Mathavan Mukund Breadth First Search Mathematics for Data Science

So, what we saw last time was that, in order to do this as a procedure, we have to have a way of representing the graph, which is not a picture. So, we came up with these two different representations. The adjacency matrix has a row and a column for every vertex. So, remember that we are assuming that our vertices are always numbered 0 to n minus 1 for convenience.

So, we have an entry at row i column j , it indicates whether or not there is an edge from i to j . So, if it is a 1, it means there is an edge if there is no 1, if it is a 0, it means there is no edge. And then we observe that this could be a wasteful representation. If there are not that many edges, a number of entries in this matrix are 0. So, instead, we could just record an adjacency list for each vertex, just record the neighbours.

(Refer Slide Time: 01:23)

Reachability in a graph

- Mark source vertex as reachable
- Systematically mark neighbours of marked vertices
- Stop when target becomes marked

Madhavan Mukund Breadth First Search Mathematics for Data Science

So, if we go back to this graph that we had just now, So, if we look at V_4 , for example, So, V_4 has an outgoing edge to V_0 , and outgoing is to V_3 , it also has an incoming edge from V_0 and V_3 , and then it has an outgoing edge and incoming it from V_7 , So, the neighbours of 4 are 0, 3, and 7.

(Refer Slide Time: 01:40)

Reachability in a graph

- Mark source vertex as reachable
- Systematically mark neighbours of marked vertices
- Stop when target becomes marked
- Choose an appropriate representation
 - Adjacency matrix
 - Adjacency list

	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	0	1	0	0	0
4	1	0	0	1	0	0	0	1	0	0
5	0	0	0	1	0	0	0	1	0	0
6	0	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	1	0	0	0	1
9	0	0	0	0	0	0	0	0	1	0

0	{1,4}	5	{3,7}
1	{2}	6	{5}
2	{0}	7	{8}
3	{4,6}	8	{5,9}
4	{0,3,7}	9	{8}

Madhavan Mukund Breadth First Search Mathematics for Data Science

Reachability in a graph



- Mark source vertex as reachable
- Systematically mark neighbours of marked vertices
- Stop when target becomes marked
- Choose an appropriate representation
 - Adjacency matrix
 - Adjacency list
- Strategies for systematic exploration
 - Breadth first — propagate marks in "layers"
 - Depth first — explore a path till it dies out, then backtrack

	0	1	2	3	4	5	6	7	8	9
0	0	1	0	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	0	1	0	0	0
4	1	0	0	1	0	0	0	1	0	0
5	0	0	0	1	0	0	0	1	0	0
6	0	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0
8	0	0	0	0	0	1	0	0	0	1
9	0	0	0	0	0	0	0	0	1	0

0	{1,4}
1	{2}
2	{0}
3	{4,6}
4	{0,7}
5	{3,7}
6	{5}
7	{8}
8	{5,9}
9	{8}



Madhavan Mukund

Breadth First Search

Mathematics for Data Science

10 2

So, if you look at this adjacency matrix, it says that in the neighbours of 4, if you look at the row 4, it has once at positions 0, 3, and 7. And here, it does not have. So, there is 1 entry missing here. So, it should be 0, 3, and 7. So, this is how we would represent the matrix as either the graph is either an adjacency matrix or adjacency list. So, what we are going to look at now is the second part of the story, which is having represented the graph in this way where we can manipulate it, how do we actually systematically explore the graph.

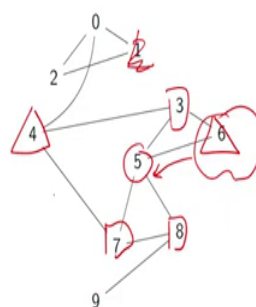
So, we did a kind of ad hoc exploration last time using the adjacency matrix. But this time, we want to do it systematically. And there are 2 systematic ways to do this 1 is called breadth-first and 1 is called depth-first. So, first, we will look at breadth-first.

(Refer Slide Time: 02:33)

Breadth first search (BFS)



- Explore the graph level by level
 - First visit vertices one step away
 - Then two steps away
 - ...
- Each **visited** vertex has to be **explored**
 - Extend the search to its neighbours
 - Do this only once for each vertex!



Madhavan Mukund

Breadth First Search

Mathematics for Data Science

10 3

So, when we explore a graph, breadth-first, we do it level by level. So, we start with the vertex, then we identify all the neighbours have this vertex, that is all the edges, which connect this vertex to its neighbours, we look at those endpoints and say that these are now all reachable from this vertex, then we go to those neighbours, and look at what is reachable from there. So, these are things which are 2 levels away. So, 1 level away at the neighbours of the source vertex, 2 levels away are the neighbours of the neighbours, and so, on.

Now, what happens, of course, is that we might end up with a situation where we start with 5, and then we identify its neighbours as say, 3, 8, and 7. And then we go to 3, and then we identify its neighbours as 4, 6, and 1, or not 4 and 6. And now i look at the neighbours of 6. And it says, oh 5 is a neighbour of 6. But we started with 5. So, we need to record that a vertex has been visited. And we need to make sure that we do not visit a vertex twice. Otherwise, we will go round and round this kind of a triangle or a cycle a number of times.

(Refer Slide Time: 03:39)

Breadth first search (BFS)

- Explore the graph level by level
 - First visit vertices one step away
 - Then two steps away
 - ...
- Each **visited** vertex has to be **explored**
 - Extend the search to its neighbours
 - Do this only once for each vertex!
- Maintain information about vertices
 - Which vertices have been visited already
 - Among these, which are yet to be explored

Mathavan Mukund Breadth First Search Mathematics for Data Science 10 3

So, we need to do this visiting and exploring. So, exploring means we go from there to 1 more level of neighbours. So, we need to do this visiting and exploring exactly 1 per vertex. So, we need to maintain some information. So, we need to maintain information about which vertices have been marked as visited, that is, which note vertices have been marked as neighbours of something we have already seen.

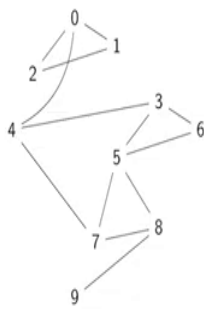
And in the process of exploration, these have been marked, but we still have not looked at their neighbours. So, there is a subsequent process after visiting a vertex of exploring its neighbours. So, whether or not such a visited vertex remains to be explored. So, we need to keep 2 things. Has it been visited, and has it been explored? So, of course, it will be explored only after it is

visited. But if it has been visited and not explored, that means this is some pending work that we have to do.

(Refer Slide Time: 04:26)

Breadth first search (BFS) ...

- Assume $V = \{0, 1, \dots, n-1\}$
- $\text{visited} : V \rightarrow \{\text{True}, \text{False}\}$ tells us whether $v \in V$ has been visited
 - Initially, $\text{visited}(v) = \text{False}$ for all $v \in V$
- Maintain a sequence of visited vertices yet to be explored
 - A **queue** — first in, first out
 - Initially empty
- Exploring a vertex i
 - For each edge (i, j) , if $\text{visited}(j)$ is False,
 - Set $\text{visited}(j)$ to True
 - Append j to the queue



So, as we said, we will always assume that we have n vertices, we call them 0 to n minus 1. So, here is an undirected graph with 9 vertices, 10 vertices, numbered 0 to 9. So, what we will do is record the visited information as a function. If you are thinking about it in programming terms in terms of the computational thinking course, you can think of it as a dictionary whose keys are 0 to 9. It does not really matter, but it is actually mathematically a function for each vertex, it tells us true the vertex has been visited or false the vertex has not been visited so, far.

So, initially, we assumed that no vertex has been visited because we have not explored the graph at all. So, initially visited of V is false for every vertex. And now, we also separately have to look at the vertices which have been visited that have been marked true by visited , but which have not been explored. So, exploration means looking at the neighbours of that vertex. So, we have not yet looked at the neighbours of that vertex. So, we have to keep a record of this. So, this is a sequence of vertices which have been so, far visited but not yet explored.

And we will keep this in a special kind of sequence called a queue. So, the queue has exactly the same meaning that you associate with an English queue. So, if you are standing in a line to buy a ticket, a queue forms you join at the end of the line, the person at the front of the line buys a ticket and moves away, the next person moves up to the front of the line. And gradually as you move forward, when you reach the head of the queue, you get your turn. So, the same way we will maintain these vertices in a queue. As they get visited, they will be added to the queue, when their turn comes, they will be explored.

So, exploring a vertex technically means the following, we want to look at all the edges which are outgoing from that vertex. So, we look at every edge i, j , which is there in the graph. And if we have not yet visited j , then we mark j as being visited, and we add j to this queue of unexplored vertices. So, j has been marked now as visited. So, it is due to be explored when it is turned comes. So, we put it at the end of the queue.

(Refer Slide Time: 06:26)

Breadth first search (BFS) ...

- Initially
 - $visited(v) = \text{False}$ for all $v \in V$
 - Queue of vertices to be explored is empty
- Start BFS from vertex j
 - Set $visited(j) = \text{True}$
 - Add j to the queue
- Remove and explore vertex i at head of queue
 - For each edge (i, j) , if $visited(j)$ is **False**,
 - Set $visited(j)$ to **True**
 - Append j to the queue
- Stop when queue is empty

Madhavan Mukund Breadth First Search Mathematics for Data Science 10

So, suppose, we start our BFS from a vertex j . So, what we will do is initially we will set $visited$ of this vertex to be true, because we start there, So, we have visited that vertex, and now we have to explore it. So, what we do is we just put it into the queue, because our procedure is going to be to systematically explore all the vertices which are in the queue. So, we set the $visited$ of j to true and we add j to this queue. And now, what we do repeatedly is we keep removing the vertex at the head of the queue. So, this queue, as I said is a line of vertices waiting to be explored.

So, we pick up the next 1, which is waiting, which is at the head of the queue the front of the queue, and explore it. And exploring it, as we said, is to check whether its neighbours which are visited, its neighbours have been visited or not. So, if a neighbour has not been visited, we mark it as visited and put it in the queue.

So, how do we stop? Well, if there is nothing in the queue to visit anymore, to explore anymore, this means that we have visited vertices and all the vertices we have visited, we have explored and there is nothing left to do. So, when the queue becomes empty, this breadth-first search as it is called BFS terminates.

(Refer Slide Time: 07:47)

BFS from vertex 7

Visited
0 False
1 False
2 False
3 False
4 False
5 False
6 False
7 False
8 False
9 False

To explore queue

head tail

Madhavan Mukund Breadth First Search Mathematics for Data Science

BFS from vertex 7

Visited
0 False
1 False
2 False
3 False
4 True ✓
5 True ✓
6 False
7 True ✓
8 True ✓
9 False

To explore queue

4 5 8

- Mark 7 and add to queue
- Explore 7, visit {4,5,8}

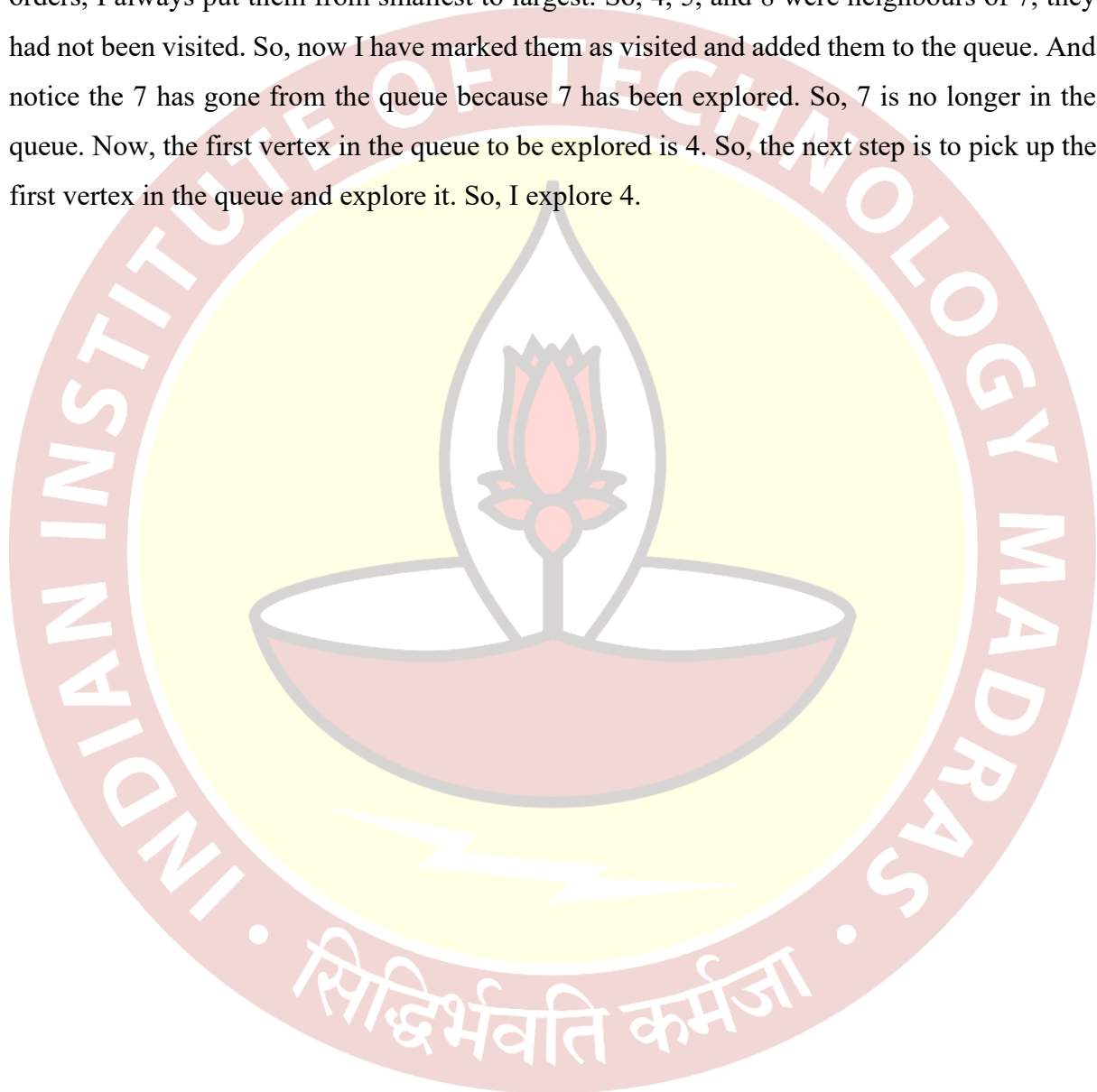
Madhavan Mukund Breadth First Search Mathematics for Data Science

So, let us try an exploration of this graph. So, let us assume that we start at this vertex 7. So, we are going to start with 7. So, as we said, initially, we have set visited to false for all the vertices. And initially, we have this queue. So, the queue, I am going to assume has the left side as the front and the right side as the end. So, the head and the tail of the queue, we joined from the current tail of the queue, and we leave from the head of the queue. So, initially, the queue is empty, because we have not started anything yet. And initially, everything is unmarked. So, everything has visited set to false. So, we are starting from 7.

So, the first thing we do is that we mark 7, as visited. So, we mark 7 as visited. And Just to illustrate, we have also marked it on the graph. And now we have also put 7 in the queue, saying that 7 needs to be explored. So, we mark 7 as visited and added to the queue. Now, the

real breadth-first search starts, which is you pick up the first element in the queue, and explore it. So, we pick up the 7, explore it. So, what are the neighbours of 7, the neighbours of 7 are 4, 5 and 8. So, in this process, 4, 5, and 8 get marked. And now they also get added to the queue.

So, I put them in some order, I have just put them in this particular case in ascending order, it does not matter, i could put it as 8,5,4 and 5,4,8. But is just convenient to put it in some fixed orders, I always put them from smallest to largest. So, 4, 5, and 8 were neighbours of 7, they had not been visited. So, now I have marked them as visited and added them to the queue. And notice the 7 has gone from the queue because 7 has been explored. So, 7 is no longer in the queue. Now, the first vertex in the queue to be explored is 4. So, the next step is to pick up the first vertex in the queue and explore it. So, I explore 4.



(Refer Slide Time: 09:35)

BFS from vertex 7

Visited	
0	True
1	False
2	False
3	True
4	True
5	True
6	False
7	True
8	True
9	False

To explore queue							
5	8	0	3				

- Mark 7 and add to queue
- Explore 7, visit {4,5,8}
- Explore 4, visit {0,3}

Madhavan Mukund Breadth First Search Mathematics for Data Science 1, Week 10

BFS from vertex 7

Visited	
0	True
1	False
2	False
3	True
4	True
5	True
6	True
7	True
8	True
9	False

To explore queue							
8	0	3	6				

- Mark 7 and add to queue
- Explore 7, visit {4,5,8}
- Explore 4, visit {0,3}
- Explore 5, visit {6}

Madhavan Mukund Breadth First Search Mathematics for Data Science 1, Week 10

BFS from vertex 7

Visited	
0	True
1	False
2	False
3	True
4	True
5	True
6	True
7	True
8	True
9	True

To explore queue							
0	3	6	9				

- Mark 7 and add to queue
- Explore 7, visit {4,5,8}
- Explore 4, visit {0,3}
- Explore 5, visit {6}
- Explore 8, visit {9}

Madhavan Mukund Breadth First Search Mathematics for Data Science 1, Week 10

So, if i look at 4, it has neighbours, 0 and 3, and 7. But 7 is already been visited. So, i do not have to do anything about 7, i pick up the 2 which have not been visited, which is 0 and 3, and set their visited status to true and i add them in the queue in some order. In this particular case, as I said, I will put 0 before 3 Just because it is a smaller number. So, now i have finished with 4. So, 4 has left the queue. But from the previous rounds 5 and 8 are still pending.

So, 7 added 4, 5, and 8, i finished 4, 5, and 8 are still pending. So, 0 and 3 will have to wait their turn, they will have to wait until we are finished. So, in some sense, 4, 5, and 8 were 1 level away. So, until I finish all the vertices, which are 1 level away, I am not going to explore the vertices, which are added at level 2, namely, 0 and 3. So, what I do is I know to pick up 5 and explore it. And in the process, I look at the neighbours of 5, so, there are 3, 7, and 6. But 3 and 7 have already been visited so far, so, we do not have to look at them again. But 6 is new.

So, I marked 6, and I put it in the queue. Similarly, now i will pick up 8. And from 8 again, I have vertices, which are 5, 7, and 9. But since 5 and 7 were already visited, what is added now is 9 and 9 gets put into the queue. So, now I finally finished the level 1 vertices in the queue and have come to the 0 which is added at level 2.

(Refer Slide Time: 11:04)

BFS from vertex 7

Visited	To explore queue
0 True	3 6 9 1 2
1 True	
2 True	
3 True	
4 True	
5 True	
6 True	
7 True	
8 True	
9 True	

- Mark 7 and add to queue
- Explore 7, visit {4,5,8}
- Explore 4, visit {0,3}
- Explore 5, visit {6}
- Explore 8, visit {9}
- Explore 0, visit {1,2}

Madhavan Mukund Breadth First Search Mathematics for Data Science 1, Week 1

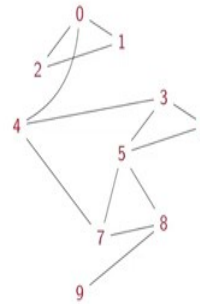
BFS from vertex 7



Visited	
0	True
1	True
2	True
3	True
4	True
5	True
6	True
7	True
8	True
9	True

To explore queue

- Mark 7 and add to queue
- Explore 7, visit {4,5,8}
- Explore 4, visit {0,3}
- Explore 5, visit {6}
- Explore 8, visit {9}
- Explore 0, visit {1,2}
- Explore 3
- Explore 6
- Explore 9
- Explore 1
- Explore 2



So, I explored the 0. So, 0 has 3 neighbours, 1, 2, and 4, but 4 is already visited earlier. In fact, 4 is where we came to 0 from in some sense. So, we now mark, 1 and 2. So, now you can see actually that everything has been marked, and 1 and 2 get into the queue. So, at this point, in principle, everything has been marked and you could stop.

But this is not how BFS stops, BFS stops by checking that every visited vertex has been explored. So, what we will do now is we will go to 3 and explore it, but we find that all the neighbours of 3 are already visited. So, exploration does not add anything to the queue, it only removes the 3 from the queue.

Next, we explore 6 similarly, all the neighbours of 6 have already been explored. So, exploration of 6 only remove 6 from the queue does not add anything to the queue does not mark anything new. Similarly, we have to explore 9 again, nothing new, explore 1, again, nothing new, explore 2, again, nothing new.

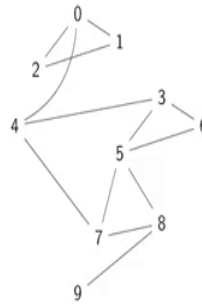
And now the queue is empty, I have run out of work to do. So, I have processed every vertex that I visited during my breadth-first search. And as a result, I have marked all the visit vertices, which I could visit starting from 7. In this particular case, you can see obviously, from this graph, that everything can be reached by some parts of the other from 7. So, everything is marked as true. So, this is breadth-first search.

(Refer Slide Time: 12:27)

Enhancing BFS to record paths



- If BFS from i sets $visited(j) = \text{True}$, we know that j is reachable from i
- How do we recover a path from i to j ?
- $visited(j)$ was set to True when exploring some vertex k
- Record $parent(j) = k$
- From j , follow $parent$ links to trace back a path to i



So, now we know for instance, that we can reach 1 from 7 because everything was marked as true. So, $visited(1)$ is true. So, clearly, there was a way to go from 7 to 1. But this information that we have recorded in this visited vertex does not tell us anything about how to do this. So, if we have visited have j equal to true, after we have explored breadth-first search for i , we know that j is reachable from i . But we do not have any information about the path. So, this is now something that we can fix. So, how did we get to i, j , we reached j , because we explored j from some k .

So, if we keep track of how we reached each vertex, we can work backward and extract the path. So, $visited(j)$ was set to true when we explored some vertex k . So, we can say that the parent of j , the reason that j got added to the visited set was k . So, now if we go follow back from j to k , through this parent link, k itself would have been added because of some other vertex. So, we can look at parent of k .

So, maybe parent of k , some vertex L . So, we go to L , and we look at parent of L , everything was eventually traced back to the starting vertex. So, that is what marked the first set of vertices visited. So, in this way, we will walk backward and find the reverse path, which we can then, of course, enumerate in the forward direction and be done with it.

(Refer Slide Time: 13:51)

BFS from vertex 7 with parent information

	Visited	Parent
0	True	4
1	True	0
2	True	0
3	True	4
4	True	7
5	True	7
6	True	5
7	True	
8	True	7
9	True	8

Path from 7 to 6 is
7-5-6

To explore queue

- Mark 7, add to queue
- Explore 7, visit {4,5,8}
- Explore 4, visit {0,3}
- Explore 5, visit {6}
- Explore 8, visit {9}
- Explore 0, visit {1,2}
- Explore 3
- Explore 6
- Explore 9
- Explore 1
- Explore 2

BFS from vertex 7 with parent information

	Visited	Parent
0	False	
1	False	
2	False	
3	False	
4	True	7
5	True	7
6	False	
7	True	
8	True	7
9	False	

To explore queue
4
5
8

- Mark 7, add to queue
- Explore 7, visit {4,5,8}

BFS from vertex 7 with parent information

	Visited	Parent
0	True	4
1	True	0
2	True	0
3	True	4
4	True	7
5	True	7
6	True	5
7	True	
8	True	7
9	True	8

Path from 7 to 2 is
7-4-0-2

To explore queue

- Mark 7, add to queue
- Explore 7, visit {4,5,8}
- Explore 4, visit {0,3}
- Explore 5, visit {6}
- Explore 8, visit {9}
- Explore 0, visit {1,2}
- Explore 3
- Explore 6
- Explore 9
- Explore 1
- Explore 2

So, this can be done along with breadth-first search with no extra cost, except that we record more information. So, as before we start by setting visited to false, we are again starting with 7. So, that we Just have the same familiar computation to do. But now we have this extra column called parent, which records the parent that is the vertex from which this merge vertex was marked as visited. So, as before, we start by marking 7 and adding it to the queue.

And then when we process 7, we add 4, 5, and 8 as marked, but we also set the parent of 4, 5, and 8 to be 7 to indicate that I came from here. So, let me draw an arrow like this. So, it says that I came from here, that is what this parent is saying that I came this way. So, whichever way you want to think of the arrow is going, it is pointing to the vertex which marked. Now, when I process the 4, similarly, it is going to mark 0, and 3, and for 0 and 3.

Now the parent becomes 4 because that is how they got mark 4 and 0 and 3 did not get marked by 7 they got mark by 4. Same way, when I explore 5, the parent of 6 becomes 5. Same way, when I explore 8, the parent of 9 becomes 8, because I got to 8 and 9 from 8. And finally, when I come to 0, then the parents of 1 and 2 becomes 0 because 0 marks them. Notice that 7 does not have a parent. And that is because we started the search at 7. So, 7 got visited, because we initiated the breadth-first search at 7 not because of some other vertex marked it.

So, except for the Source vertex, all the other visited vertices will have a parent node marked in there. And now we can recover this information. So, we come back to the end. And finally, after we have explored everything we have emptied, the queue and breadth-first search is over. Now, we can ask for instance, what is the path from 7 to 6. So, the path from 7 to 6, that breadth-first search discovered, well, I go to 6, and 6 says the parent of 6 is 5. So, 6 says I came from here. And 5 now says I came from 7.

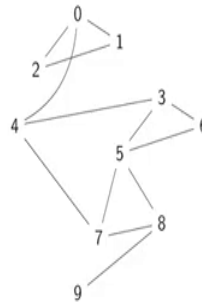
So, by following these parent links, and traceback this path from 6 to 7, and I read this path in reverse and I get 7, 5, 6. So, here is a longer path, what is the path from 7 to 2. So, 2 says I came from 0, 0 says I came from 4, 4 says i came from 7. So, therefore reading it backward, it is now 7, 4, 0 and then 2. So, in this way, by keeping this parent information as we are exploring the graph, we also record the path for every reachable vertex or a path for every reachable vertex from the source vertex.

(Refer Slide Time: 16:45)

Enhancing BFS to record distance



- BFS explores neighbours level by level
- By recording the level at which a vertex is visited, we get its distance from the source vertex
- Instead of `visited(j)`, maintain `level(j)`
- Initialize `level(j) = -1` for all j
- Set `level(i) = 0` for source vertex
- If we visit j from k , set `level(j)` to `level(k) + 1`
- `level(j)` is the length of the shortest path from the source vertex, in number of edges



So, how about the distance. So, we have explained that BFS explores neighbours level by level. So, in some sense, the level of a vertex indicates the earliest that I can get to that vertex from the source vertex. So, all the vertices which are level 1 could be reached directly in 1 edge from the source vertex. Anything which is at level 2 is reachable from level 1 but was not reachable directly. So, I need two edges to reach it, and so on. So, the level gives us some notion of distance from the source vertex. And can we do that? So, it turns out that we can modify our breadth-first search.

So, instead of just keeping this true false information for visited, we can replace it by this level information. So, every vertex which is reachable will have a level, which is 0 for the source vertex, and it has 1 or more for every other vertex that is reachable. So, since the minimum level is 0, for any reachable vertex, I can initialize the level to be minus 1.

So, any vertex whose level at the end remains minus 1 is as good as something whose visited value was false. That is, I never reached it. Otherwise, what I do is I set the level to be 0 for the source vertex, assuming that I am starting from i . And whenever we visit a vertex j from vertex k , we already have a level assigned to k .

So, we assign the level of j to be the level of k plus 1. So, it should be clear that because we are doing this level by level, the length of the shortest path, in terms of number of edges that we have to travel is given by the level. So, if I look at level of j , I know exactly how many edges I need to take to get from i to j , there might be longer ways, but there is no shorter way.

(Refer Slide Time: 18:35)

BFS from vertex 7 with parent and distance information

Level	Parent
0	-1
1	-1
2	-1
3	-1
4	1 ✓ 7
5	1 ✓ 7
6	-1
7	0
8	1 ✓ 7
9	-1

To explore queue
4 5 8

- Mark 7, add to queue
- Explore 7, visit {4,5,8}

Madhavan Mukund Breadth First Search Mathematics for Data Science

BFS from vertex 7 with parent and distance information

Level	Parent
0	-1
1	-1
2	-1
3	-1
4	-1
5	-1
6	-1
7	0
8	-1
9	-1

To explore queue
7

- Mark 7, add to queue

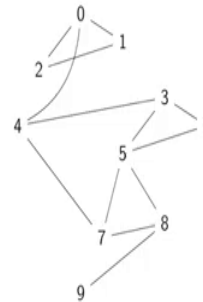
Madhavan Mukund Breadth First Search Mathematics for Data Science

BFS from vertex 7 with parent and distance information



Level	Parent
0	-1
1	-1
2	-1
3	-1
4	-1
5	-1
6	-1
7	-1
8	-1
9	-1

To explore queue



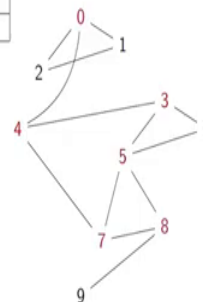
BFS from vertex 7 with parent and distance information



Level	Parent
0	2 ✓ 4
1	-1
2	-1
3	2 ✓ 4
4	1 7
5	1 7
6	-1
7	0
8	1 7
9	-1

To explore queue
5
8
0
3

- Mark 7, add to queue
- Explore 7, visit {4,5,8}
- Explore 4, visit {0,3}



So, here is that breadth-first search again. So, I have now replaced this column visited by this column level. And as we said before, we put all the levels to be initially minus 1. So, now we start with our favourite starting point 7. So, we start by setting this level to be 0. So, instead of setting it to be true, as visited, we set its level to be 0. Now, when we explore 7, its neighbours get to level 1. So, 4, 5, and 8 are all at level 1. Now, when we explore the neighbour of 4, I get to 0 and 3. So, these now have level 0032 because the level of 4 was already 1, so, it is 1 plus 1 is 2.

(Refer Slide Time: 19:17)

BFS from vertex 7 with parent and distance information

Level	Parent
0	2
1	-1
2	-1
3	2
4	1
5	1
6	2
7	0
8	1
9	-1

To explore queue							
8	0	3	6				

- Mark 7, add to queue
- Explore 7, visit {4,5,8}
- Explore 4, visit {0,3}
- Explore 5, visit {6}

Madhavan Mukund Breadth First Search Mathematics for Data Science

BFS from vertex 7 with parent and distance information

Level	Parent
0	2
1	-1
2	-1
3	2
4	1
5	1
6	2
7	0
8	1
9	2

To explore queue							
0	3	6	9				

- Mark 7, add to queue
- Explore 7, visit {4,5,8}
- Explore 4, visit {0,3}
- Explore 5, visit {6}
- Explore 8, visit {9}

Madhavan Mukund Breadth First Search Mathematics for Data Science 1 Week 10

Now if I explore 5, though I am exploring 5, after 4, the level of 5 was the same as the level of 4. So, what I can reach from 5 is also at level 2, it is not that it is level has increased because I am exploring it after 4. So, both 4 and 5 were added at the same time. So, they were both at level 1. I just happen to have happened to choose to process 4 before 5. So, 6 also becomes level 2.

Similarly, when I look at 8, 8 was also originally a level 1 vertex, so 9 which is reachable from it also becomes. So, at this point, if we look, we can see that the 2 vertices which have not yet been visited, are the ones which have level minus 1. So, that is, of course going to be fixed next. When I explore 0, so notice that explorer 0 means that 0 has level two. So, these two become 3, because in order to reach 1 and 2, I have to first go to 0, and 0 was already 2 edges

away. So, this gives us the breadth-first search. And as before, so, this is about paths, but as before, we could also record the, we have the parent vertices.

(Refer Slide Time: 20:23)

BFS from vertex 7 with parent and distance information

	Level	Parent
0	2	4
1	3	0
2	3	0
3	2	4
4	1	7
5	1	7
6	2	5
7	0	
8	1	7
9	2	8

To explore queue

- Mark 7, add to queue
- Explore 7, visit {4,5,8}
- Explore 4, visit {0,3}
- Explore 5, visit {6}
- Explore 8, visit {9}
- Explore 0, visit {1,2}
- Explore 3
- Explore 6
- Explore 9
- Explore 1
- Explore 2

Madhavan Mukund | Breadth First Search | Mathematics for Data Science

So, you can get the path. But we can also get the shortest distance. So, you can keep only the level only the parent both whatever you want.

(Refer Slide Time: 20:33)

Summary

- Breadth first search is a systematic strategy to explore a graph, level by level
- Record which vertices have been visited
- Maintain visited but unexplored vertices in a queue
- Add parent information to recover the path to each reachable vertex
- Maintain level information to record length of the shortest path, in terms of number of edges
- In general, edges are labelled with a **cost** (distance, time, ticket price, ...)
- Will look at **weighted graphs**, where shortest paths are in terms of cost, not number of edges

Madhavan Mukund | Breadth First Search | Mathematics for Data Science

So, to summarize breadth-first search is a systematic strategy to explore a graph level by level. So, remember that we said that broadly, the way we explore a graph is to systematically propagate these marks from a source vertex to the neighbours, to the neighbours, and so, on. But we need a way to do this. So, that we do not go around and around in cycles. So, we need a way to do it. So, that we terminate sensibly.

So, breadth-first search is one such strategy. So, what we do is we record the vertices which have been visited, and we maintain this queue of visited vertices, which are yet to be explored. So, exploration means explore the neighbours, then we saw that we can add parent information to recover the path. And we can maintain level information to record the distance, which is the shortest path in terms of number of edges. And what we will see is that in general, when we have a graph, we could record more information than just the edge.

For instance, if you had airline time graph, or a railway graph representing the route of railway network or an airline network, then typically with each edge, you would have associated with Some number, which we will abstractly call a cost. Now the cost could be a real cost, it could be the price of a ticket to go from the station to that station, or from this airport at that airport.

But the cost could also be a distance, it could also be how many kilometres this route travels. Or it could be time. For instance, if you have a train, the same distance could be covered in different times depending on the quality of the tracks and the number of stops in between.

So, in general, to go from one point to another point, you have to traverse real-time or pay Some real money, or spend Some distance traveling, say, supposing it is a road network, then it will give you a measure of how many liters of petrol you would expect to consume complete this distance.

So, in these kinds of graphs, which are called waited graphs, the shortest paths are no longer just in terms of the number of edges, we could have a short edge, a single edge, which has a very high cost. It could be that a direct flight costs much more than a flight which goes by an intermediate airport because the airline is trying to encourage people to take these flights to unpopular airports in order to fill the planes. So, it is not always the shortest number of edges is also the shortest path. So, we will explore this separately when we look at waited graphs.