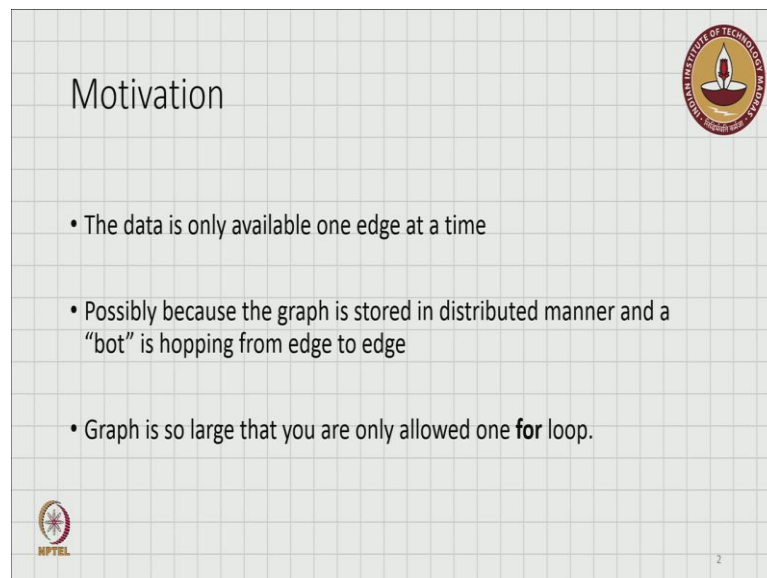


Algorithms for Big Data
Prof. John Ebenezer Augustine
Department of Computer science and Engineering
Indian Institute of Technology, Madras

Week - 07
Lecture - 44
Solving Graph Problems in the Streaming Model



Today we are going to talk about graph problems, but we are going to talk about graph problems in the Streaming Model and you may be wondering how this can be done. So, let us dive into it to see how it works.

(Refer Slide Time: 00:32)



Motivation

- The data is only available one edge at a time
- Possibly because the graph is stored in distributed manner and a “bot” is hopping from edge to edge
- Graph is so large that you are only allowed one **for** loop.

2

But before we get started, let us motivate the context a little bit. Now, in regular graph problems that you might have studied in typical algorithms courses, you expect the entire graph to be available to you, but that is not always possible when the graph is very very large. So, this is after all a course in algorithms for big data. So, we want to consider our context where the graph is so large that it would not fit into your one computer.

And as a result the entire graph is not accessible to you, but rather its only accessible one edge at a time. And this could be for a variety of reasons - one common reason is that the graph is actually not stored in one computer, but it is stored across multiple computers,



multiple servers in a large data center. So, the only way you can explore the graph is to actually have a bot which is basically a little program that hops from one server to another server following the edges. As far as from the bot perspective, it only sees one edge at a time. So, this is one perspective of how we might be able to or how we might encounter a situation where we only see one edge at the time.

And the other possibility is that the graph is so large that you cannot really load it up into main memory in one shot, but you just do a single for loop over the entire set of edges, and you get the edges in some order and your perform your operations on that for loop. So, you only get one pass through the entire set of edges. So, these are just a few motivating contexts where such a streaming algorithms might be useful. So, let us study this a bit formally.

(Refer Slide Time: 02:49)

Model

- There is a graph $G = (V, E)$ that is possibly weighted (when explicitly mentioned).
 - # of nodes is n and # of edges is m . $O(n^2)$
- Access to the graph is through a stream of edges one at a time.
- Space: $O(n \text{ poly}(\log n))$. (aka semi-streaming model.)
- Variants: Edges can also be deleted.

4

Let us begin with a clear statement of what the model is the computational model. Our real input is a graph G with vertex set V and (Refer Time: 03:02) set E , we use n to denote the number of nodes and m to denote the number of edges. And the edges might be weighted or unweighted. For most of this lecture we are going to assume that the graph is unweighted, but when weights are relevant we will mention them.

Now, this graph is not directly accessible to us, but rather the access to the graph is through a stream of edges one at a time. So, the access to the graph is only through a stream of edges, and we see one edge at a time and after you have seen that edge we have to let go of that edge, and the next edge shows up with a process that edge and then we let go of that edge, and then the subsequent edge shows up and so on.

And as we run through the edges we allow some space, but we are limited to some space that is of the order of n times the polynomial n , \log of n . So, of course, n is the number of vertices and keep in mind that the actual graph, the size of the graph is actually could be dominated by m . So, that can be as high as O of n square.

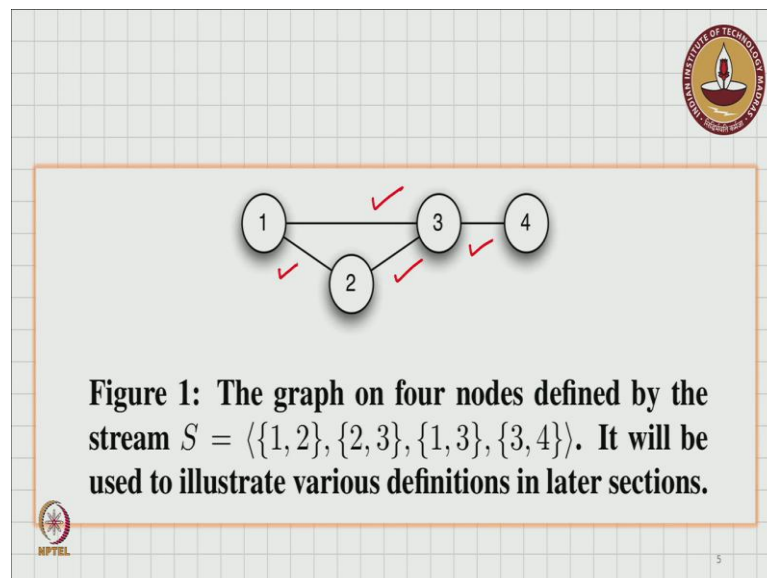
Now, we are not allowed to store, if we are allowed space of O of n square we can simply store all the edges and then let all the edges go through store all of them and then process all the edges in one shot and we will basically be able to store the entire graph. But that is not allowed here. Only allowed to store n over poly \log of n number of bits, which means that we have to be a little bit careful and this is slight relaxation of the usual stringing model where the actual memory that we can store is actually much smaller, say typically only $\log n$ or may be \log squared n or something like that. But with such small memory graph problems tend to become often times impossible to solve and so we give ourselves a little bit more memory and this as a result, this model is called the semi streaming model.

So, now we can think of variance, in this variant we have getting one edge at a time. So, we can think of it as we are starting with an understanding of the graph being just the empty graph and then as an edge come, we can think of it as getting more and more information about the edges. But it is also possible in variations that you can think of where edges are added, but they could also be deleted and that complicates things, but we will not be considering them in our lecture today.

But you can easily imagine that these are possibilities. Again other varieties include changes in the edge weights and so on and so forth, that can be streamed into the system. But we will ignore all of those variations for now; we are going to stick with this variation where we receive one edge at a time. So, we are going to stick to the basic

model and of all are (Refer Time: 06:38) times simplicity we are going to assume that the vertex set is basically labeled 1 2 n. So, this will mean that we have n vertices and we can assume, they have an integral label from 1 to n.

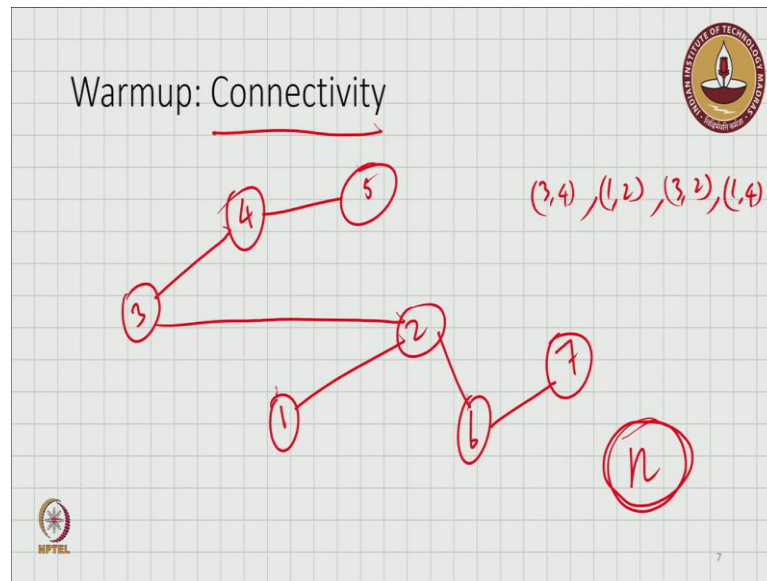
(Refer Slide Time: 06:57)



So, here is an example, just quick example of a graph with four nodes. So, an example input stream will be first that we get the edge 1 comma 2. So, we get this edge and then we get the edge 2 comma 3 which means we get this edge and then, 1 comma 3, then 3 comma 4. But there is no reason why we should get it in this particular order, we could have also got it in completely different order.

For the example the first edge could have been 3 comma 4, then the second edge could have been 1 comma 2, then the third edge could have been 1 comma 3 and then 2 comma 3 at the very end. So, we are not making any assumption about the order in which we are going to receive the edges, we are going to receive the edges one after another and with that we have to answer some questions.

(Refer Slide Time: 08:04)



Without further do, let us start with a very basic question of that of connectivity. So, we have this graph and we are getting one edge at a time and we want to know whether the graph is connected or not. So, how do we do that? Let us say that the first edge we get is say 3 comma 4, we include that. And let us say we keep the next edge that we get is 1 comma 2 at the moment this is all the information we have and we keep collecting such edges. So, let us say the next edge that we get is 3 comma 2, so we build our graph. And then the fourth edge that we get is 1 comma 4, now the question is what do we do with the edge 1 comma 4.

So, just to remind ourselves, we got the edge 3 comma 4 first, then we got the edge 1 comma 2, then we got the edge 3 comma 2 and now we have got in the edge 1 comma 4. But notice that the whole point of this current problem is to check for connectivity and given the information that we have, already know that 1 and 4 are connected, and with that being the case there is no real virtue in including this edge 1 to 4 in our data structures. So, we simply can avoid this edge.

And then let us say we get an edge say 4 comma 5, we add that 6 comma 7 and let us say this tree stops at this point. Then we know that the graph is actually not connected, but on the other hand if we were following this we got an edge 2 comma 6 then we know


that the graph is connected.

So, as the algorithm progresses we are maintaining a forest and at termination time if it is a forest, but not a tree then we know that the graph is not connected. But if it is a tree at termination time, we know that the graph is indeed connected. So, this is a simple example of a streaming algorithm that reports to us whether the graph is connected or not.

Let me add one little point about the model, we know we have n vertices we are being assume in general that this n is known to us in advance. This is not a very important assumption sometimes we can get rid of it, but for simplicity for our purposes we are going to assume that n is always known. So, in this case if n is always known to us, the moment we create a tree on n vertices even we can terminate at that point itself. So, now, let us ask slightly more advanced question.

(Refer Slide Time: 11:37)

Spanners



DEFINITION 1 (SPANNER). Given a graph G , we say that a subgraph H is an α -spanner for G if for all $u, v \in V$,

$$d_G(u, v) \leq d_H(u, v) \leq \alpha \cdot d_G(u, v).$$

where $d_G(\cdot, \cdot)$ and $d_H(\cdot, \cdot)$ are lengths of the shortest paths in G and H respectively.

NPTEL

8

For that, let us define a notion called as Spanners. What is a Spanner? Well, let us say you are doing a graph G , what you want is a smaller sub graph that sort of mimics the connectivity properties of the distance properties of the original graph. So, the original graph could be a much more complicated graph and what we want is a sparser graph H

and typically a sub graph H , and the distance in the sub graph H should be an approximation of the distances in the original graph and this is very useful because if the original graph is too complicated, but if we construct a spanner that has fewer number of edges it will occupy less space. So, this one way of taking a large data set and approximating it by a smaller data set that gives you reasonably good approximate answers to whatever distant queries you might have.


Let us be little bit more formal now. Given a graph G we say that a sub graph H is an α spanner, if your α is the approximation parameter we say that H is an α spanner for the graph G if for every pair of vertices. So now you have a graph now you take any, this is let us say G any pair of vertices u and v there is a distance between u and v in that graph G . So, that is your $d_G(u, v)$. Now multiply that by α . So, let us give ourselves some example, now let us say $d_G(u, v)$ is 100 and let us say α is 1.5. So, let us assume α equal to 1.5 as an example. So, this is 100 and this is going to be 150. So, this gives us a range to work with.

And from G we want to construct a sub graph which is H , and we say that H is a sub graph if the distance between, I am sorry, H is an α spanner if the distance between u and v in H is sandwiched between the original distance between u and v and α times the original distance between u and v . This means that, well this is going to be, this inequality is going to be easy to attain, but what might happen what needs to be ensured is that while making with sub graph smaller we have to ensure that this inequality is also respected.

In other words when you reduce the number of edges in the graph H you do not want to blow up the distance beyond a α times $d_G(u, v)$, and this must be true remember for all pairs of vertices u and v . So, this is the definition of a spanner and why would a spanner be useful only if H has a far fewer number of edges than G . If that is the case then H will occupy a lot less space and we will be a very good approximation of graph sheet.

(Refer Slide Time: 15:55)


unweighted



Algorithm 1: Spanner

- 1 $H \leftarrow \emptyset;$
- 2 **for each** $\{u, v\} \in S$ **do**
- 3 **if** $d_H(u, v) > \alpha$ **then** $H \leftarrow H \cup \{\{u, v\}\};$
- 4 **return** H

Why is this algorithm correct?



9

So, let us look at an algorithm to construct a spanner and we are going to look at it from the streaming models point of view, is actually very, very simple. You start with the spanner being the empty set, remember this is our spanner H and we stream through the edges, so for each edge $u v$ now here is the condition that we need to check. Now we are going to maintaining a graph H , initially this H is going to be empty and when we get an edge $u v$ we ask ourselves what is the distance between; we are maintaining a current H and what is we are asking what is the distance between u and v in this graph H that we are maintaining.


And if that H , and just to remind ourselves where thinking often this is an unweighted graph; now if the actual distance between u and v , that is the let us say the shortest path between u and v in H . If that distance is greater than α then; that means, by including this H you are going to be ensuring that you are not over shooting that α approximation, because now this means that you have to add this edge $u v$ if this condition were to hold, and which is exactly what we are doing.

Now, if this condition was to hold and we did not add $u v$ then we are immediately violating the α spanner condition. So, that is why we must add $u v$ and as it turns out from this what we realize here is that just by ensuring that $u v$ is added in this, when this

condition is encountered then the algorithm overall is correct. So, for each edge $u v$ it checks if the distance in the spanner that we are maintaining so far between the vertices u and v is greater than α . If it is greater than α then we must include the edge $u v$ in the spanner otherwise we do not have to execute this inclusion line over here and at the end we just return H .

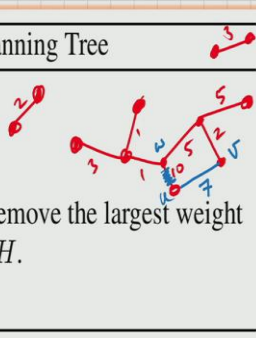
Now, why is this algorithm correct? So, this algorithm is correct because now if you have a path this spanner condition should hold for any pair of vertices, right. So, now, let us say you have a path from a to b to c to d . Now in the spanner you may not have added all of these edges. So, let us say you have a and b , I mean you have $a b$, but you do not include $b c$. But why would we not have included $b c$ only, because there were some other path from b to c that was of distance at most α and as a result the blow up you can go from a to d by just taking these d towards and that will not blow up the distance by more than a factor α . So, that is the reason why this algorithm will indeed be an α spanner. And of course, now the interesting part of be to ensure that the size of the spanner is within reasonable limits. Will skip that for our purposes now; that the idea of how to construct a spanner is very important and I hope that part is clear to you.

(Refer Slide Time: 20:16)

Minimum Spanning Tree (weighted Graph) 

Algorithm 2: Minimum Spanning Tree

- 1 $H \leftarrow \emptyset$;
- 2 for each $\{u, v\} \in S$ do
- 3 $H \leftarrow H \cup \{(u, v)\}$;
- 4 If H includes a cycle, remove the largest weight edge in the cycle from H .
- 5 return H



11

Now, let us look at a weighted graph. So, now, we are looking at a weighted graph. Now

we want to compute, let us say the minimum spanning tree, how do we do that? Well this again is a very simple algorithm, we of course, start with the empty minimum spanning tree and we look at one edge at a time, and whenever we encounter an edge $u v$ we ask if; first we add the edge $u v$ to H and ask if that created a cycle.

If that created cycle then we need to be a bit careful now. So, whenever a cycle is created and we want to maintain a minimum spanning tree what we should do is in that cycle we can safely remove the largest weight H . So, let us say we created, we have currently a graph that looks like we are maintaining; the invariable we are maintaining is at the start of each for loop the algorithm H will have a forest, will represent a forest and the forest will actually be weighted, let us say this is 2 3 1 5 1 10 2 5 and let us say these are the weights.

And now let us say we add this edge over here, and that edge, this is let us say u and this is v and so we receive the edge $u v$ and we have added the edge $u v$ and it happens to have a weight of 7.

Now what do we do? We have found a cycle in which case now in this cycle we have to remove the largest weight H and that is going to be let us say this edge is this vertex is w , that will be the edge $u w$ and that edge $u w$ must be removed from H . By repeating this procedure for every edge $u v$, we will be maintaining a spanning forest and if at the end the graph is connected that H will actually be representing the minimum spanning tree.