



IIT Madras

ONLINE DEGREE

Computational Thinking
Professor Madhavan Mukund
Department of Computer Science
Chennai Mathematical Institute
Professor G Venkatesh
Indian Institute of Technology, Madras
Pseudocode for edge labelled graph

(Refer Slide Time: 00:14)

Pseudocode: Edge Labelled Graphs

Diagram illustrating a directed graph with nodes v , w , and c . Edges are labeled a (from v to w), b (from w to v), and c (from v to c).

So, we have seen that a graph consists of a set of nodes connected by a set of edges, so when we draw a graph we typically draw them as nodes like this and then we draw edges with arrows indicating what is connected to what. Now we can do one more step of information gathering by putting some labels which carry meanings on these edges. So what we are going to look at in this session is what we call an Edge Labelled Graph.

(Refer Slide Time: 00:46)

Trains

- Train dataset — information about trains and stations
 - Each train is a route list of stations
 - Each station is a route list of trains passing through
- Compute pairs of stations connected by a direct train
- Represent start and end station of trains in a nested dictionary
 - `trains[t][start], trains[t][end]`

Sl. No.	Station Name	Arr.	Dep.	Days
1	Kolkata	—	18:31	1
208	Dhanbad	21:52	21:58	1
687	Vasrasi	02:30	02:30	2
1014	Kanpur	06:05	06:20	2
1483	New Delhi	11:31	11:45	2
1629	Loharu	14:25	14:28	2
1678	Sadulpur	15:09	15:08	2
1738	Churu	16:06	16:08	2
1779	Patangmah	17:30	17:32	2
1844	Sri Dungargarh	17:47	17:48	2
1918	Bikaner	18:15	—	2

Train No.	Arr.	Dep.	Days
12261	04:10	04:15	M Tu Th F
12221	04:10	04:15	Tu Sa
02396	05:15	05:20	Tu Sa
12270	05:15	05:20	W Tu
12389	07:20	—	M Tu Th F Sa Su
12390	—	20:40	M Tu Th F Sa Su
12269	21:00	21:06	MTF
02385	21:00	21:06	Tu Sa
12222	23:15	23:20	Th Sa
12262	23:15	23:20	M Tu W F

Pseudocode: Edge Labelled Graphs 2/7

IIT Madras ONLINE DEGREE

So, let us go back to the train's data set. So remember that we have these train cards and we have this station cards and what we said was we would try to represent in a graph all pairs of stations which are connected by a direct train. So for this what we said was that a direct train would connect the starting point and the ending point. We are not going to connect, for example Kanpur to Churu. We are not going to call this a direct connection.

We are only going to allow Kolkatta to Bikaner to be a direct connection for such a train. So for each train we set up a dictionary entry where the keys is a train, train number and then there are two special fields, key start and end which indicate the starting and the ending. So in this particular train 12259 the start station is Kolkata and the end station is Bikaner.

(Refer Slide Time: 0:01:35)

Adding information to direct route graph

- Stations A and B such that a train starts at A and ends at B
- Create a matrix to record direct routes
 - Compile the list of stations from `trains`
 - Map stations to row, column indices
 - Populate the matrix

```

Procedure DirectRoutes(trains)
  stations = {}
  foreach t in keys(trains) {
    stations[trains[t][start]] = True
    stations[trains[t][end]] = True
  }
  n = length(keys(stations))
  direct = CreateMatrix(n,n)
  stnindex = {}
  i = 0
  foreach s in keys(stations) {
    stnindex[s] = i
    i = i+1
  }
  foreach t in keys(trains){
    i = stnindex[trains[t][start]]
    j = stnindex[trains[t][end]]
    direct[i][j] = 1
  }
  return(direct)
End DirectRoutes

```

Kolkata 7 → 12259 → 12 Bikaner s2

Pseudocode: Edge Labelled Graphs 3/7

IIT Madras ONLINE DEGREE

So, from this dictionary, then we wrote this procedure called direct routes which will first compute, if you remember the set of stations which are actually present as starting stations and ending stations because we do not know this in advance. This is implicitly given in the train dictionary. So we pick it up from the train's dictionary. Then we create a matrix to represent the graph that we are going to construct. That is called direct.

But for that we need to first map all our stations from names to positions 0 to n minus 1. So we did that and then we did this simple loop. We just looked at every possible pair of stations. No, sorry we looked at every possible train and picked up the index of the starting station and the ending station and created a direct i, j entry for this.

So, this is a simple graph which takes the numbers of the station. So 0 to n minus 1 as nodes and draws an edge from i to j, if there is a direct train from i to j. But once we have drawn this direct edge from i to j we do not know which train it was and if there are more than 1 train. So what if we want to record this information, right?

So, we want to record more than just the facts. So we want to suppose Howrah is, Kolkata is say station 7 and maybe Bikaner is station 52, then we would just have drawn this edge from 7 to 52. But I want to say that this is actually an edge which is serviced by the train 12259. So how do I put this information into my graph?

(Refer Slide Time: 03:07)


Adding information to direct route graph

- Stations A and B such that a train starts at A and ends at B
- Create a matrix to record direct routes
 - Compile the list of stations from `trains`
 - Map stations to row, column indices
 - Populate the matrix
- Keep track of trains connecting stations
 - Each entry in the matrix is now a dictionary
 - Initially empty dictionary — no direct connection
 - Add a key for each train connecting a pair of stations

```


Procedure LabelledDirectRoutes(trains)
:
foreach r rows(direct) {
  foreach c columns(direct) {
    direct[i][j] = {}
  }
}
foreach t in keys(trains){
  i = stnindex[trains[t][start]]
  j = stnindex[trains[t][end]]
  direct[i][j][t] = True
}
return(direct)
End DirectRoutes2
          
```

- Information about trains is recorded as a label on the edge
- Edge-labelled graph



Pseudocode: Edge Labelled Graphs

3/7



So, what we can do is instead of just keeping 01 as my entries in the direct matrix, I can make the entries actually dictionaries. So what I do is that initially, I set each pair i j to have an empty dictionary. So this means that there is no route that I know of between station i and

station j . Now, whenever we discover a route, instead of just saying direct $i j$ is equal to 1 as we did before, we change the 0 to a 1, what we do is we take this dictionary that we have for that position and we add a key for the train that we have just discovered.

So, notice is therefore the direct $i j$ could have more than 1 train. There could be for example between Mumbai and Delhi, there could be multiple trains. So what will end up? We will end up doing after we scan all the trains are that for every train that runs between Mumbai and Delhi. If Mumbai is i and Delhi is j , there will be a key for that train. So there will be multiple keys attached to that entry.

If there is no connection between a pair of cities, then there will be no train and the dictionary will remain empty. So this is what we call an Edge Labelled Graph. So we have taken the edges between the cities and we have put some information on this edge which gives us more information than just the fact that there is route. In this particular case we have recorded all the trains that create this route.

There could be 1, there could be many and of course, if there is no train then we will have an empty dictionary. So this notion of an Edge Labelled Graph is a very useful thing and it is very often used in conjunction with graphs to do computations.

(Refer Slide Time: 04:49)

Distances between stations

- For each direct train record the distance it travels
 - Add an extra key, `trains[t][distance]`
- Compute the shortest distance by direct trains
 - Trains may take different routes between the same pair of stations
- Graph `directdist`
 - Edge label `directdist[i][j]` is the shortest direct distance between i and j

Pseudocode: Edge Labelled Graphs 4/7

IIT Madras
ONLINE DEGREE

Trains

- Train dataset — information about trains and stations

- Each train is a route list of stations
- Each station is a route list of trains passing through

- Compute pairs of stations connected by a direct train

- Represent start and end station of trains in a nested dictionary

```
trains[t][start], trains[t][end]
```

Train	Station Name	Arr	Dep	Day
12259	Kolkata	—	18:31	1
206	Dhanbad	21:52	21:58	1
687	Varanasi	02:20	02:30	2
1014	Kanpur	06:25	06:30	2
1463	New Delhi	11:31	11:45	2
1629	Loharu	14:23	14:28	2
1679	Sadulpur	15:03	15:08	2
1738	Churu	16:06	16:08	2
1779	Ratangarh	17:00	17:02	2
1844	Sri Dungargarh	17:47	17:48	2
1916	Bikaner	19:15	—	2

Station	Train No	Arr	Dep	Day
Nagpur	12281	04:10	04:15	M T W T F
	12221	04:10	04:15	Tu Su
	02196	05:15	05:20	Tu Su
	12175	05:15	05:20	W Su
	12289	07:20	—	M T W T F Su Su
	12290	—	20:40	M T W T F Su Su
	12268	21:00	21:06	M F
	02285	21:00	21:06	Th Su
	12222	23:15	23:20	Th Su
	12262	23:15	23:20	M T W F



Pseudocode: Edge Labelled Graphs

2/7



So, here is 1 natural computation that we might want to do with our own train data set. Supposing we need to know not just whether or not there is a train from A to B, a direct train but also how much distance it travels. Now if you remember this information is there on that card because there is a distance column here. So this tells us for example this card that is 1916 km from Kolkata to Bikaner.

So, in a similar way, I might want to know the distance between every pair of cities which is directly connected by a train. So this means that I would have to record that distance first of all in my train's dictionary. So I have my train's dictionary for each train t. I had a start station and end station and now I will also keep a third key which is a distance travelled from start to end by this train.

Now, I need to compute this information for every pair of stations. So I have an Edge Labelled Graph where in the previous one I had created edge labels consisting of the train numbers which connect A to B. But now I want to know distance from A to B. So this is an Edge Labelled Graph in which the Edge Label is a number.

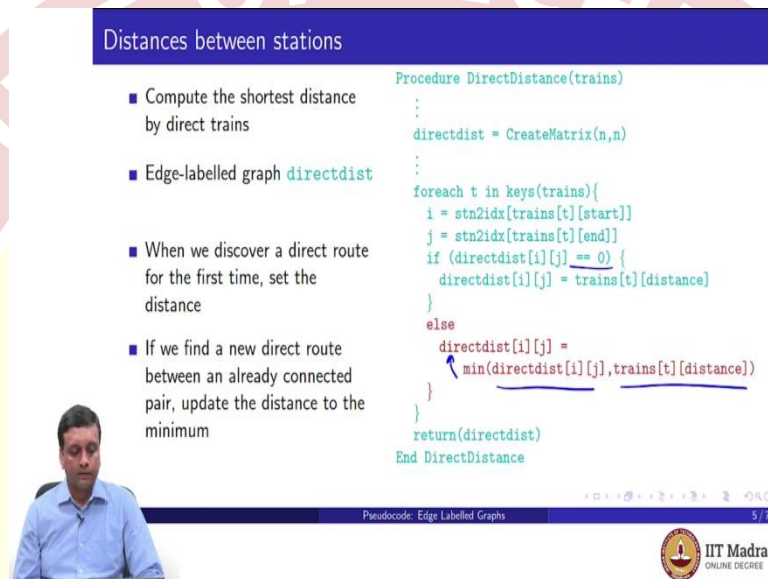
So trains might now, you might ask, I mean if it is connected then surely there is only 1 distance but if you look at the data set actually there are for long distance trains like say from Howrah to or Kolkata to Delhi. There are some trains which go by one route and some trains which go by another route. So even though the starting station and the ending station are the same, it is possible that these trains follow different routes and travel different distances.

So, as we go along finding out all the trains we might need to update this distance as we go along. So we are going to represent this in this Edge Labelled Graph which I have called

directdist. So the Edge Labelled Directdist is going to be a number and directdist i j is going to indicate the shortest direct distance between i and j among all the trains and if there is no train, if i and j are not connected by a direct train then I am going to just represent it by 0.

So, we assume that no pair of stations can be connected by 0 distance. So, therefore if I see a 0 there is no edge, but if I see a non-zero entry is not going to be just 1. It is going to be the actual distance in kilometres by the shortest route which takes me from i to j .

(Refer Slide Time: 07:10)



Distances between stations

- Compute the shortest distance by direct trains
- Edge-labelled graph **directdist**
- When we discover a direct route for the first time, set the distance
- If we find a new direct route between an already connected pair, update the distance to the minimum

```

Procedure DirectDistance(trains)
:
directdist = CreateMatrix(n,n)
:
foreach t in keys(trains){
i = stn2idx[trains[t][start]]
j = stn2idx[trains[t][end]]
if (directdist[i][j] == 0) {
directdist[i][j] = trains[t][distance]
}
else
directdist[i][j] =
min(directdist[i][j], trains[t][distance])
}
return(directdist)
End DirectDistance
  
```

Pseudocode: Edge Labelled Graphs 5/7

IIT Madras
ONLINE DEGREE

So, here is how we do this. We first create, so remember we do all that stuff to figure out in the direct edge case to figure out how many stations there are and how to map them and all the. So let us assume we have done all that. So now we have got n which is the number of stations in our network. So we create this empty matrix which has all 0 entries of n by n .

Then we do this index mapping so that we get an index for every station and now we are in business to compute the distances. So we go through every train. We take the starting point and the ending point and then we see if this is the first time we have discovered this distance. If it is the first time that we have discovered this distance then we just take the current train's distance entry and copy it into Directdist i j .

So, you remember that if the value of the distance is 0, that means that so far these 2 stations are assumed to be disconnected. So the very first time I find a connection, I just take whatever distance I get and I copy it. On the other hand, if the distance is not 0 then this train connects a pair of stations which I have already found before as being connected.

So, there is already a distance recorded for this. So I want to find the shortest distance. So I take the current value that I have $\text{directdist } i \ j$ which is a current. Remember it is not 0 because I have come to the else part. So I have come to the else part which means that it is not 0. So there is some existing route for which I have computed this distance.

So, I take that distance that I have already computed, I take the distance of the new train that I am examining now. I take the minimum of the 2 and replace it back into $\text{directdist } i \ j$. So this is the standard minimum calculation that we have done it for marks, everything. When I see a new mark, I keep the minimum that I have got so far, if the new mark is smaller than the minimum that I have got so far, I replace the minimum with the new mark.

This is exactly the same thing except now we are doing it in a graph format. So for each $i \ j$ which is flagged by the current train, I replace its distance by the minimum of the current distance that I already knew and the distance of the train that I am looking at.


(Refer Slide Time: 09:27)

One hop distance

- We start with the matrix of direct distances
- Initialize one hop distance to direct distance
- Each time we discover a one hop route, update the one hop distance
- Iterate this to find length of the shortest path between each pair of stations
 - Modify the transitive closure calculation to record minimum distance path


```

Procedure OneHopDistance(directdist)
  n = length(keys(directdist))
  onehopdist = CreateMatrix(n,n)
  foreach i in rows(directdist) {
    foreach j in columns(directdist) {
      onehopdist[i][j] = directdist[i][j]
      foreach k in columns(directdist) {
        if (directdist[i][k] > 0 and
            directdist[k][j] > 0) {
          newdist = directdist[i][k]
                      + directdist[k][j]
          if (onehopdist[i][j] > 0){
            onehopdist[i][j] =
              min(newdist, onehopdist[i][j])
          }
          else{
            onehopdist[i][j] = newdist
          }
        }
      }
    }
  }
  return(onehopdist)
End OneHopDistance
            
```



Pseudocode: Edge Labelled Graphs

6/7



IIT Madras
ONLINE DEGREE

So, we can do this for direct distances which we just did. We can also do this because we just took the train's dictionary and directly computed the 1 edge distance you know, without changing trains. Now of course I can do this for hops as well. So we start with this matrix of direct distances that we have already computed in the previous procedure and now as before we will say that okay, if I want to go with at most 1 hop, then if I can go directly then that is my minimum distance.

If I cannot go at all it will be 0. So I initialise the 1 hop distance from i to j to be the same as a direct distance from i to j . Then after that we do our usual thing, we check if there is an

intermediate k , now remember that in this graph the value is not 01. The value is 0 or not 0. So if there is an intermediate k for which the distance is bigger than 0 from i and from k to j is also bigger than 0 that means that I have a direct train from i to k and a direct train from k to j and not only that these 2 distances are the shortest distances that I know of between i and k and k to j respectively.

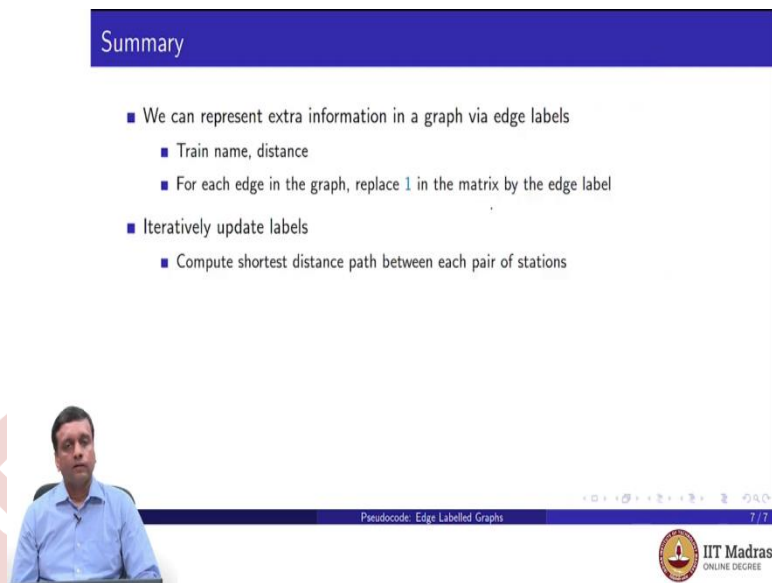
So, if I now go from i to j via k then the new distance will be the sum of these 2 distances because I have to go from i to k and add the distance from k to j . So I will call this a new distance. So I have discovered a new route from i to j . As before now what I need to do is to check whether this needs to be updated or not. So if the distance that I already have is not 0 then I will take the minimum distance that I already have and the new distance I have computed.

If it is 0 I will just update this to be the new distance. So this exactly the same thing that we did before except that we started off by starting with the direct distance and initialising 1 hop distance to be that and then for every k that we see, intermediate station. So exactly remember, we did this 1 hop connectivity. We said, for every k if I can go from i to k and k to j then I will go from i to j .

So, here because we have to compute the distance for every k we will say, let me take the distance from i to k , the distance from k to j and then take the minimum of that new sum and the sum that I already know and if I do not have it then I will add a new entry. So you can imagine now that I can take the 1 hop distance and the direct distance and I can get the 2 hop distance. I can take the minimum of the 1 hop plus the direct.

Then I can take the minimum of 2 hop direct. So just as we did this transitive closure to find out the pairs of cities which are connected by longer and longer paths I can find out the distance between the trains which are connected by longer and longer paths. So the transitive closure calculation that we did for just the 0 1 k , either there is a path or there is no path, we can replace it by this more elaborate minimum calculation to compute the shortest distance along any of these paths.

(Refer Slide Time: 12:26)



Summary

- We can represent extra information in a graph via edge labels
 - Train name, distance
 - For each edge in the graph, replace 1 in the matrix by the edge label
- Iteratively update labels
 - Compute shortest distance path between each pair of stations

Pseudocode: Edge Labelled Graphs 7/7

IIT Madras
ONLINE DEGREE

So, to summarise what we have done is, shown that by keeping extra information as labels on the edges we can make our graph contain more useful data. So in the first case we said that we could record for every train which connects every pair of stations connected by a train not just the fact that there is direct connection but also label it with every train which actually serves these pair of cities.

Then we said that you can also take this distance and now record the minimum distance between a pair of cities and then iterate it. So instead of just keeping 01 we keep a number and iteratively we can keep updating this number to get the minimum direct edge and then by doing this transitive closure calculation we can keep the minimum of all the paths that we discover by doing this transitive closure and in the process we will find the shortest path connecting any two pair of, any pair of cities which are actually connected in the train network.