# IIT Madras

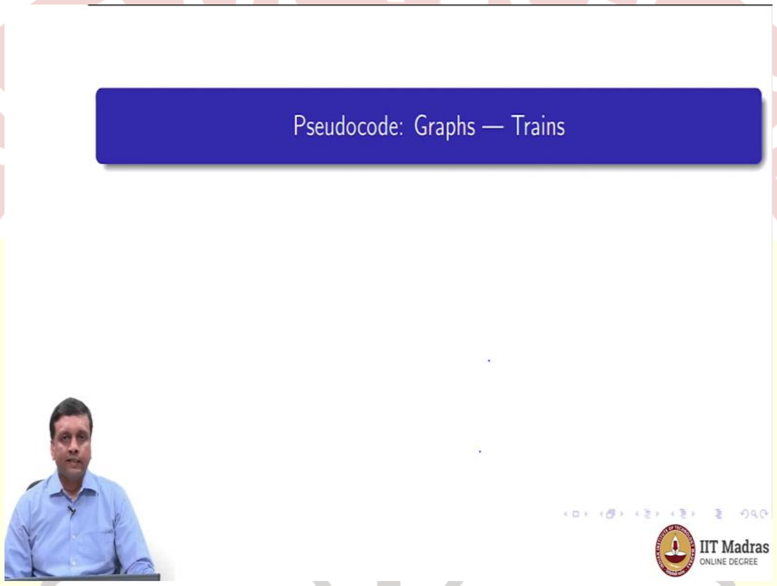
ONLINE DEGREE

# Computational Thinking

**Professor Madhavan Mukund**
**Department of Computer Science**
**Chennai Mathematical Institute**
**Professor G. Venkatesh**
**Indian Institute of Technology, Madras**

## Pseudocodes for finding a route between two stations using no hops, one hop, two hops and n hops

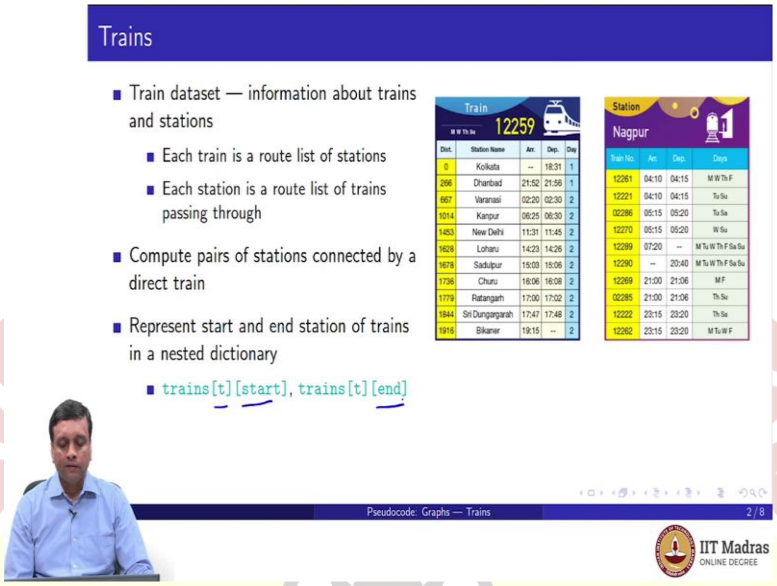
(Refer Slide Time: 00:14)



Pseudocode: Graphs — Trains

So, we saw how to use graphs to discover interesting things about the student scores database. Now, let us look at how to discover things about the train's data set which we have not explored so far in our Pseudocode.

(Refer Slide Time: 00:29)



## 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 12259
M W Th Su

| Dist. | Station Name | Arr. | Dep. | Day |
|---|---|---|---|---|
| 0 | Kolkata | -- | 18:31 | 1 |
| 266 | Dhanbad | 21:52 | 21:56 | 1 |
| 667 | Varanasi | 02:20 | 02:30 | 2 |
| 1014 | Kanpur | 06:25 | 06:30 | 2 |
| 1453 | New Delhi | 11:31 | 11:45 | 2 |
| 1628 | Loharu | 14:23 | 14:26 | 2 |
| 1678 | Sadulpur | 15:03 | 15:06 | 2 |
| 1736 | Churu | 16:06 | 16:08 | 2 |
| 1779 | Ratangarh | 17:00 | 17:02 | 2 |
| 1844 | Sri Dungargarah | 17:47 | 17:48 | 2 |
| 1916 | Bikaner | 19:15 | -- | 2 |

Station
Nagpur

| Train No. | Arr. | Dep. | Days |
|---|---|---|---|
| 12261 | 04:10 | 04:15 | M W Th F |
| 12221 | 04:10 | 04:15 | Tu Su |
| 02286 | 05:15 | 05:20 | Tu Sa |
| 12270 | 05:15 | 05:20 | W Su |
| 12289 | 07:20 | -- | M Tu W Th F Sa Su |
| 12290 | -- | 20:40 | M Tu W Th F Sa Su |
| 12269 | 21:00 | 21:06 | M F |
| 02285 | 21:00 | 21:06 | Th Su |
| 12222 | 23:15 | 23:20 | Th Sa |
| 12262 | 23:15 | 23:20 | M Tu W F |

Pseudocode: Graphs — Trains 2 / 8

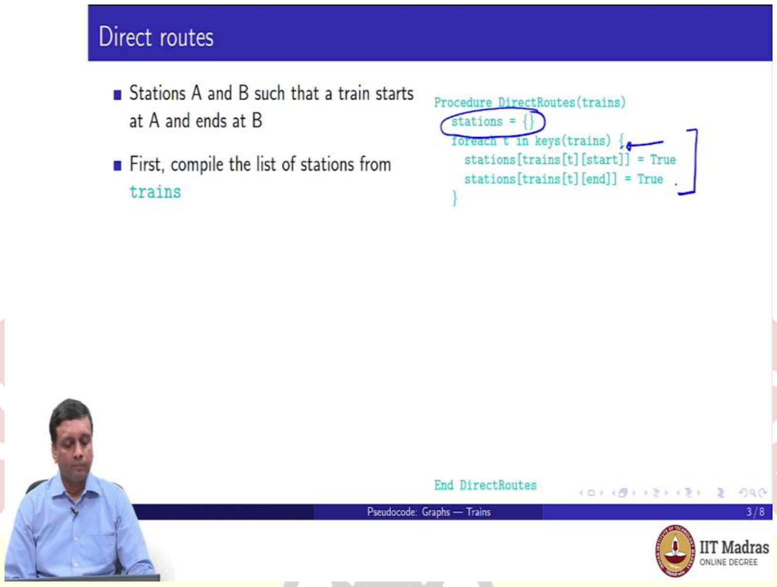
IIT Madras
ONLINE DEGREE

So, remember that in the train's data set we have two kinds of cards, we have a card describing trains, so each train has a train number and it has a list of sequence that it goes through from an starting station to an ending station. And similarly for each station we have a card and this card lists out all the trains passing through that station, so it gives the train number, the arrival time, the departure time and the days of the week on which this train runs.

So, from this we wanted to look at interesting information about which trains take us from where to where, which pairs of cities are connected by trains and so on. So, let us start with the most basic problem, we want to compute all pairs of stations that are connected by a direct train. Now, to make our problem a little easier we are going to think of a direct train as one that starts at the first station and ends at the last station.

So, we are not looking at trains which are connected, which are station which are connected by trains through intermediate stops. So, for each train, we will construct a dictionary entry which has the train number as a key and it will have two entries for that train number start and end which is the station at which it starts and the station at which it ends. So, for instance for this train that is this visible here 12259 the starting station would be Calcutta and the ending station would be Bikaner.

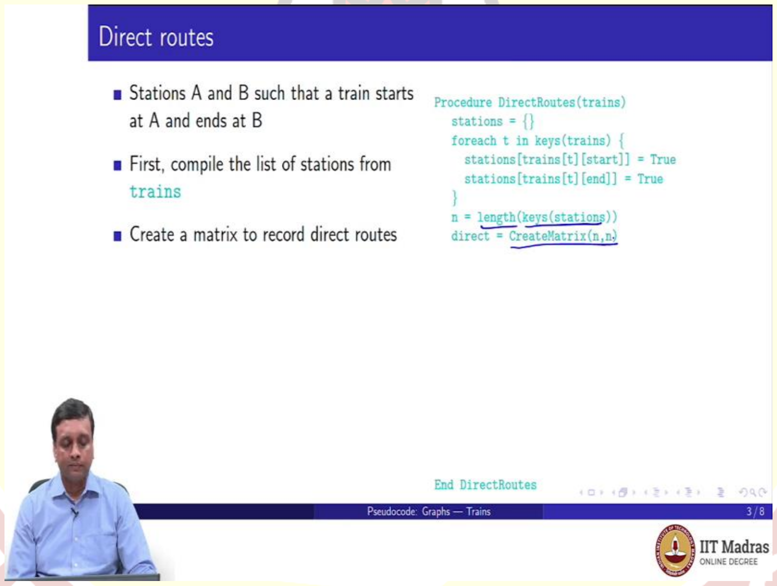
(Refer Slide Time: 01:54)



## Direct routes

- Stations A and B such that a train starts at A and ends at B
- First, compile the list of stations from `trains`

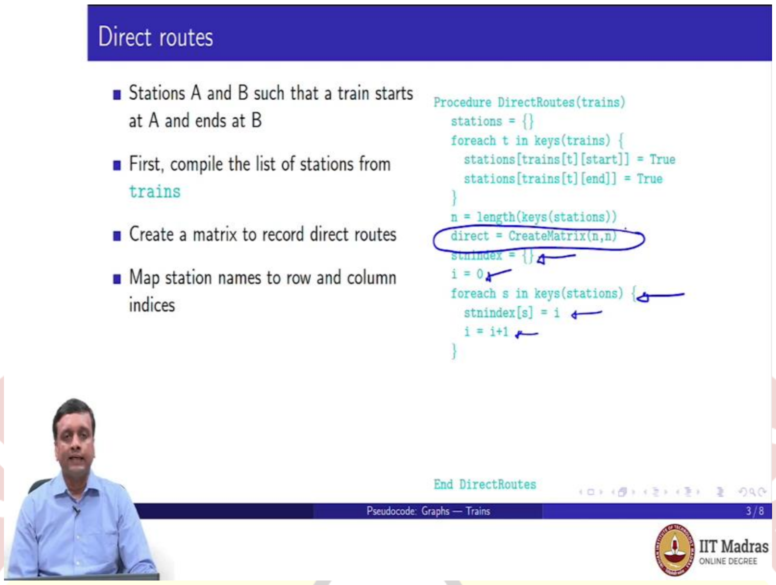
```
Procedure DirectRoutes(trains)
  stations = {}
  foreach t in keys(trains) {
    stations[trains[t][start]] = True
    stations[trains[t][end]] = True
  }
End DirectRoutes
```



## Direct routes

- Stations A and B such that a train starts at A and ends at B
- First, compile the list of stations from `trains`
- Create a matrix to record direct routes

```
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)
End DirectRoutes
```



## Direct routes

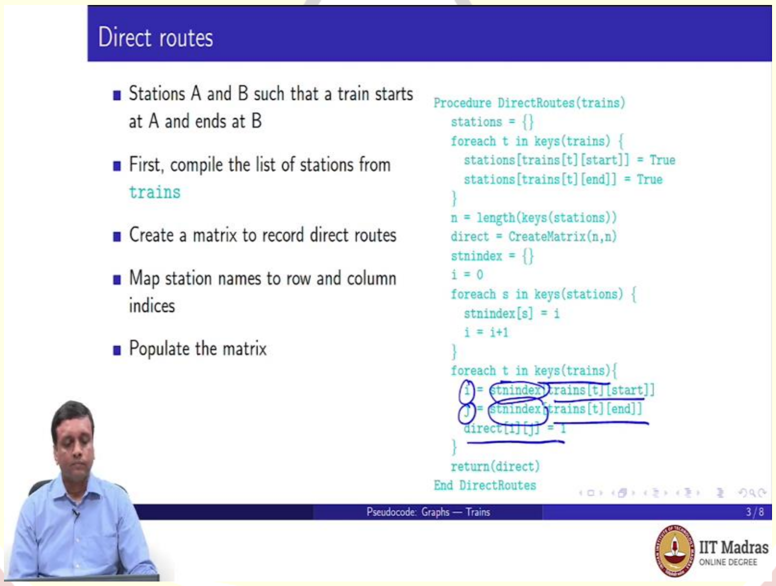
- Stations A and B such that a train starts at A and ends at B
- First, compile the list of stations from `trains`
- Create a matrix to record direct routes
- Map station names to row and column indices

```
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
  }

End DirectRoutes
```

Pseudocode: Graphs — Trains 3 / 8

IIT Madras ONLINE DEGREE



## Direct routes

- Stations A and B such that a train starts at A and ends at B
- First, compile the list of stations from `trains`
- Create a matrix to record direct routes
- Map station names to row and 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
```

Pseudocode: Graphs — Trains 3 / 8

IIT Madras ONLINE DEGREE

So, now given this dictionary, so this dictionary records all the relevant information that we need about trains giving us the starting station and the ending station of every card, from this we want to compute the set of direct route. So, as we said a direct route is one which connects two stations A and B such that there is a train that starts at A and ends and B, it is not that A and B are intermediate stations along that train route.

So, the first thing we have to do is to figure out which all stations we need to keep track of. So, there is a bunch of stations, but they are, we are looking only at the train's cards, we do not have the station's cards with us, so we do not know in advance which stations are there. So, what we do is we go through all the train's entries in the dictionary and we systematically collect in a new dictionary called stations, which we initialize to the empty dictionary, in this new dictionary we systematically collect entries for every station that we encounter either as a starting point or as a ending point of some train.

So, the end of this for each loop what we have done is we have populated this new dictionary stations with keys for every station that is either the starting point or the ending point of some train. So, now we will like to have a graph which represents which pairs of stations are connected by a direct train So, for a graph remember that we store it as a matrix and the matrix has one row for every node and one column for every node and the ijth entry will be 1 if there is an edge from i to j.

Now, what is n in this case? Well in n in this case is the number of stations well we do not know the number of stations in advance, but we have just calculated it in the previous loop, so we have a dictionary in which every key is a station, so we can get the number of station by taking the keys of this dictionary and taking the length of that list and setting that to be n and once we know n we can create the matrix which we will call direct to represent the direct connections between stations by our trains in our data set.

So, we have now these stations and we have one row per station but the rows and the columns in our matrix will be numbered from 0 to N minus1, where is the stations in our card were recorded with names, so we have to construct a mapping which maps every named station to a number between 0 and N minus 1, so that is what we do next. So, what we do is we start with this new dictionary called station index, so what station index will do is it will take as a key the name of a station and give back as a value, a position between 0 and N minus1 indicating which row or which column corresponds to that station.

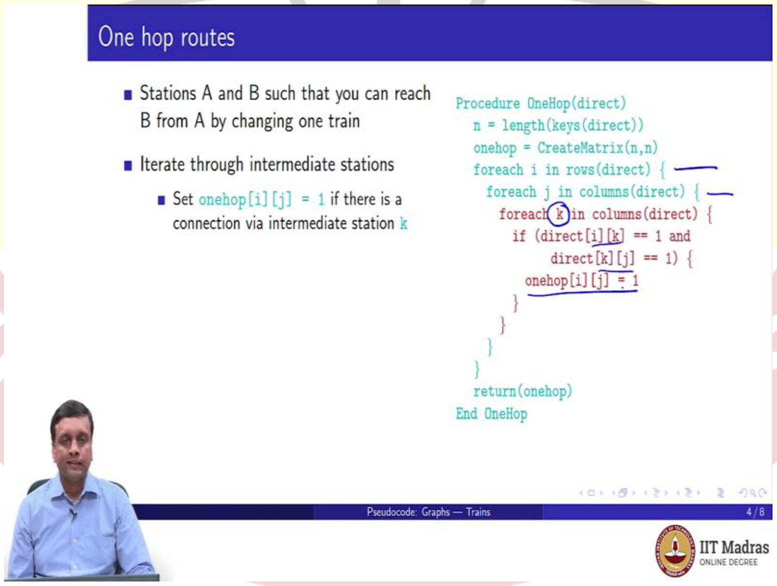
So, we just systematically go through all the keys and we start with the index 0, because remember that our rows and columns are going to be numbered from 0 to N minus1, so for each station that we pick up we assign it the current value of the index i and then we increment i. So, in this way i will run from 0 to N minus 1 and each station that we pick up in whatever order we pick it up it does not really matter to us, in whatever order it comes to us from the keys of this new dictionary stations that we populated above we will now have a mapping from stations to indices.

So, now we are finally set to populate this matrix, so our interest is to set up this matrix direct which will have an edge from i to j, if there is a train that connects station i to station j. So, we just have to populate this matrix by going through all the trains in our train's dictionary, so for every train which is available in our dictionary we pick up the starting station and the ending station, then we use this station index dictionary to map it to a number i and a number j and this says the train t connects i to j, so given this we will now set the entry direct ij to be 1.

So, we said direct ij to be 1, if the train that we are currently looking at connects i to j, starts at station i and ends at station j. And what are i and j? Those are the indices corresponding to the names of the stations that we have in our dictionary. So, that is why we need this station indexed dictionary.

So, this kind of systematic process takes us from the train dictionary which has information about the starting and ending point of each train and constructs from it, this graph of direct connections between stations where a direct connection remember is a connection which takes us from a starting point of a train to the ending point of a train.

(Refer Slide Time: 06:34)



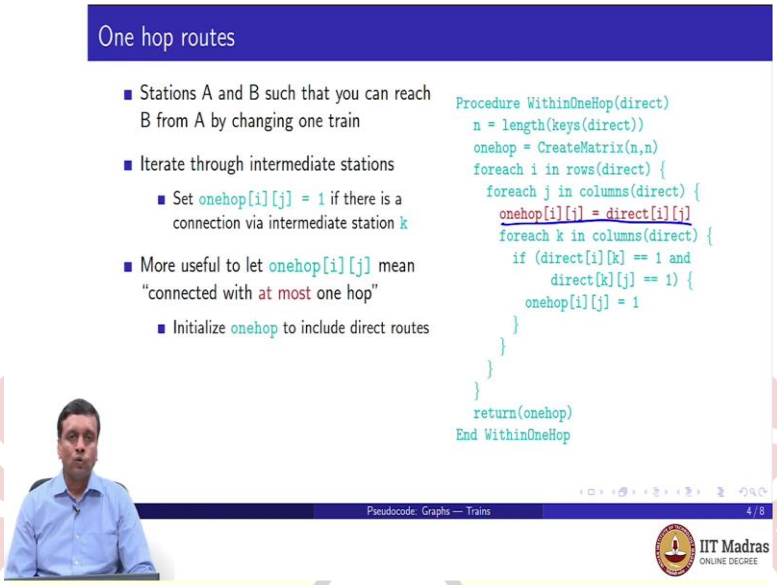
## One hop routes

- Stations A and B such that you can reach B from A by changing one train
- Iterate through intermediate stations
  - Set onehop[i][j] = 1 if there is a connection via intermediate station k

```
Procedure OneHop(direct)
  n = length(keys(direct))
  onehop = CreateMatrix(n,n)
  foreach i in rows(direct) {
    foreach j in columns(direct) {
      foreach k in columns(direct) {
        if (direct[i][k] == 1 and
            direct[k][j] == 1) {
          onehop[i][j] = 1
        }
      }
    }
  }
  return(onehop)
End OneHop
```

Pseudocode: Graphs — Trains 4 / 8

IIT Madras ONLINE DEGREE



## One hop routes

- Stations A and B such that you can reach B from A by changing one train
- Iterate through intermediate stations
  - Set onehop[i][j] = 1 if there is a connection via intermediate station k
- More useful to let onehop[i][j] mean "connected with at most one hop"
  - Initialize onehop to include direct routes

```
Procedure WithinOneHop(direct)
  n = length(keys(direct))
  onehop = CreateMatrix(n,n)
  foreach i in rows(direct) {
    foreach j in columns(direct) {
      onehop[i][j] = direct[i][j]
      foreach k in columns(direct) {
        if (direct[i][k] == 1 and
             direct[k][j] == 1) {
          onehop[i][j] = 1
        }
      }
    }
  }
  return(onehop)
End WithinOneHop
```

So, given that we know how to go directly from one station to another by a train, the next logical question to ask is where all can be go if we are allowed to change trains once. So, let us call this a one hop route. So, we are allowed to go from A to B by first going to an intermediate station C getting off at C and changing trains and going from C to B. Remember the changing trains means that the station that we get off with must be the terminus of the first train, it must be the last station of the train from A to C and then it must be the first station of the train from C to B.

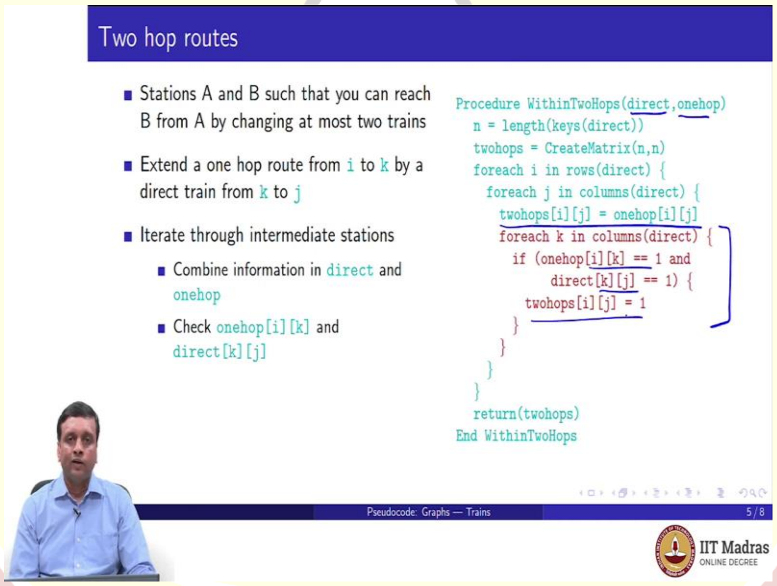
Now, we could make this more general and talk about all possible connections with intermediate stations is just to simplify the code that we have taken this approach right here. So, we want to know where all we can go by going via an intermediate station C. So, what we have to do for this is to iterate through all the intermediate stations. So, we take every station, i every station j and then we look for every intermediate station k whether it is possible to go from i to k and then from k to j by a direct train. If this is the case then we set this one hop to be 1.

So, this is how we can now take the direct train information and expand it in some sense to create new entries we have a new graph now, so one hop is a different graph direct ij was 1, if there was a train with started i and ended a j, one hop ij is 1, if there is a train that starts at i goes to some other station k and there is another train that starts at that k and goes to j. Now, this is kind of disjoint from direct, so one hop as we have constructed requires one hop, direct does not have any hops, but usually it will turn out that it is more convenient to combine the two.

So, rather than have exactly one hop, it is better to have the dictionary the graph that we construct to be at most one hop, so it must be either a direct route or via one hop. So, this is very easy to organize, so all we have to do is that before we start computing the one hop routes, we first initialize all the direct routes to be available to us. So, we add this line before saying that if there is a line if direct i, j is 1 then one hop ij is also 1.

In other words, even with 0 hops I can go from i to j. Of course if there is no direct route, then at this point one hop i j remains 0 and it will have to be set to 1 only by finding a new route. So, we have now calculated modified version of one hop which we the procedure is called within one hop which means that with at most one change I can go from i to j. So, why should we stop here?

(Refer Slide Time: 09:15)



## Two hop routes

- Stations A and B such that you can reach B from A by changing at most two trains
- Extend a one hop route from i to k by a direct train from k to j
- Iterate through intermediate stations
  - Combine information in direct and onehop
  - Check onehop[i][k] and direct[k][j]

```
Procedure WithinTwoHops(direct,onehop)
  n = length(keys(direct))
  twohops = CreateMatrix(n,n)
  foreach i in rows(direct) {
    foreach j in columns(direct) {
      twohops[i][j] = onehop[i][j]
      foreach k in columns(direct) {
        if (onehop[i][k] == 1 and
              direct[k][j] == 1) {
          twohops[i][j] = 1
        }
      }
    }
  }
  return(twohops)
End WithinTwoHops
```

Pseudocode: Graphs — Trains 5 / 8

IIT Madras ONLINE DEGREE

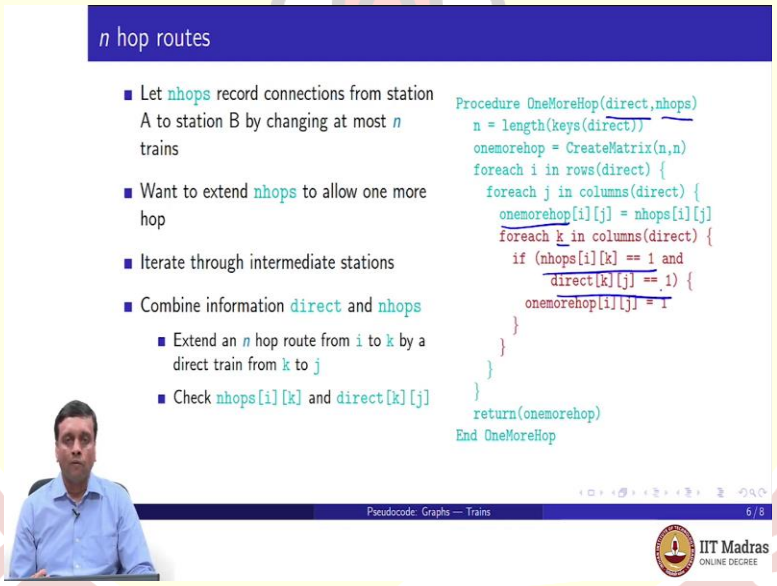
So, now that we know how to go from i to j changing trains at most once, can we go from i to j now changing trains at most twice? How can we do this? Well, we know how to go up in one hop, so if we go in one hop from A to B and then we can go from B to C by direct train, then we can go in two hops from A to C. So, we do the same thing as before except that we break up the two segments, earlier we were looking for a direct train from i to k and another direct train from k to j and saying that with these two direct routes I get a one hop route.

Now, we are saying that one of the segments can have one hop, the other one should be direct. So, we have a very similar loop as before except now this procedure must take two pieces of information, it must take the original graph, the direct edges and also the one hop graph which we have computed in the previous procedure which has at most one hop stop between i and j and taking these two I can now populate this matrix as before.

So, first of all as before we will say that this is at most two hops. So, the first thing we do is we initialize every entry two hops ij to be the same as one hop ij, so if there was already a one hop route, that is at most one hop route, then there is also an at most two hop route. Of course if there was not a one hop loop then we have to find it and that is what we are doing in this red section.

So, what we are saying is that for every intermediate station k if I can go from i to k via one hop and I can take a direct train from k to j, then in two hops I can go from i to j. So, this is a very natural extension of the previous thing to compute the one hop matrix I needed only the direct matrix, to compute the two hop matrix I need both the direct entries and the one hop entries.

(Refer Slide Time: 11:02)



## *n* hop routes

- Let nhops record connections from station A to station B by changing at most *n* trains
- Want to extend nhops to allow one more hop
- Iterate through intermediate stations
- Combine information direct and nhops
  - Extend an *n* hop route from i to k by a direct train from k to j
  - Check nhops[i][k] and direct[k][j]

```
Procedure OneMoreHop(direct,nhops)
  n = length(keys(direct))
  onemorehop = CreateMatrix(n,n)
  foreach i in rows(direct) {
    foreach j in columns(direct) {
      onemorehop[i][j] = nhops[i][j]
      foreach k in columns(direct) {
        if (nhops[i][k] == 1 and
            direct[k][j] == 1) {
          onemorehop[i][j] = 1
        }
      }
    }
  }
  return(onemorehop)
End OneMoreHop
```

Pseudocode: Graphs — Trains 6 / 8

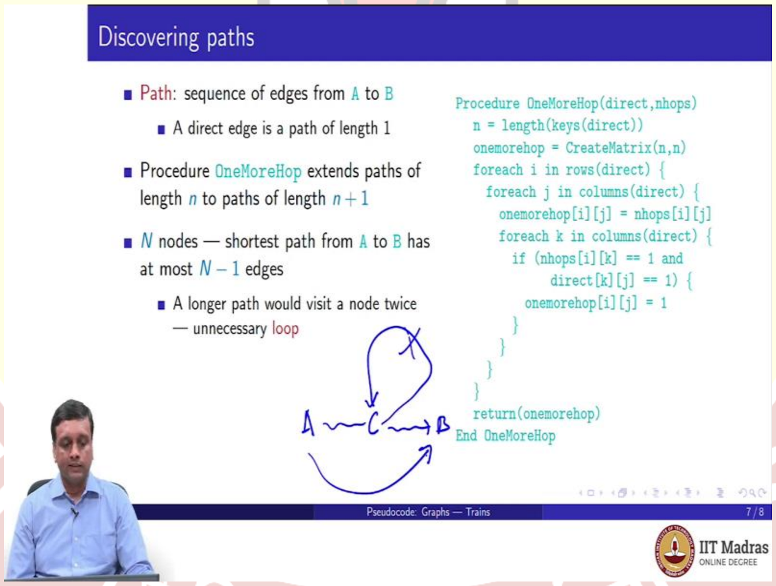
IIT Madras ONLINE DEGREE

So, we can extend this argument even further, so we can go not just from one to two, but we can go from two to three and three to four and so on. And in general we can go from any n to any n plus 1. So, suppose we have computed in this iterative process an n hops matrix which tells us how to go whether we can go from A to B changing at most n trains in between, so we can change trains n times. Now, we want to know whether we can go from A to B by changing trains n plus 1 times.

So, here as before we need the directory information and the n hops information, because if I want to go with n plus one hops, then the logical thing to do is to first go from i to an intermediate session k with just n hops and then take a direct train from k to j, I can always break up n plus one hops as n hops plus the last hop. So, that is exactly what we did for two and it happens to work for any n plus 1, so when I go from one to two, I take one hop and make a two hops, when I take n to n plus 1, I take n hops and I will get a new matrix which I can call one more hop in this case.

So, we are calling it one more hop, we can also call it n plus 1 hops if you wish. So, we are just doing the same thing. So, we are saying is there a k such that I can go from i to k in n hops and then I can go from k to j without stopping in a direct way. So, now what we saw therefore is that we can start with the direct matrix and compute the one hop matrix, when the one hop matrix and the direct matrix we can compute the two hop matrix, in the same way with the direct and two hop you could have computed the three hop and here we have seen that with direct and any number of hops we can compute one more hop.

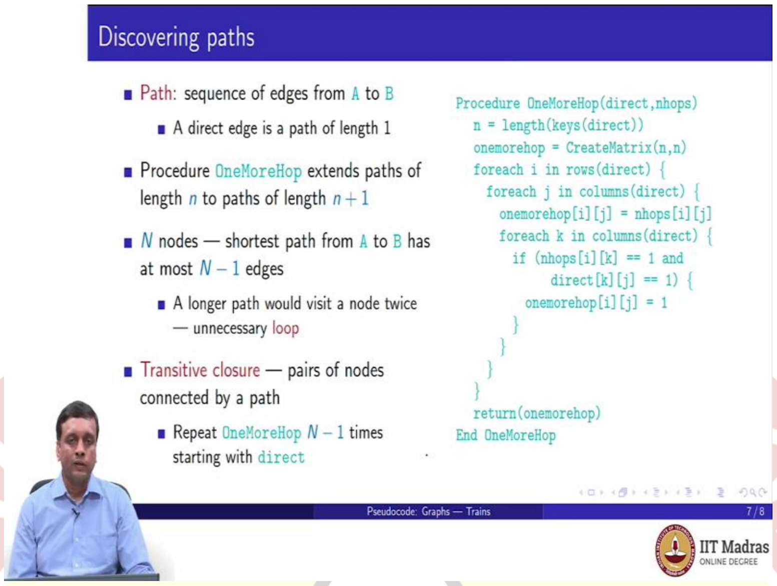
(Refer Slide Time: 12:47)



## Discovering paths

- Path: sequence of edges from A to B
    - A direct edge is a path of length 1
- Procedure OneMoreHop extends paths of length $n$ to paths of length $n+1$
- $N$ nodes — shortest path from A to B has at most $N-1$ edges
    - A longer path would visit a node twice — unnecessary loop

```
Procedure OneMoreHop(direct,nhops)
  n = length(keys(direct))
  onemorehop = CreateMatrix(n,n)
  foreach i in rows(direct) {
    foreach j in columns(direct) {
      onemorehop[i][j] = nhops[i][j]
      foreach k in columns(direct) {
        if (nhops[i][k] == 1 and
            direct[k][j] == 1) {
          onemorehop[i][j] = 1
        }
      }
    }
  }
  return(onemorehop)
End OneMoreHop
```



## Discovering paths

- Path: sequence of edges from A to B
  - A direct edge is a path of length 1
- Procedure OneMoreHop extends paths of length $n$ to paths of length $n+1$
- $N$ nodes — shortest path from A to B has at most $N-1$ edges
  - A longer path would visit a node twice — unnecessary loop
- Transitive closure — pairs of nodes connected by a path
  - Repeat OneMoreHop $N-1$ times starting with direct

```
Procedure OneMoreHop(direct,nhops)
  n = length(keys(direct))
  onemorehop = CreateMatrix(n,n)
  foreach i in rows(direct) {
    foreach j in columns(direct) {
      onemorehop[i][j] = nhops[i][j]
      foreach k in columns(direct) {
        if (nhops[i][k] == 1 and
            direct[k][j] == 1) {
          onemorehop[i][j] = 1
        }
      }
    }
  }
  return(onemorehop)
End OneMoreHop
```

So, what we are doing in this process is discovering what are called paths in a graph, so a path as you would expect is a way to go from A to B, so you have to follow edges, so it is a sequence of edges. So, the simplest path is a direct edge, so a direct Edge consists of just that single edge and it is a path of length 1. Now, if I take one edge and then I follow it by another edge, so I go from A to C for example and then I take another edge and go from C to B, then I have a path of length **two** from A to B.

So, what we have done with that one more hop procedure that we just described is to take paths which are of length n and extend them to paths of length n plus 1. Now, in general if we have capital N nodes in our graph and I want to know if there is a path from A to B, then it turns out that I need not look at paths which I have more than N minus 1 edges, because if I have N minus 1 edges then each edge that I take from A will take me to potentially a new node, but there are totally only n nodes, so I start with A, I take the first edge I get to a new node.

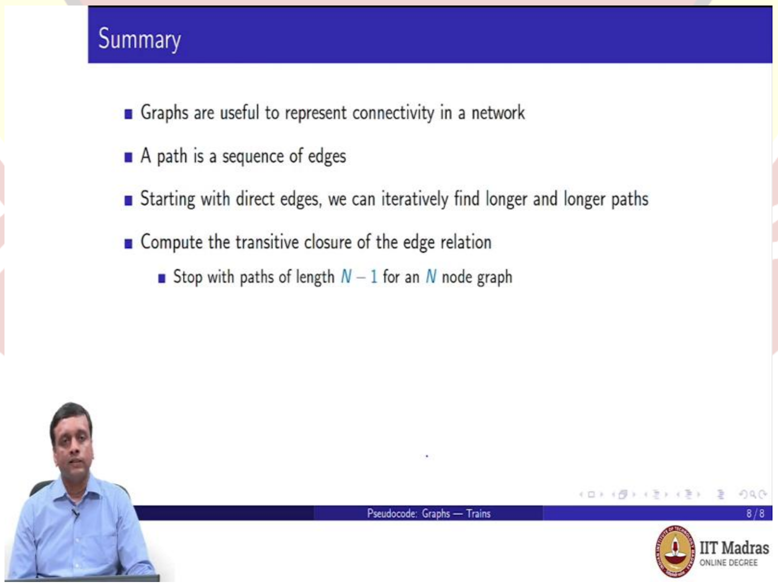
So, now I have seen two nodes, I take the second edge I have come to a new node I see the third node, I take N minus 1 edges I have seen every possible node in my graph and of course if I visit the same node twice, so if I start at A and then I go somewhere and I come to a C and then after some time I come back to a C and then I continue and go to B then this whole segment from C to C is not needed to connect A to B, it is true that there is a way to go from A to C and then C back to C and then go to B.

But my interest in knowing is weather I can go from A to B by a path or not, for that I need not look at paths with loops. So, therefore if I only look at paths without loops, then such a path cannot have more than N minus 1 edges, so what this means is that by doing this one more hop starting with the direct thing which has one single edge paths and doing this one more hop N minus 2 times we can compute all the paths which are of length N minus1 and once I have computed all the paths of length N minus 1 there is not going to be any more change in terms of which connections are there between stations.

So, this process of finding all the things which are reachable, all the pairs which are connected by some sequence of edges by some path, this is what is called the transitive closure. So, mathematically we have a relation and edge relation connecting some pairs of elements and if we now take edge followed by edge followed by edge and I say if I can take an arbitrary number of edges, how far can I go and what are the pairs which are connected?

This is called the transitive closure of the original relation. So this procedure that we have described this one more hop procedure takes the original edge relation and it keeps updating the relation with one hops, two hops, three hops and so on. And then we know because it is an n node graph that after N minus 1 such iterations we can stop.

(Refer Slide Time: 15:54)



## Summary

- Graphs are useful to represent connectivity in a network
- A path is a sequence of edges
- Starting with direct edges, we can iteratively find longer and longer paths
- Compute the transitive closure of the edge relation
  - Stop with paths of length $N-1$ for an $N$ node graph

Pseudocode: Graphs — Trains 8/8

IIT Madras ONLINE DEGREE

To summarize what we have seen in this example as different from the example to do with students and you know relationships between students, here we have seen that when you have a kind of spatial graph like a train network, then a graph is a very natural way to represent connectivity in this network. So, in this network edges represent direct trains which go from A to B and then paths represent routes that we can follow by changing trains.

So, a path is a sequence of edges and we saw that we start with direct edges and by iteratively applying this one more hop procedure we can find longer and longer paths and in particular if the graph is of a fixed size of N, capital N then we know that after we do this iteration N minus 1 times we have discovered all the connected pairs, there are not going to be any more connected pairs because any longer path will have a loop which does not connect any new pair. So, this is what is called the transitive closure. So, we have seen how to compute the transitive closure of the edge relation of a graph.