

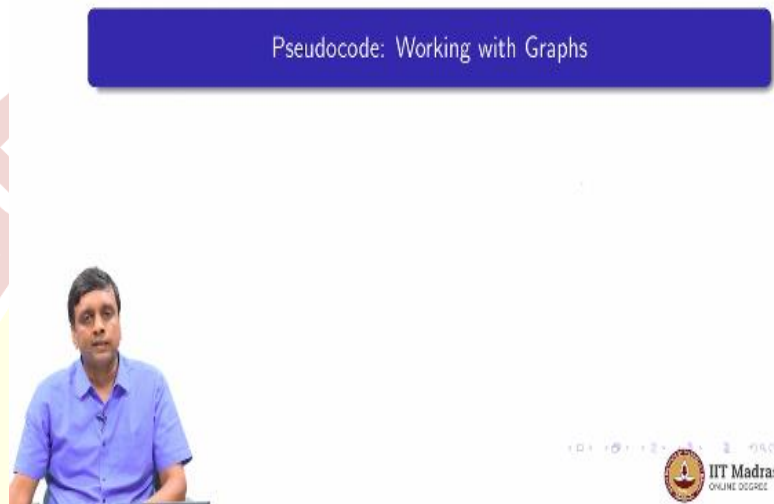


# IIT Madras

ONLINE DEGREE

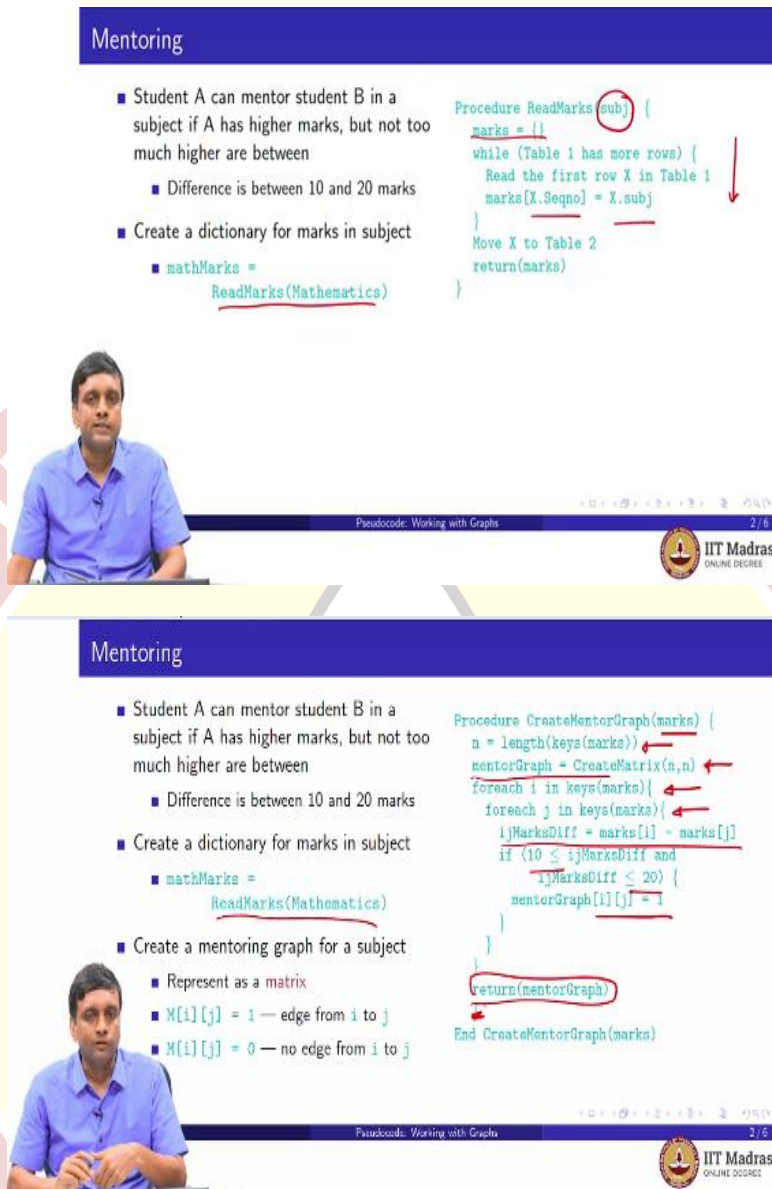
**Computational Thinking**  
**Professor G. Venkatesh**  
**Department of Computer Science**  
**Indian Institute of Technology, Madras**  
**Pseudocodes for Real-Time Examples Using Graphs**

(Refer Slide Time: 00:14)



We know have matrices as new collections that are disposable. These are two disposals these are two dimensional tables where we can access the value at any row and any column in constant time. So these are random access collections two dimensional tables and these are most useful as we mentioned to represent graphs. So let us look at examples where we are working with graphs and see how we can use matrices to write pseudocodes for these examples.

(Refer Slide Time: 00:39)



**Mentoring**

- Student A can mentor student B in a subject if A has higher marks, but not too much higher are between
  - Difference is between 10 and 20 marks
- Create a dictionary for marks in subject
  - `mathMarks =`  
`ReadMarks(Mathematics)`

```
Procedure ReadMarks(subj) {  
  marks = {}  
  while (Table 1 has more rows) {  
    Read the first row X in Table 1  
    marks[X.Seqno] = X.subj  
  }  
  Move X to Table 2  
  return(marks)  
}
```

**Mentoring**

- Student A can mentor student B in a subject if A has higher marks, but not too much higher are between
  - Difference is between 10 and 20 marks
- Create a dictionary for marks in subject
  - `mathMarks =`  
`ReadMarks(Mathematics)`
- Create a mentoring graph for a subject
  - Represent as a matrix
  - `M[i][j] = 1` — edge from i to j
  - `M[i][j] = 0` — no edge from i to j

```
Procedure CreateMentorGraph(marks) {  
  n = length(keys(marks))  
  mentorGraph = CreateMatrix(n,n)  
  foreach i in keys(marks){  
    foreach j in keys(marks){  
      ijMarksDiff = marks[i] - marks[j]  
      if (10 ≤ ijMarksDiff and  
          ijMarksDiff ≤ 20) {  
        mentorGraph[i][j] = 1  
      }  
    }  
  }  
  return(mentorGraph)  
End CreateMentorGraph(marks)
```

So the first example is from the classroom dataset where we talked about mentoring. So we said that students can mentor each other depending on their relative marks. So students with higher marks can mentor a student with lower marks in a subject provided their difference in marks is substantial so that there is a visible difference in the performance, but it is not so substantial that the weaker student will get intimidated by the stronger student.

So, in particular we just choose two values for this. We said that the marks should be at least 10 apart and should be no more than 20 apart. So A can mentor B if the marks of A in a given subject so this is mentoring per subject. So for example A can mentor B in mathematics if A's maths marks are at least 10 more than B's maths marks, but no more than 20 more. So let us try to represent this mentoring relationship who can mentor whom in a subject as a graph.

So we start by creating the information about the marks in that subject. So here is a simple procedure which walks down our table of marks and fixes a given subject and what it does is it reads each row in the table and it creates a dictionary which is indexed the keys are the students IDs in this case there are the sequence number of the card and the value is the subject marks in the particular subjects that I have chosen.

So if I call it like this as a math mark is equal to read marks mathematics then sub is equal to mathematics. So it will go down the table pull out the maths marks for each card and assign it against the roll number or the ID of the student so that we have a dictionary at the end which maps IDs to maths marks. Now from this dictionary we want to create a graph which represents the mentoring relation.

So how do we represent a graph in a matrix? So a graph remember consists of some vertices these are these objects which are connected by edges. So in this case for our student thing it is very convenient that the student IDs were also numbered from 0 to 29. So remember that in a matrix the rows and the columns are numbered from 0 to  $n-1$ . So we can use the row and the column index directly to represent the object in this case which is a student ID.

And now you want to record in the matrix whether or not there is an edge from  $i$  to  $j$ . So by default when you create a matrix we get all 0s so 0s consists of a graph in which there are no edges. So when we add an edge we reset that 0 to 1 so if  $M[i][j]$  is 1 that means there is an edge from  $i$  to  $j$  if  $M[i][j]$  is 0 there is no edge from  $i$  to  $j$ . So now let us see how we can take this dictionary that we created with the maths marks and create the mentoring graph for mathematics.

So here again this is a general procedure which creates a mentoring graph for any subject marks. So it just takes a dictionary of subject marks it does not know which subject it is all it knows is that it is a dictionary in which the keys are the roll numbers and the values are the marks in some subject. So the first thing that this procedure does not know is how many students there are in a class.

But that we can get because if we look at the keys we are just assuming that the roll numbers are in 0 to  $n-1$ , but we do not know what  $n$  is. So we can get  $n$  by asking how many keys are there. So give me the length remember the length function for list the length of the list consisting of the keys of the dictionary. So this is the first step and once we know that this is going to be our graph.

Our graph will have that many nodes so we will create a matrix with  $n$  rows and  $n$  columns. So we want to represent for each student  $i$  and each student  $j$  whether  $i$  can mentor  $j$   $i$  ranges from  $0$  to  $n-1$   $j$  ranges from  $0$  to  $n-1$ . Now it is a very simple matter of assigning these edges. Remember that by default when I create a matrix I have a disconnected graph nothing is connected to everything anything else because all the entries are zero.

So now I just iterate through all the marks. So I look for every student and for every other student I compute the marks difference between  $i$  and  $j$ . So I look at the marks of  $i$  and I subtract the marks of  $j$  and what we said was that if this marks difference is bigger than  $10$  and smaller than  $20$  so  $i$  must be bigger than  $j$  for  $i$  to mentor  $j$  marks of  $i$  must be bigger than marks of  $j$ , but within the range  $10$  to  $20$ .

So if it is within this range then I will update this mentor graph which is the matrix I created. I will update this mentor graph to have an edge there so this is how I create the edges. I just check for every pair by brute force I check for every pair whether or not if mentoring relation exists and I will add it. So this remember it is a directed thing so  $i$  can mentor  $j$ , but  $j$  does not mentor  $i$  if  $j$  were to mentor  $i$   $j$  would have to have more marks than  $i$ .

Of course in a given subject it cannot be both ways if  $A$  has got more marks than  $B$  then  $B$  has less marks than  $A$  if  $A$  mentors  $B$  it cannot work the other way so this is going to be what is called a directed graph and this is how we set it up and finally what we do is we return this graph so we return the graph this bracket should not be there. So we return the graph so the outcome of this is I pass a dictionary with marks and I get back the mentoring graph for that subject for whichever subject I pass a dictionary.

(Refer Slide Time: 06:00)

**Pairing students in study groups**

- A can mentor student B in one subject and B can mentor A in the other
- Study groups in Maths and Physics
  - Create mentoring graphs for each
- Use the mentoring graphs to pair off students

```
mathMarks = ReadMarks(Mathematics)
phyMarks = ReadMarks(Physics)
mathMentorGraph = CreateMentorGraph(mathMarks)
phyMentorGraph = CreateMentorGraph(phyMarks)
paired = {}
foreach i in rows(mathMentorGraph)
  foreach j in columns(mathMentorGraph)
    if (mathMentorGraph[i][j] = 1 and
        phyMentorGraph[j][i] = 1 and
        not(paired[i]) and
        not(paired[j])) {
      paired[i] = j
      paired[j] = i
    }
```

Handwritten notes on the code: *i → j*, *not (isKey(paired,i))*

Video inset: A man in a blue shirt speaking.

Footer: Pseudocode: Working with Graphs, IIT Madras ONLINE DOOR

So now the next step was to pair up students. So we said let us construct this mentoring relation in two subjects for instance maths and physics and then see whether we can group students in pairs such that A can help B in maths and B can help A in physics. So this will give them some incentive we said to study together because each is learning from the other, but each is also teaching the other.

So each of them gets satisfaction both ways it is not like a one sided thing where all the subjects are being taught by one student to another student. So it is a more equitable thing so we wanted to create this study groups. So if you want to do this in maths and physics what we have to first do of course is create the mentoring graphs for the two subjects. So as before we first set up the dictionaries by reading the marks in maths and reading the marks in physics and creating these two dictionaries.

Then we call our mentor graph procedure with the maths marks and the physics marks and we get back these two dictionaries two graphs rather the maths mentor graph and the physics mentor graph. Now what do we want to know? We want to know whether there is a pair such that A can mentor B in maths and B can mentor A in physics. So there must be an edge in one direction in one graph and an edge in the opposite direction in the other graph.

So what we will do is we will just keep track of who has been paired off and whenever we find the pair that can be joined together we will join them and record this. So we start off with a empty dictionary called paired. So the intention of paired is that paired will say that i is



paired with  $j$ . So the paired will have as keys student ID and the value will be another student ID namely the pair of  $j$ .

So if pair of  $i$  is  $j$  then pair of  $j$  will be  $i$  because they will be actually symmetric, they will be in pairs. So how do we do this? Well we look at every row for example in the math mentor graph and every column in the math mentor graph. So we talk through math mentor graph systematically looking at every entry. So whenever we find that this should be double equal to whenever we find the entry in the math mentor graph is 1.

So that  $i$  can mentor  $j$  in maths. We check whether  $j$  can mentor  $i$  in physics. So if  $i$  can mentor  $j$  in maths and  $j$  can mentor  $i$  in physics and more importantly both of these students are not currently paired off so it is not the case that paired of  $i$  is there and it is not the case that paired to  $j$  is there then we will pair  $i$  with  $j$  and pair  $j$  with  $i$ . So here what we are really checking is, so this should technically be not is key paired  $i$  so that is what this means actually.

So I will correct that, but when I say not paired of  $i$  I mean that  $i$  is not currently yet a key in paired and similarly not paired of  $j$  means  $j$  is not yet a key in paired. So if this is true then these two are available they are compatible because they can pair each other. So now I will update it so that paired of  $i$  is  $j$  and paired of  $j$  is  $i$ . So this is sort of doing it in one direction so it looks like we are only looking at situations where  $i$  can pair  $j$  in maths and  $j$  can pair  $i$  in physics what is if the other way round.

What if  $i$  can pair  $j$  in physics and  $j$  can pair  $i$  in maths well actually this is not a problem because what will happen is that we will discover it eventually the other way round. So for every entry so if  $j$  can pair I mean for a mentorship thing there has to be an entry in both matrices. So somewhere there will be an entry for  $j$ ,  $i$  also in the maths mentor graph saying it is 1 and at that point I will discover that  $i, j$  is in physics.

So since I am systematically going through the entire math mentorship graph I will look at  $i, j$  and  $j, i$  separately. So  $i$  will become  $j$  and  $j$  will become  $i$  so it really does not matter that it looks like I am doing it in only one direction I will actually uncover all the mentorship pairs and collect them in some order. So this is how we can use these mentorship graphs to create some interesting objects in this case the study groups.

(Refer Slide Time: 10:16)

**Popular students**

- A student who can be mentored by many other students is **popular**
- Create mentoring graphs for all three subjects

```
CodemathMarks = ReadMarks(Mathematics)
phyMarks = ReadMarks(Physics)
chemMarks = ReadMarks(Chemistry)

mathMentorGraph =
  CreateMentorGraph(mathMarks)
phyMentorGraph =
  CreateMentorGraph(phyMarks)
chemMentorGraph =
  CreateMentorGraph(chemMarks)
```

**Mentoring**

- Student A can mentor student B in a subject if A has higher marks, but not too much higher are between
  - Difference is between 10 and 20 marks
- Create a dictionary for marks in subject
  - `mathMarks = ReadMarks(Mathematics)`

```
Procedure ReadMarks(subj) {
  marks = {}
  while (Table 1 has more rows) {
    Read the first row X in Table 1
    marks[X.Seqno] = X.subj
  }
  Move X to Table 2
  return(marks)
}
```

So another question that we addressed related to mentoring was the idea of popularity. So we said perhaps it could be that one measure of popularity in a class is how many students are able to mentor you. So a student who is popular is one who could be associated with many other students in study groups. So what we want to now check is how many incoming edges there are across the mentor graphs.

So in each subject a student could be mentored by another student. So a student is popular if they can be mentored by as many students as possible across the 3 subjects. So here there are 3 subjects; maths, physics and chemistry. So the beginning is the same as before we create these three dictionaries by reading the marks of each of these into separate dictionaries and we pass this dictionary separately and create three mentoring graphs.



So now we have the math mentor graph, the physics mentor graph and the chemistry mentor graph. Now we want to check across these three graphs how many incoming edges a student has, how many times a student can be mentored. So this is one way to do this so we create a dictionary called popularity the popularity will have as its keys the roll numbers of the student and the values will be how many incoming edges there are.

So we start with an empty dictionary. Now for each student who has math marks and for every student say again, so basically for every pair of students we are assuming that all students in this class have marks in all these three subjects. So I can pick up any one of those dictionaries and iterate through all the pairs in that. If I see that  $j$  can mentor  $i$  in maths then the popularity of  $i$  increases by 1.

If I see that  $j$  can mentor  $i$  in physics again the popularity increases by 1 and similarly in chemistry it increase by 1. So really what we are doing is across these three graphs we are looking at incoming edges, is there an edge from  $j$  to  $i$ .  $i$  is a student we are interested in,  $i$  is the student whose popularity we are calculating. So every time we see an edge from  $j$  to  $i$  we are going to increase the popularity of  $i$  by 1.

So we do this by systematically scanning all the possible  $j$  so that is why we are looking at every possible  $j$ . So notice that a student cannot mentor himself or herself because there will be a difference of 0 and we know that the difference for mentoring is between 10 and 20. So there is something unsatisfactory about this solution that we just saw which is that we are actually if you have a student who is weak in three subjects.

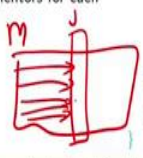
And can be mentored by a stronger student in across all these three subjects this will actually account for three incoming edges. So the same student mentoring same  $j$  mentoring  $i$  will be counted more than once. So this popularity that we are accounting here is actually over estimate because we are double counting or even triple counting some cases where a weak student has many people mentoring them on the same person mentoring them on different subjects.

(Refer Slide Time: 13:18)

### Popular students

- A student who can be mentored by many other students is **popular**
- Create mentoring graphs for all three subjects
- Count incoming mentoring edges for each student
- Avoid duplicates
  - Explicitly keep track of mentors for each student and count them

```
mentors = {}
popularity = {}
foreach j in columns(mathMentorGraph) {
  mentors[j] = {}
  foreach i in rows(mathMentorGraph) {
    if (mathMentorGraph[i][j] == 1){
      mentors[j][i] = True
    }
    if (phyMentorGraph[i][j] == 1){
      mentors[j][i] = True
    }
    if (chemMentorGraph[i][j] == 1){
      mentors[j][i] = True
    }
  }
  popularity[j] = length(keys(mentors[j]))
}
```



### Popular students

- A student who can be mentored by many other students is **popular**
- Create mentoring graphs for all three subjects
- Count incoming mentoring edges for each student
- Avoid duplicates
  - Explicitly keep track of mentors for each student and count them

```
mentors = {}
popularity = {}
foreach j in columns(mathMentorGraph) {
  mentors[j] = {}
  foreach i in rows(mathMentorGraph) {
    if (mathMentorGraph[i][j] == 1){
      mentors[j][i] = True
    }
    if (phyMentorGraph[i][j] == 1){
      mentors[j][i] = True
    }
    if (chemMentorGraph[i][j] == 1){
      mentors[j][i] = True
    }
  }
  popularity[j] = length(keys(mentors[j]))
}
```

So to account for that we can slightly change the code so what we will do now is we will actually keep track explicitly for each  $i$  who are the people who can mentor  $i$  and popularity will now be derived from this. So mentors will keep a unique list of mentors for each student and we will keep this unique list by making a dictionary as we will see and at the end we will just count how many mentors there are and make that the popularity.

So the rest is pretty much similar, but now we are going to use the graph version of this so earlier we used the dictionaries to iterate. So let me just iterate through the columns of the math mentor graph. So why are we iterating the columns because really what we are saying in a mentor graph is that if this is my matrix  $M$  and I am looking at a  $j$  I want to know how many incoming edges there are.

So I want to fix a column and go down and go through all the rows and check how many 1s there are in column  $j$  that is what is the incoming. The row  $i$  in a matrix will have the outgoing edges if I look at all the entries in row  $i$  it will say what  $i$  is connected to. So these are the edges which  $i$  is connected to whereas the column  $j$  has all the values of incoming edges for each row is it connected to  $j$ .

So that is why we are doing it with this form of iteration. So we are saying for every  $j$  which is in the columns of the math mentor graph you start off by assuming that  $j$  has no mentors and now again we scan each of these graphs to check whether or not there is somebody who can mentor them. So for every other  $i$  again we can go through the rows of the math mentor graph it does not matter the rows and the columns in this case are both the same.

Because it is a square matrix index by all the students but the main point is that if I find a mentor in math who can mentor  $j$  then I assign that key to be true. Similarly, if I find in physics I assign that key to be true and if I find in chemistry assign that key to be true, but the advantage of a dictionary is that whether this key was created once, twice or three times finally there is only one copy of the key.

So a dictionary is a very useful way to keep a kind of indirectly keep a list without duplicates instead of creating. So I could have also created an explicit list I could have said that mentors of  $j$  is a list starting with an empty list and every time I see a new entry I will add  $i$  to the list, but then I could add  $i$  multiples times and I have to keep track of duplicates. So instead of keeping mentors of  $j$  as a list I am keeping mentors of  $j$  as a dictionary.

And the members of that list are the keys which are present. So at the end of this whole thing I have a bunch of mentors who can mentor  $j$  and now if I want to assign the popularity I can just take the number of keys which are there so I take keys of mentor  $j$ . These are all the students who in one or more subjects can mentor  $j$  take the length of that and assign that to be the popularity.

So this illustrates that couple of things the main thing that this illustrates is that keeping a unique list. A list of unique values is very often conveniently done by converting that list into the keys of a dictionary. So this is a useful trick to use if you are trying to code with unique list with no duplicates.

(Refer Slide Time: 16:40)

**Similar students**

- Two students are similar if they have similar marks in all subjects
  - Difference is within 10 marks
- Dictionaries with marks in each subject

```
mathMarks = ReadMarks(Mathematics)
phyMarks = ReadMarks(Physics)
chemMarks = ReadMarks(Chemistry)
```
- Create a similarity graph

```
Procedure
CreateSimilarityGraph(marks1, marks2, marks3) {
  n = length(keys(marks1))
  similarityGraph = CreateMatrix(n, n)
  foreach i in keys(marks1) {
    foreach j in keys(marks1) {
      iDiff1 = abs(marks1[i] - marks1[j])
      iDiff2 = abs(marks2[i] - marks2[j])
      iDiff3 = abs(marks3[i] - marks3[j])
      if (iDiff1 ≤ 10 and
          iDiff2 ≤ 10 and
          iDiff3 ≤ 10) {
        similarityGraph[i][j] = 1
      }
    }
  }
  return(similarityGraph)
End CreateSimilarityGraph(marks)
```

Pseudocode: Working with Graphs 5/6

Logo: IIT Madras ONLINE DEGREE

So, finally let us look at this example that we said of similar students. So we said that two students are similar if they perform roughly at the same level in all three subjects. So it could be higher in one lower in another. So we are looking not in the mentorship case we strictly wanted  $i$  to be bigger than  $j$ . We said that  $i$  can mentor  $j$  if  $i$  has at least 10 marks more not more than 20 marks more.

Here we are saying that  $i$  and  $j$  are similar if their marks are within 10, but could be plus minus. So it could be that  $i$  has 5 marks more than  $j$  in maths 6 marks less in physics and 3 marks more in chemistry then  $i$  and  $j$  are said to be similar. So their difference in across the three subjects if I take the absolute value ignore which is bigger and which is smaller I get something which is less than 10.

So as usual I will assume that we have constructed these dictionaries which have the marks in each of the subjects and now I am going to create the similarity graph. So the usual thing first I have to determine what is the size of this graph. So the size of the graph is given by the number of students in the class which is the number of keys in any one of these dictionaries assuming that all three of the dictionaries have the same size and the same students.

So for example I take marks 1 so I will typically do this with I will pass as arguments the maths marks, the physics marks and the chemistry marks and then create the similarity graph from that. So those are my three dictionaries coming in. So I am passing a marks in three subjects. So I create this  $n$  by  $n$  matrix called the similarity graph and now I just have to decide for every  $i, j$  whether they are similar or not.

So that is easy enough as usual I go through all the  $i$  and  $j$  in my student list which could be for instance the keys of one of the dictionaries and I compute the absolute value of the difference in each subjects. So for  $i$  and  $j$  I look at marks of  $i$  minus marks of  $j$  marks 1, marks 2, marks 3 so the three subjects. Each case I do not want the difference as such, but I want the absolute value of the difference.

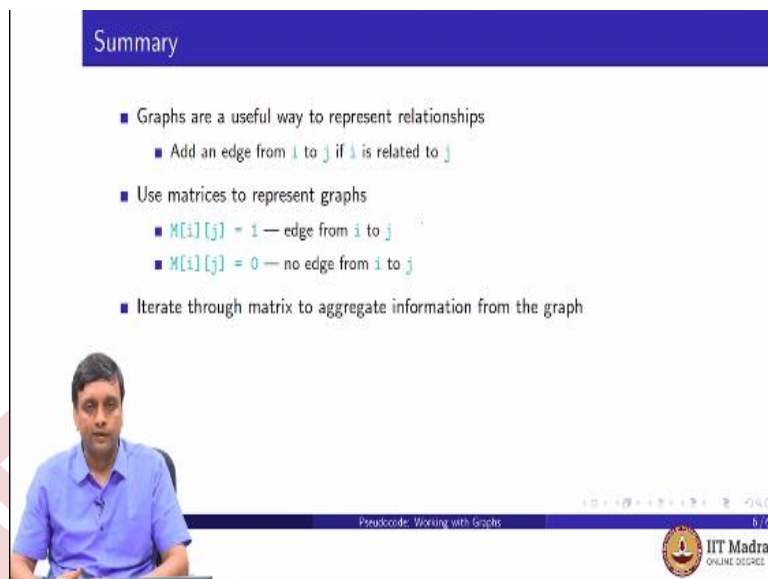
So I would call this function which I have just assumed which computes the absolute value. So remember the absolute value if it is a positive number it gives return so absolute value of 6 is 6 the absolute value of minus 7 is plus 7. The absolute value just removes the minus sign. So I get the difference in the first subject as  $ijdiff1$ , second subject  $ijdiff2$ , third subject  $ijdiff3$  and what we want is that all of these three differences in the three subjects are less than equal to 10 and if this is the case then I will set  $similarity\ graph[i][j]$  to be 1.

Now notice that if  $i$  is similar to  $j$  then  $j$  is also similar to  $i$ . So unlike the mentorship graph where if  $i$  mentors  $j$  then  $j$  cannot mentor  $i$ . So here we actually have something which is symmetric if  $i$  is similar to  $j$ ,  $j$  will be similar to  $i$  but we are only assigning it once we are not assigning the similar the symmetric pair because they will be because we are explicitly again doing all possible  $i$  and all possible  $j$  the symmetric pair will come up again.

So if I see 1, 3 as similar later on when I am doing row 3 and column 1 I will see 3, 1. So the symmetric pair will get added as a separate calculation on this so we do not have to worry about it, but this will be a symmetric graph in that sense. If there is an edge from  $i$ ,  $j$  there will also be edge from  $j$  to  $i$ . So sometimes you can in such situations you can discard the directions and call it an undirected graph.



(Refer Slide Time: 20:02)



**Summary**

- Graphs are a useful way to represent relationships
  - Add an edge from  $i$  to  $j$  if  $i$  is related to  $j$
- Use matrices to represent graphs
  - $M[i][j] = 1$  — edge from  $i$  to  $j$
  - $M[i][j] = 0$  — no edge from  $i$  to  $j$
- Iterate through matrix to aggregate information from the graph

Pseudocode: Working with Graphs 6/6

IIT Madras  
ONLINE DEGREE

So to summarize I just reiterating what we discussed in the class earlier which is that graphs are extremely useful way to represent relationships and the way we represent a relationship in a graph is to create an entity or a vertex for each element in our set in this case say the number of students and we add an edge from  $i$  to  $j$  if  $i$  is related to  $j$ . So if  $i$  can mentor  $j$  if  $i$  is similar to  $j$  we add an edge.

And the way we represent this in matrices is to use this 0, 1 entries. So  $M[i][j]$  when  $M$  represents the matrices over  $n$  vertices it will have  $n$  rows and  $n$  columns and  $M[i][j]$  will be 1 when there is an edge from  $i$  to  $j$  and  $M[i][j]$  will be 0 when there is no edge from  $i$  to  $j$  and then we saw that by systematically iterating through the matrix we can collect information about all the edges for instance when we are aggregating things about popularity and so on.

So once we have something in a graph form and convert it into a matrix form we can use the matrix to do interesting calculations about the graph.