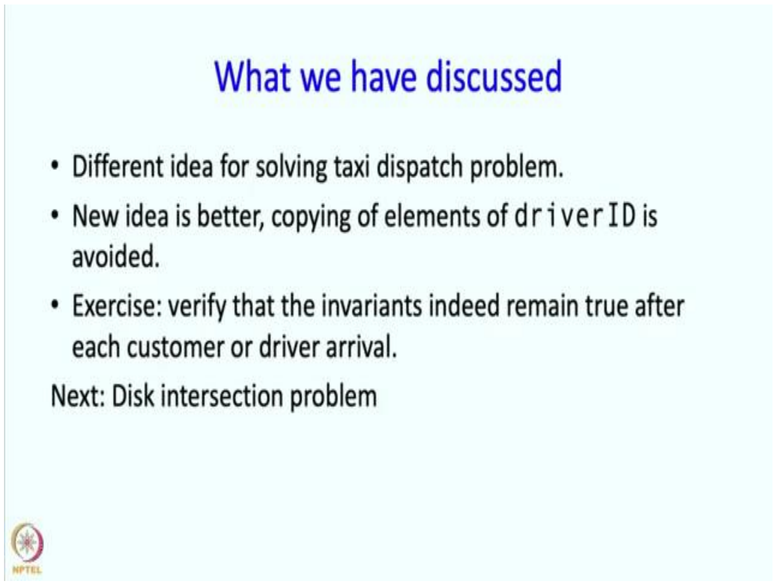
An Introduction to Programming through C++

Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay

Lecture 15 Part-18
Array Part-1
Disk Intersection

(Review Slide Time: 0:18)



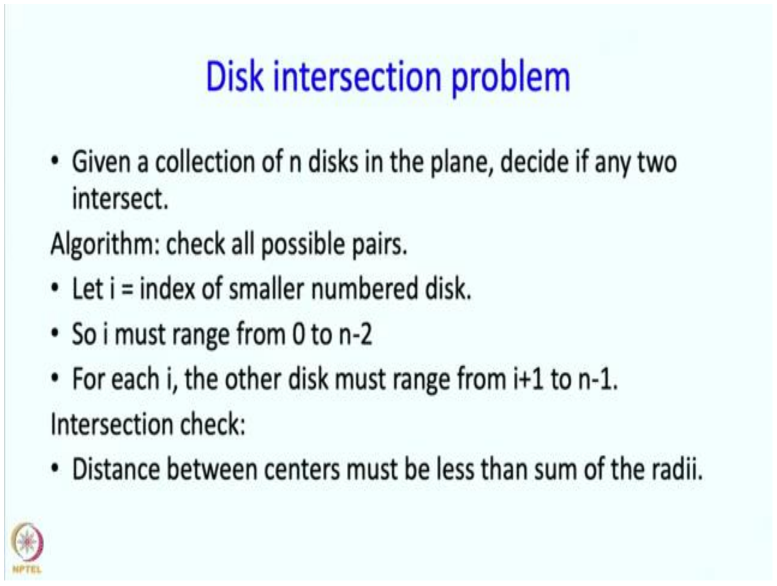
What we have discussed

- Different idea for solving taxi dispatch problem.
- New idea is better, copying of elements of `driverID` is avoided.
- Exercise: verify that the invariants indeed remain true after each customer or driver arrival.

Next: Disk intersection problem

Welcome back, in the last segment we saw a second idea for solving the taxi dispatch problem. In this segment, we are going to take another interesting use of arrays. And this is in a problem called disc intersection problem. So, what is the disc intersection problem?

(Refer Slide Time: 0:40)



Disk intersection problem

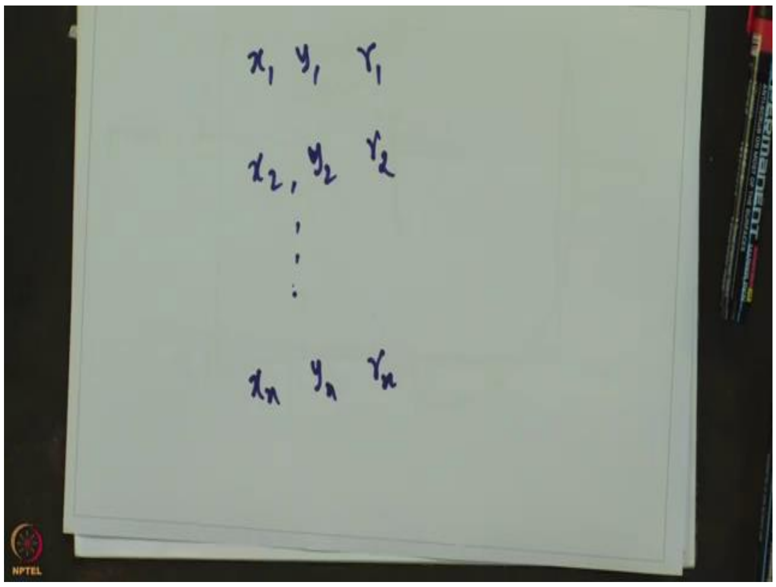
- Given a collection of n disks in the plane, decide if any two intersect.

Algorithm: check all possible pairs.

- Let i = index of smaller numbered disk.
- So i must range from 0 to n-2
- For each i, the other disk must range from i+1 to n-1.

Intersection check:

- Distance between centers must be less than sum of the radii.

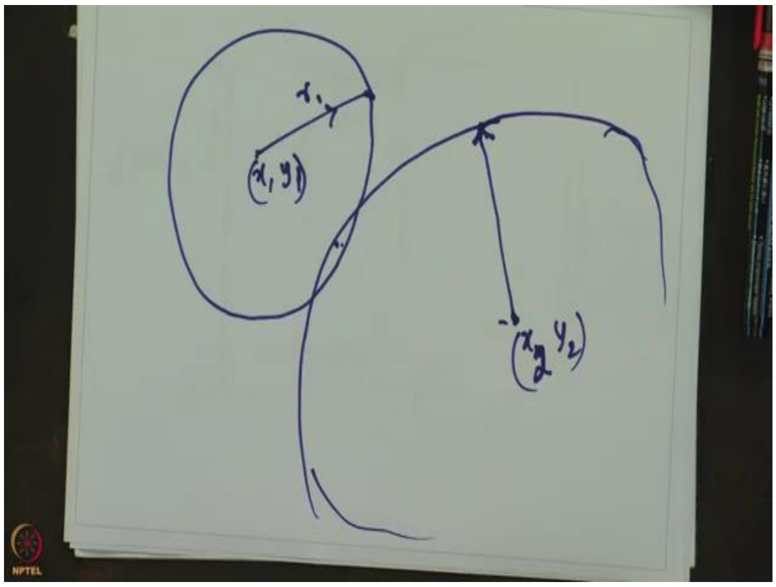


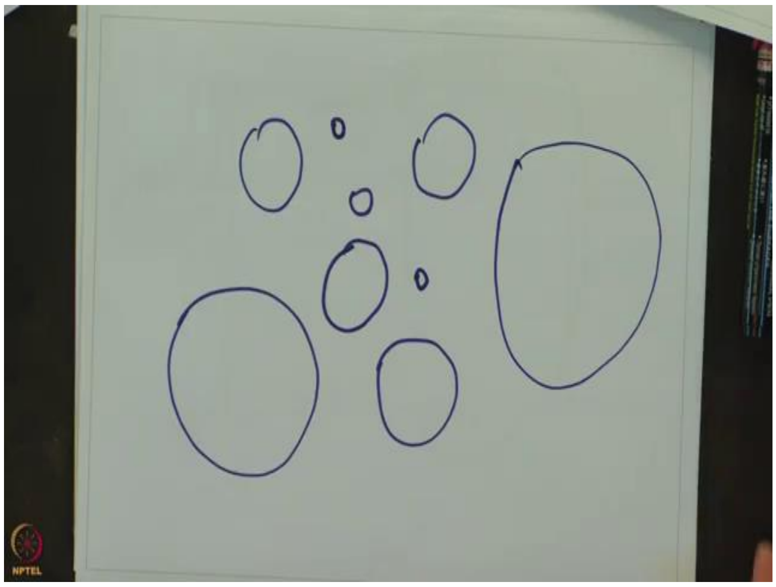
$x_1, y_1 \quad r_1$

$x_2, y_2 \quad r_2$

$\vdots$

$x_n \quad y_n \quad r_n$





So, given a collection of n discs in the plane, decide if any two intersect. So, before I get into the algorithm, I should explain to you what exactly is given as input. So what is given as input are, say x1, y1, which are the centre of the first disc, and r1, which is the radius off the first disc. x2, y2 are the co-ordinates of the centres of the first disc, and r2, which is the radius of the second disc, and so on until some xn, yn, rn.

So, you are given these n triples of numbers, the first two numbers give the co-ordinates of the centre of the disc, and the last number gives the radius of that disc. So, you are given 3n numbers, each triple representing a disc, and you want to know wither the discs are intersecting.

So, pictorially, if this is x1, y1, and if this is r1, then this is the first disc. Let us say this is x2, y2 the point x2, y2 and this distance is the radius. So, then the disc will look something like this, and if the radius is large enough then there is an intersection over here. So, in this case, if this happens for any pair of discs, then we are supposed to print an answer 1. On the other hand, if all the pairs, all the circles, they do not intersect at all, maybe they look something like this, they could be really small, they do not have to be of the same size of course. If they look something like this, then we are supposed to print out a 0 or a false, that there is no intersection. 1 or 2 if there is an intersection, and 0 if there is no intersection whatsoever.

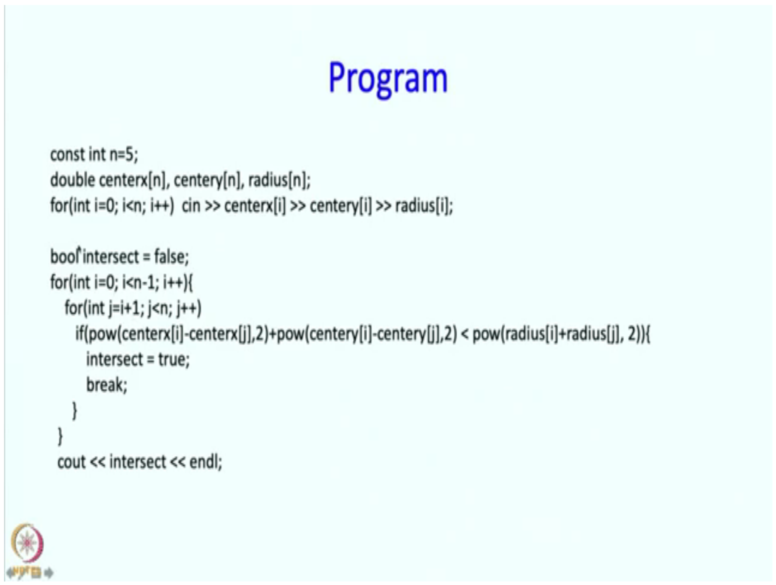
You could make the problem more general by asking, say, for one thing to print which discs intersect or say maybe, how many pairs of discs intersect? So, there are many variations possible, but let us just worry about the simplest. So now how do we solve this problem? Well we could check all possible pairs.

So, there are some n discs, and in these discs say 1 through n, how many pairs are there? Well there are n choose two pairs. We could check all such pairs. What do I mean by that, so let i denote the index of some disc, and let us say it is the smaller numbered disc in a pair. So what could other elements be, other discs in the pair possibly be.

So, first off all it is the smaller numbered disc in a pair, then i must range from 0 to n minus 2, because if it is n minus 1, then other disc will have to have a smaller number, and we are saying that i is the index of the smaller number disc within a pair. And what should j be. For each i, the other disc must range from i plus 1 to n minus1.

So, we want to find such i and such other disc numbers j, so that j goes from i plus 1 to n minus 1 for every i, which in turn goes from 0 to n minus 2. But that is basically it as far as deciding, how do you go over all the pairs? And the intersection check is reasonably simple, the distance between the centres must be less than the sum of the radii. So, we have all the information about the centres and about the radii and we just have to check that.

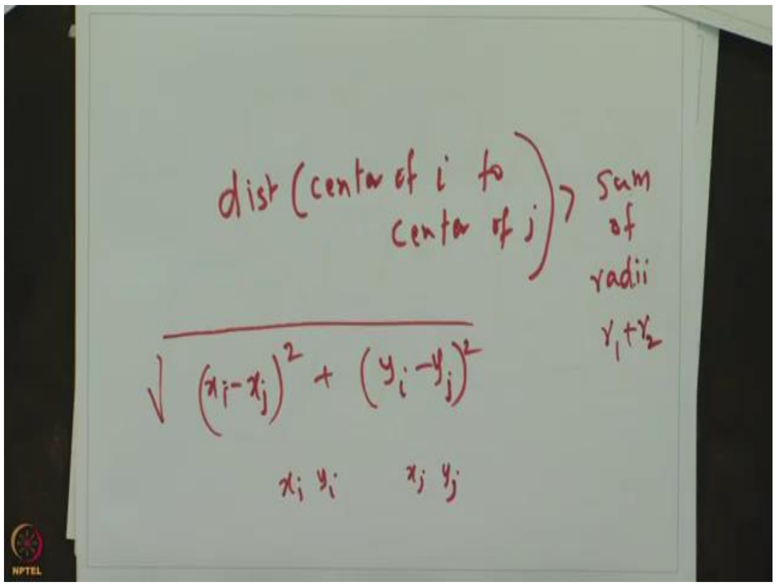
(Refer Slide Time: 5:12)



Program

```
const int n=5;
double centerx[n], centery[n], radius[n];
for(int i=0; i<n; i++)  cin >> centerx[i] >> centery[i] >> radius[i];

bool intersect = false;
for(int i=0; i<n-1; i++){
  for(int j=i+1; j<n; j++)
    if(pow(centerx[i]-centerx[j],2)+pow(centery[i]-centery[j],2) < pow(radius[i]+radius[j], 2)){
      intersect = true;
      break;
    }
 }
 cout << intersect << endl;
```


dist (center of i to center of j) > sum of radii
$r_1 + r_2$
$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$
$x_i\ y_i \quad x_j\ y_j$

So, here is the programme, so again I am going to use the name n, which is going to denote the number of discs. So, I am going to have an array to keep track of all the x co-ordinates of the centres of all the discs. An array to keep track of the y co-ordinate of all the centres of all the discs. And an array radius to keep track of the radii of all the discs.

Then I am going to read in this data, so that is all simple enough. Then is the code, which goes over all pairs of object, all pairs of discs. And this is really the interesting part in this. So, we have n objects, and how do you go over all pairs of them?

So first, before going over all pairs we are going to have a boolean variable intersect, which will say whether we have found the intersection or not, so originally, we have not found an intersection so it is going to be false. And then, our pair is going to be i j, but we said earlier than i has to lie between 0 and n minus 2, so i less than n minus 1 is exactly what we want, between 0 and n minus 2. I could have written this as i less than or equal to n minus 2.

So, in each iteration we well consider each such value of i, and therefore we are incrementing i at the end of the iteration. Now corresponding to every i, j is going to take values from i plus 1 to n minus 1, and therefore these parameters have been set in this manner.

And of course In each iteration, at the end of the iteration am going to increment j. And what do I do in each iteration. In each iteration, I have to check whether disc i and disc j are intersecting. So, how do I do my check?

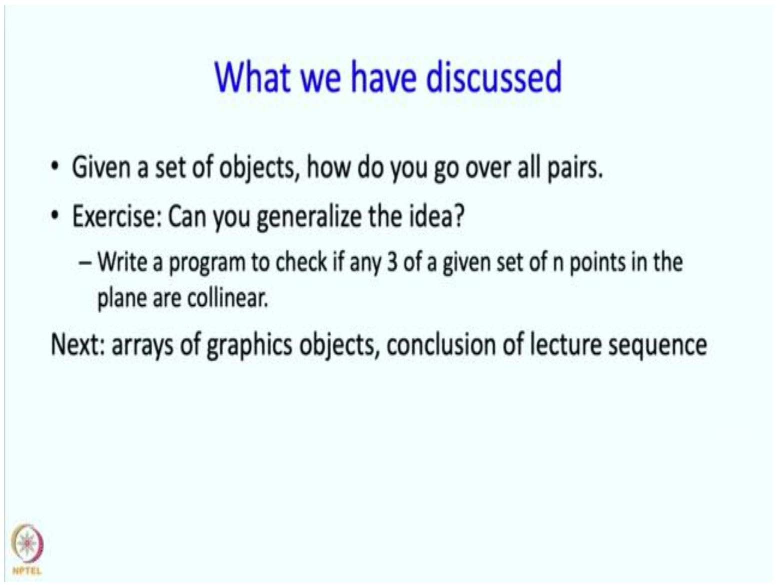
Well, my check is going to be done, normally I should check wither the distance from centre of i to centre of j is greater than the sum of the radii. But the distance from centre of i to centre of j is going to be something like or ((xi-xj)^2+(yi-yj)^2)^0.5. If the ith disc is at centred at xi, yi and the jth disc is centred at xj, yj.

So, I could say, that I need to calculate this and take the square root of it. Instead of that, I am going to square this distance. So, I will just get (xi-xj)^2+(yi-yj)^2. And if I square it, am going to take the sum of radius squared. So, I am going to check wither this is r1 plus r2, but instead of that I am also going to square that.

So, instead of doing a square root, I will just square, because squaring is easier. So, that is what is happening over here. So, this power of centre of i minus centre j squared is this xi minus xj square term. This is centre of y minus centre j power 2, so this is yi minus yj squared. And as I said I could have written square root over here, but instead of that I am taking radius of i plus radius j and squaring it.

So, if I find that the centres are closer than the sum of the radii, or the squares of the distance between the centre is less than sum of the radii squared, then I know that the intersection is happening and therefore I am going to set intersection equal to true. And the moment I find that at an intersection is happening I can break, because my variable intersect is only going to take values false or true, and I have found it true earlier, and I am have found the intersection and I must report that. So, if I run through this entire nested loop, saying that, and do not find an intersection, then my intersection will remain false. So, in any case when I come to this point, either through the break or after going through all the iterations, intersect will tell me whether discs intersect or not. So, that is what I am going to print and that is going to be the end of my programme. And of course I have to put this in a main programme and I have to include header files and things like that.

(Refer Slide Time: 10:36)



What we have discussed

- Given a set of objects, how do you go over all pairs.
- Exercise: Can you generalize the idea?
  - Write a program to check if any 3 of a given set of n points in the plane are collinear.

Next: arrays of graphics objects, conclusion of lecture sequence

Alright, So what have we discussed in this? We have discussed, given a set of objects, how do you go over all pairs? So, this is an interesting operation that happens from time-to-time. And as an exercise I would like you to generalise the idea, so specifically, write a programme to read in n points, co-ordinates of n points and check if any 3 of them are collinear.

So, you know how to figure out wither 3 points are collinear. You just have to calculate slopes and things like that. And while doing this calculation, I will suggest that you avoid division. Because division will create problems, if the co-ordinates have, if the points have the same x co-ordinate or the same y co-ordinate, and if you are calculating the slope, you are dividing by the difference of the by co-ordinates and that happens to be 0.

So, instead of that you can do a little bit multiplication rather than some division, you cross multiply the terms and you get the division.

Alright so, that is basically it. In the next segment, we are going to talk about, how you deal with whether we can have arrays of graphic objects and we will also conclude this entire lecture sequence. So, we will take a short break.