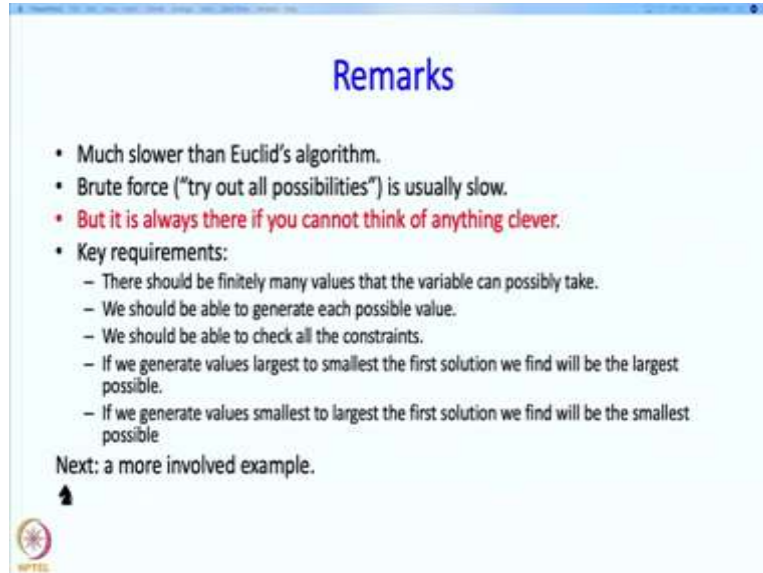


An Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture No. 09 Part - 2
Loops in various applications
Finding Pythagorean Triples


(Refer Slide Time: 00:31)



Remarks

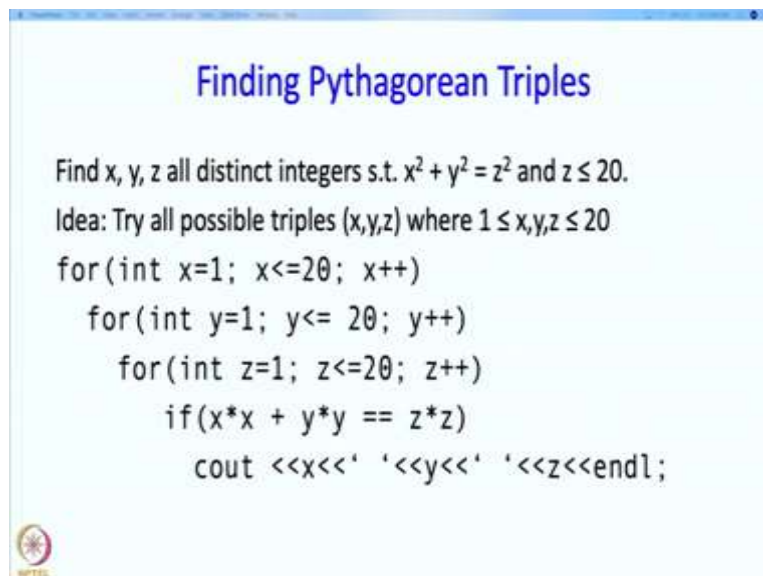
- Much slower than Euclid's algorithm.
- Brute force ("try out all possibilities") is usually slow.
- **But it is always there if you cannot think of anything clever.**
- Key requirements:
 - There should be finitely many values that the variable can possibly take.
 - We should be able to generate each possible value.
 - We should be able to check all the constraints.
 - If we generate values largest to smallest the first solution we find will be the largest possible.
 - If we generate values smallest to largest the first solution we find will be the smallest possible

Next: a more involved example.



Welcome back, in the previous segment we looked at how to solve GCD using brute force search. We said that this is slower than Euclid's algorithm, but we said that brute force search will be useful if we do not have clever algorithms for some problem provided the number of possible candidate solutions that we need to consider are finite in number.

(Refer Slide Time: 00:51)



Finding Pythagorean Triples

Find x, y, z all distinct integers s.t. $x^2 + y^2 = z^2$ and $z \leq 20$.

Idea: Try all possible triples (x, y, z) where $1 \leq x, y, z \leq 20$

```
for(int x=1; x<=20; x++)
  for(int y=1; y<= 20; y++)
    for(int z=1; z<=20; z++)
      if(x*x + y*y == z*z)
        cout <<x<<' '<<y<<' '<<z<<endl;
```

Now, we are going to take another example of this, and this example is finding Pythagorean triples. So the problem as we discussed is find x, y, z such that $x^2+y^2=z^2$. And z is smaller than 20, and of course x, y, z are positive integers. The idea we mention was that let us try out all possible triples x, y, z where, x, y, z lie between 1 and 20. So, how do you generate such triples, that is the natural question. And that actually very easily done.

So we are going to have a loop which is going to have control variable or index variable x which will go between 1 and 20. Nested inside that will have a loop with control variable y , which will go between 1 and 20 as well. And then we will have a loop with control variable z again ranging between 1 and 20.

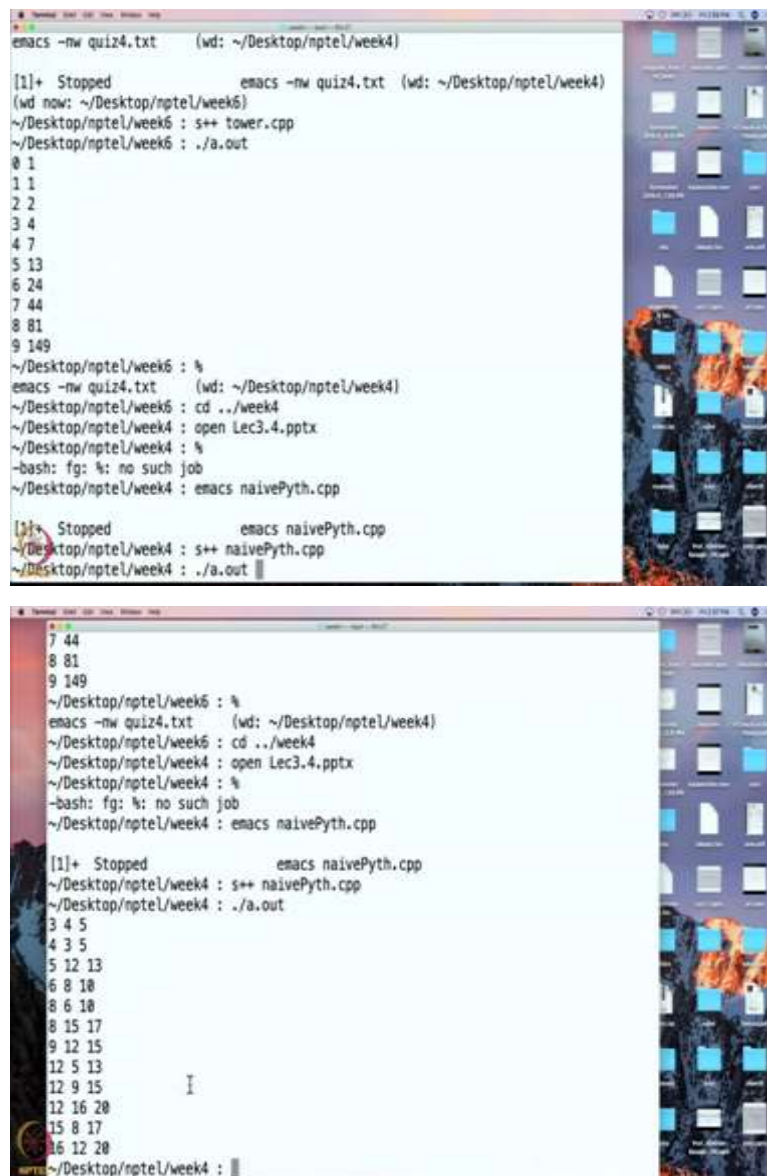
So now, whatever we put over here, we can use the variables x, y, z and in different iterations, they will assume all the possible 8000 triples of the form x, y, z where, x, y, z are integers between 1 and 20. So, if you want to check for all possible triples, the check has to be put over here, and that is the check. We want to know whether $x^2+y^2=z^2$. And if it is we want printout that triples. So, that is it, that is the program.

(Refer Slide Time: 02:46)



So, let me show you a demo of it, have called it naivePythagoras dot CPP and you will see the reason for it.

(Refer Slide Time: 03:26)



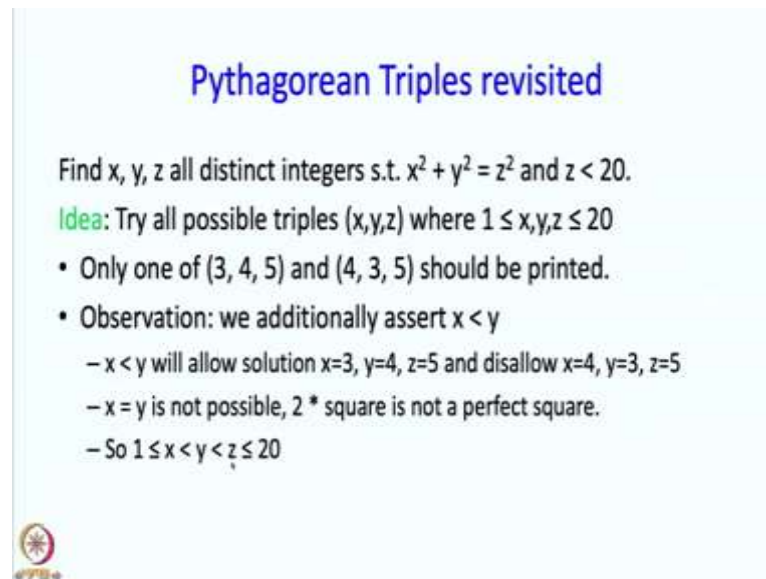
```
emacs -nw quiz4.txt (wd: ~/Desktop/nptel/week4)
[1]+ Stopped emacs -nw quiz4.txt (wd: ~/Desktop/nptel/week4)
(wd now: ~/Desktop/nptel/week6)
~/Desktop/nptel/week6 : s++ tower.cpp
~/Desktop/nptel/week6 : ./a.out
0 1
1 1
2 2
3 4
4 7
5 13
6 24
7 44
8 81
9 149
~/Desktop/nptel/week6 : %
emacs -nw quiz4.txt (wd: ~/Desktop/nptel/week4)
~/Desktop/nptel/week6 : cd ../week4
~/Desktop/nptel/week4 : open Lec3.4.pptx
~/Desktop/nptel/week4 : %
-bash: fg: %: no such job
~/Desktop/nptel/week4 : emacs naivePyth.cpp
[1]+ Stopped emacs naivePyth.cpp
~/Desktop/nptel/week4 : s++ naivePyth.cpp
~/Desktop/nptel/week4 : ./a.out

7 44
8 81
9 149
~/Desktop/nptel/week6 : %
emacs -nw quiz4.txt (wd: ~/Desktop/nptel/week4)
~/Desktop/nptel/week6 : cd ../week4
~/Desktop/nptel/week4 : open Lec3.4.pptx
~/Desktop/nptel/week4 : %
-bash: fg: %: no such job
~/Desktop/nptel/week4 : emacs naivePyth.cpp
[1]+ Stopped emacs naivePyth.cpp
~/Desktop/nptel/week4 : s++ naivePyth.cpp
~/Desktop/nptel/week4 : ./a.out
3 4 5
4 3 5
5 12 13
6 8 10
8 6 10
8 15 17
9 12 15
12 5 13
12 9 15
12 16 20
15 8 17
16 12 20
~/Desktop/nptel/week4 :
```

So, this is our program exactly what we have written earlier, so let me compile and run it. So, it has printed, all the triples, let me move it out a little, okay, it has printed all the triples, 3, 4, 5 4, 3, 5 5, 12, 13 okay these are all the triples that it found, satisfied the property, that the sum of first number squared plus the second number squared equals the third number square, you can check it out. So, first of course we have the familiar triples, 3 squared plus 4 squared equal to 5 square.

Now, is this a satisfactory list? Well it is not, why? Because, this triple and this triple, we would like to think of as the same triple. Right, I mean if we want to send this to somebody, they will say look why are sending me this twice? I know, these numbers the order is not really important in these numbers. So, and of course is that has happened in many places. So, 8, 15, 17 is there, 15, 8, 17 is there. So the question is, can we avoid this? And yes, we can and we are going to do that in just a minute.

(Refer Slide Time: 05:05)



Pythagorean Triples revisited

Find x, y, z all distinct integers s.t. $x^2 + y^2 = z^2$ and $z < 20$.

Idea: Try all possible triples (x, y, z) where $1 \leq x, y, z \leq 20$

- Only one of $(3, 4, 5)$ and $(4, 3, 5)$ should be printed.
- Observation: we additionally assert $x < y$
 - $x < y$ will allow solution $x=3, y=4, z=5$ and disallow $x=4, y=3, z=5$
 - $x = y$ is not possible, $2 * \text{square}$ is not a perfect square.
 - So $1 \leq x < y < z \leq 20$

So we are going to look at this problem again, and make a modification which will allow us to generate similar triples only once. So this is our problem again, and this was our idea, and this was our problem. So, we want 3, 4, 5 to be printed, and not 4, 3, 5 or maybe the other way around. But we could not say that look, why do not we print this in ascending order, rather than this looks a little funny.

So, we only want one of these things to be printed, so how can we enforce that? Well here is the idea. So, in addition to the constraints which are already there, in the problem, we will additionally assert that x should be less than y . What does this do? So, this will allow, this solution because this is x and this is y , and x is instead less than y , but this solution will be avoided. Because this is not less than this. So that is the idea, so we are going to have the previous program, but in the previous program we are somehow going to enforce the condition that x is less than y there are various ways of enforcing this condition, but we are going to pick a particularly nice one.

And by the way note that, x equal to y does not have to be considered, why? Because, if we look at $x^2 + y^2$, then that if they are equal, would be $\sqrt{2}$ times x squared and that cannot

be a perfect square. So it cannot equal any z square. So, x equal to y does not have to be considered, we do not want x greater than y , and therefore, we only be asserting x less than y .

And in fact, you can see that the conditions really could be something like, this. Because we know we are asserting that y is bigger than x , and of course z has to be bigger than y . because its square is strictly more than the square of y and x . So, let us see how we can assert these conditions somewhat indirectly.

(Refer Slide Time: 07:17)



```
File Edit Options Buffers Tools C++ Help
#include <simplecpp>

main_program{
  for(int x=1; x<=20; x++)
    for(int y=1; y<= 20; y++)
      for(int z=1; z<=20; z++)
        if(x*x + y*y == z*z)
          cout <<x<< ' ' <<y<< ' ' <<z<<endl;
}
```

So, let us look at the code, so here is the code. So, the first loop is exactly as before, x goes from 1 to 20. The next loop however, has y skipping all the values including whatever the value x is currently taking and smaller values. So, y only starts at x plus 1, and it goes on till 20, and z similarly, skips all the even, all the smaller values. Because we know that, has to be bigger than y . And so, it starts off at y plus 1, and goes on till 20. And the condition is still the same. If x squared plus y squared equal to z square, then we print it out. So, let see this.

(Refer Slide Time: 08:05)

```
emacs -nw quiz4.txt (wd: ~/Desktop/nptel/week4)
~/Desktop/nptel/week6 : cd ../week4
~/Desktop/nptel/week4 : open Lec3,4.pptx
~/Desktop/nptel/week4 : %
-bash: fg: %: no such job
~/Desktop/nptel/week4 : emacs naivePyth.cpp

[1]+ Stopped emacs naivePyth.cpp
~/Desktop/nptel/week4 : s++ naivePyth.cpp
~/Desktop/nptel/week4 : ./a.out
3 4 5
4 3 5
5 12 13
6 8 10
8 6 10
8 15 17
9 12 15
12 5 13
12 9 15
12 16 20
15 8 17
16 12 20
~/Desktop/nptel/week4 : %
emacs naivePyth.cpp

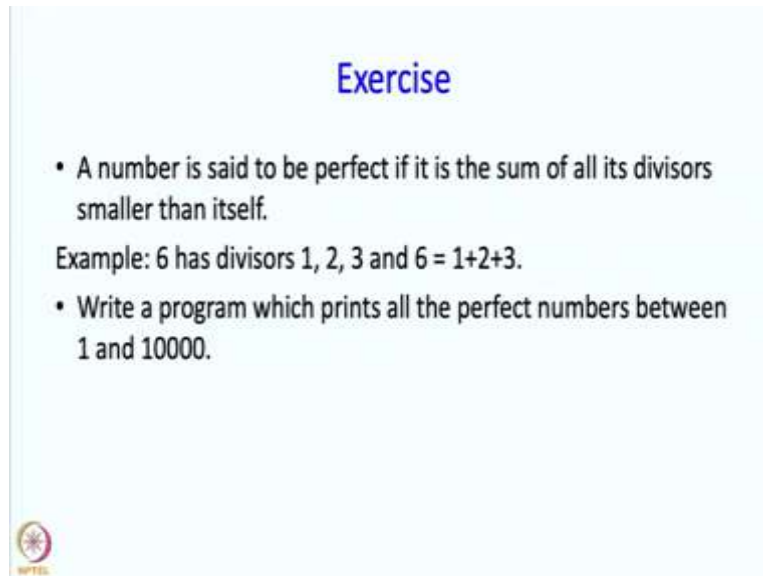
[1]+ Stopped emacs naivePyth.cpp
~/Desktop/nptel/week4 : s++ p||

~/Desktop/nptel/week4 : s++ naivePyth.cpp
~/Desktop/nptel/week4 : ./a.out
3 4 5
4 3 5
5 12 13
6 8 10
8 6 10
8 15 17
9 12 15
12 5 13
12 9 15
12 16 20
15 8 17
16 12 20
~/Desktop/nptel/week4 : %
emacs naivePyth.cpp

[1]+ Stopped emacs naivePyth.cpp
~/Desktop/nptel/week4 : s++ pyth.cpp
~/Desktop/nptel/week4 : ./a.out
3 4 5
5 12 13
6 8 10
8 15 17
9 12 15
12 16 20
~/Desktop/nptel/week4 : ||
```

So, let us compile it, and run it, so that is it, it has indeed done the right thing, it has printed every triple just once, and so, 3, 4, 5 is printed as 3, 4, 5 but not as 4, 3, 5, which was happening earlier.

(Refer Slide Time: 08:47)




Exercise

- A number is said to be perfect if it is the sum of all its divisors smaller than itself.

Example: 6 has divisors 1, 2, 3 and $6 = 1+2+3$.

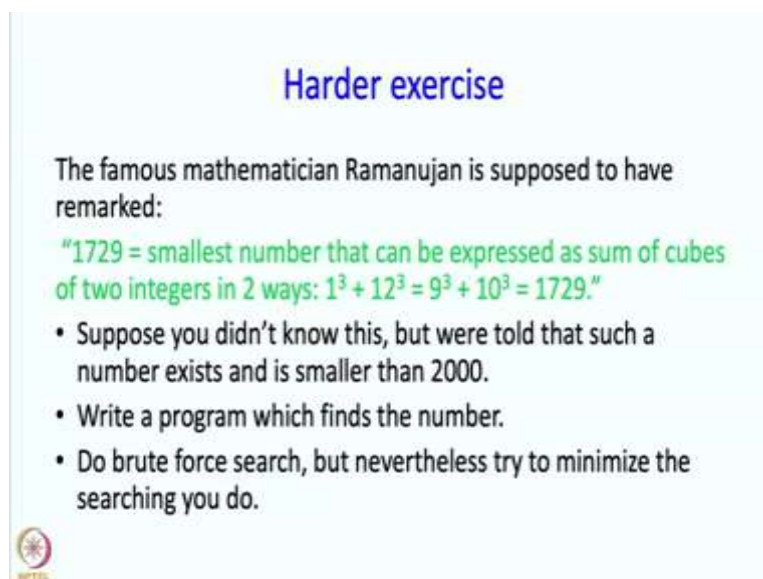
- Write a program which prints all the perfect numbers between 1 and 10000.



So, let us get back to our presentation. So, I want to give you some exercises. So, the first exercise is dealing with something called, perfect numbers. A number is said to be perfect if the sum of all its divisors smaller than itself. So 6 for example, has divisors 1, 2, 3 smaller than itself, and 6 is indeed the sum of those divisors. So therefore, 6 is said to be perfect.

The problem that i am asking you, to write a program for is, to find or print all the perfect numbers between say 1 and 10000. So, you have to go through all the numbers between 1 and 10000, and check if they are perfect, and if you find that a particular number is perfect, just print it.

(Refer Slide Time: 09:39)




Harder exercise

The famous mathematician Ramanujan is supposed to have remarked:

"1729 = smallest number that can be expressed as sum of cubes of two integers in 2 ways: $1^3 + 12^3 = 9^3 + 10^3 = 1729$."

- Suppose you didn't know this, but were told that such a number exists and is smaller than 2000.
- Write a program which finds the number.
- Do brute force search, but nevertheless try to minimize the searching you do.

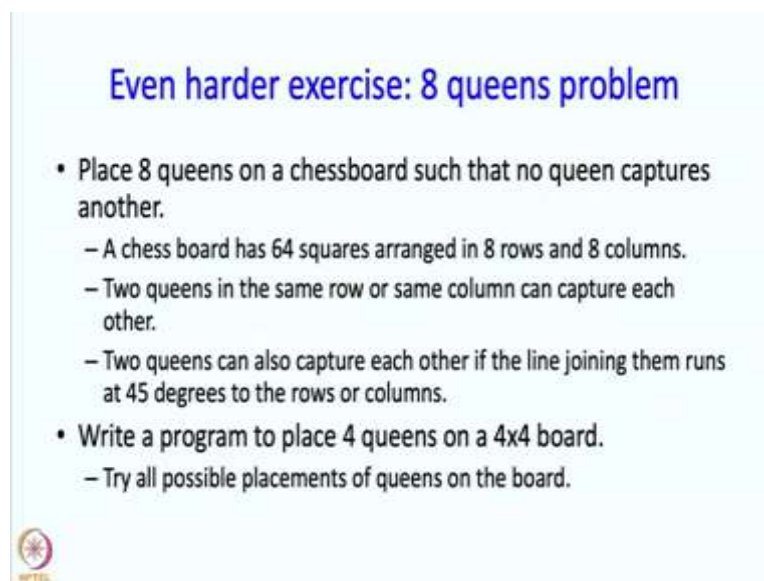


A harder exercise relates to something that, the famous mathematician Ramanujan is supposed have said. So he said, sometime that 1729 is the smallest number that can be expressed as sum of cubes of two integers in 2 ways. In particular, 1 cube plus 12 cube is 1729. Because 1 cube is 1, 12 cube is 1728, 9 cube is 729 and 10 cube is 1000 and therefore, 9 cube plus 10 cube is also 1729. So this is an interesting fact and you may wonder suppose, I did not know this, could I have figured it out?

So, that is where, what our problem is going to be about. So, I am going to assume that suppose, you did not know this, and you were told however, that such a number exists and is smaller than 2000. Can you find it? And in fact, you could ask look I want the smallest such number, so that is the problem. So, write a program which finds the smallest number, and you know and assume, you are assuming that it is smaller than 2000. So, the smallest number which can be expressed as sum of cubes of two integers in 2 different ways.

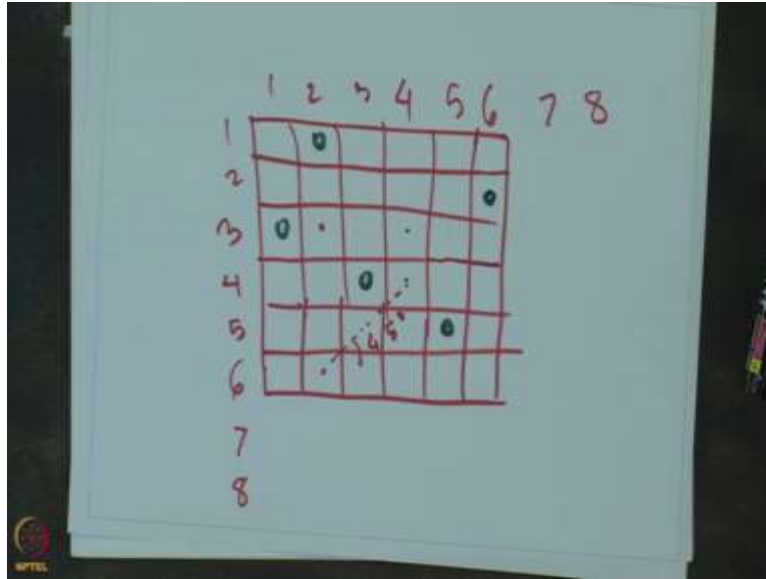
So, you are expected to do brute force search. So we will have to figure out exactly how to organize that brute force search. So we have expressed as a constraint satisfaction problem. So, think about it, it will as I said, it is a slightly harder exercise. So the number that you want to find is certainly one of the variables that you want. But what about those numbers 1 cube, 12 cube, 9 cube, 10 cube. So, how do we exactly get them into our constraints? So that is what you make to figure out.

(Refer Slide Time: 11:59)



Even harder exercise: 8 queens problem

- Place 8 queens on a chessboard such that no queen captures another.
 - A chess board has 64 squares arranged in 8 rows and 8 columns.
 - Two queens in the same row or same column can capture each other.
 - Two queens can also capture each other if the line joining them runs at 45 degrees to the rows or columns.
- Write a program to place 4 queens on a 4x4 board.
 - Try all possible placements of queens on the board.



I want to list out an even harder exercise but which may challenge some of you and which some of you may like, and this is the so called, 8 queens problem. So, this problem asks you to place 8 queens on a chessboard such that, no queen captures another. So, for those of you, who do not know chess, here is a quick explanation. A chessboard has 64 squares arranged in 8 row and 8 columns.

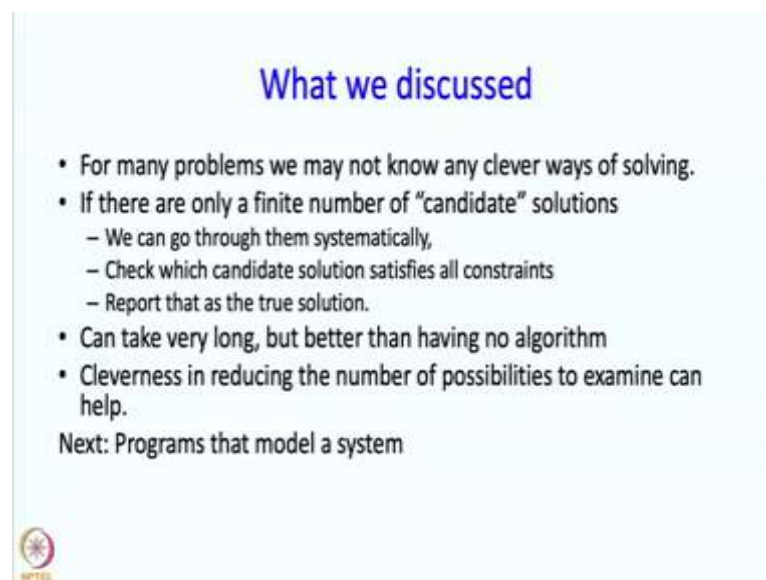
So, it looks like this, so there are going to be 8 rows and 8 columns, whatever this is not 8 but there should be 8 rows and 8 columns. So, how many is this, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6. So, if you had 7 and 8, and 7 and 8, then it would be a chessboard. Now, that is what a chessboard is and this says if 2 queens are in the same row or in the same column then, they will capture each other. So, if we have queen over here and a queen over here, then they will capture each other. So, if you place a queen over here, then you should not be placing the queen over here. That is what the constraint tells you. And similarly if you place a queen over here, you cannot be placing queen in any square in the same row. So, you cannot place the queen over here. Because, they would capture each other, and you want place queens, such that they do not capture each other, but this is not only way queens capture each other. So queens capture each other, even if they are sort of on the same diagonal or since the notion of diagonal only refers to the principle diagonal which is from corner to corner, we can say the same thing in different manner. So, suppose there are 2 queens, so these 2 queens, and if we draw the line joining them, then this line is at 45 degrees angle with respect to the rows and the columns. So, even then they can capture each other so, this is also not allowed. So for example, you could have a placed, if you want to place 3 queens, you could place one over here, another one over here, the third one over here maybe.

So now, no queen is capturing each other, may be even a fourth one over here. So, in this manner you could have place the queens. And turns out that in this case, you can place the 6 queen over here. So, you have place 6 queens on a 6 by 6 chessboard, but you want to do this on in 8 by 8 chessboard.

Well, that is the original problem, but we do not want get carried away with having too many variables and writing too much code, and in particular the spirit of the problem is still the same even if you are looking at the 4 by 4 board. So, if you want place 4 queens on a 4 by 4 board, how will you do it? You want to write a program which will do it for you. So basically the idea is that you try out all possible placements of the queens on the board. So, you will have to figure out, how do you represent this problem using numbers, and suppose you say that this queen is placed on the square, and other queen is placed on this square. What does it mean for the queens to not capture each other or capture each other?

So the constraints here are expressed in terms of this geometry, but you have to get them into arithmetic. So you want that constraints to be in terms of numbers. So that is what makes this exercise harder. But once you do that again after that you are not expected to be clever. After that you are just supposed to try out all possible ways of placing the queens.

(Refer Slide Time: 16:30)



The slide has a light blue background. At the top center, the title "What we discussed" is written in a blue, sans-serif font. Below the title is a bulleted list of five points. The first point is "For many problems we may not know any clever ways of solving." The second point is "If there are only a finite number of 'candidate' solutions", followed by three sub-points: "We can go through them systematically," "Check which candidate solution satisfies all constraints", and "Report that as the true solution." The third point is "Can take very long, but better than having no algorithm". The fourth point is "Cleverness in reducing the number of possibilities to examine can help." Below the list, the text "Next: Programs that model a system" is written. In the bottom left corner, there is a small circular logo with a star and the letters "NPTEL" below it.

What we discussed

- For many problems we may not know any clever ways of solving.
- If there are only a finite number of "candidate" solutions
 - We can go through them systematically,
 - Check which candidate solution satisfies all constraints
 - Report that as the true solution.
- Can take very long, but better than having no algorithm
- Cleverness in reducing the number of possibilities to examine can help.

Next: Programs that model a system

Alright, so, what did we discussed in this segment? Well, these 2 segments actually. So, what we discussed was that, for many problems we may not know any clever ways of solving them. But, if there are only a finite number of candidate solutions, then we can go through them systematically. We can check which candidate solutions satisfies all constraints, and we report that as the real solution.

This can take very long, but it is better than not having an algorithm. But, of course there is room for cleverness. We may have to do things like, let that enforce additional constraints, say x is less than y or something like that. And notice by the way that that actually also reduces the amount of searching we have to do. So, some any kind of cleverness that we may, that we are capable of in reducing the number of possibilities always helps, it allows you to solve bigger problems in the same amount of time.

So, that finishes this topic of brute force search, and now I am going to take a break, and after the break, we will discuss how to write programs that model a physical system.