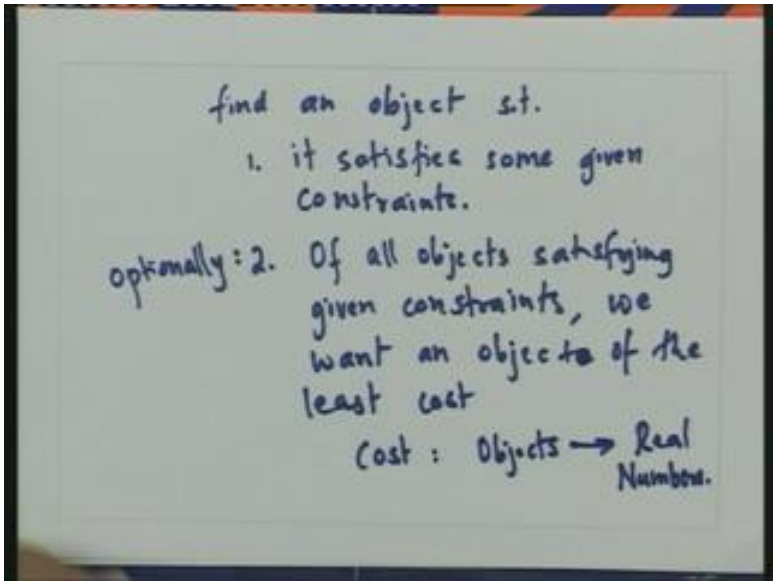
Design and Analysis of Algorithms
Prof. Abhiram Ranade
Department of Computer science Engineering
Indian Institution of Technology, Bombay

Lecture - 16
Combinatorial Search and Optimization – I

Welcome to the course on design and analysis of algorithms. Our topic for today is combinatorial search and optimization. The general problem that we are going to address is something like the following.

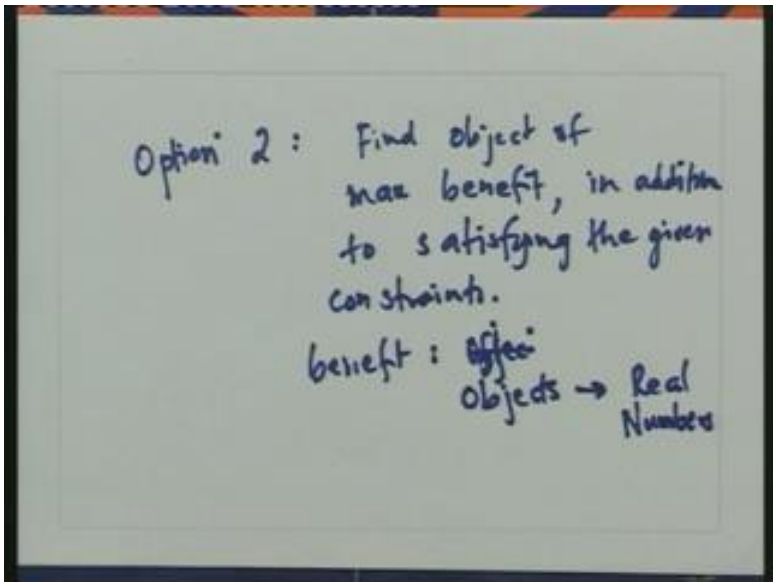
(Refer Slide Time: 01:27)



So, we want to find an object and this will usually be a discrete object say a set of numbers or a graph. And since it is a discrete objects object when we talk about a set of number. We will usually mean set of integers. So, even perhaps set of zero and values. And this object must satisfy certain given constraints.

Find us find an object such that first it satisfies some given constraints. And further more optionally then might be a second requirement that of all objects satisfying those constraints. We want an object of the lease cost, where the cost is a function defined on the objects? Is a function from the space of objects to real numbers?

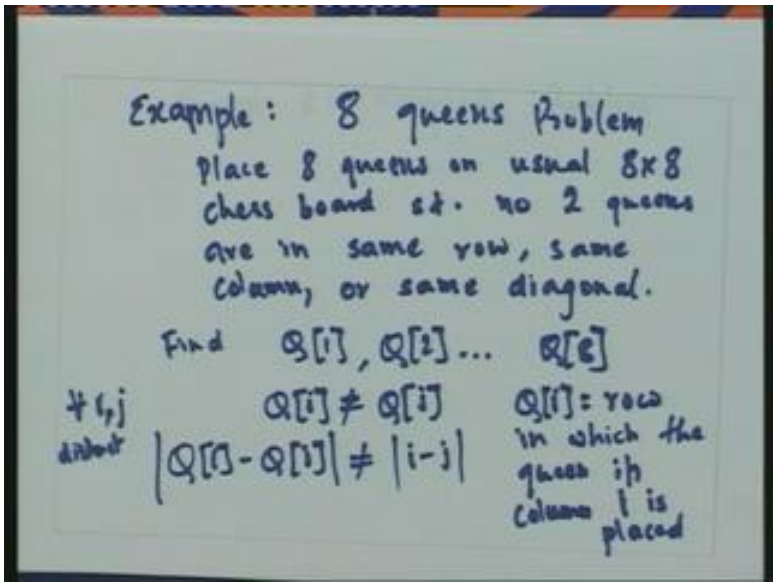
(Refer Slide Time: 03:24)



Optionally, we might also have an alternate definition in which instead of saying we want least cost. So option 2, it could be stated differently instead of asking for the object of least cost we want an object of maximum benefit. Find object of maximum benefit of course in addition to satisfying the given constraints.

Benefit is also a function from the objects to real numbers. Given note initiately that these are not really too distinct for simulations. But, we can set benefit equal to negative of the cost. And then instead of maximizing the benefit it is the same as minimizing the cost. So, this is an abstract definition and let me give you a few examples initiately.

(Refer Slide Time: 05:09)



So, let me first start with the familiar example in which there is no objective function given. So, this cost function for example, or this benefit function is also often called an objective function. And so in this first example, we are not given an objective function, but here only given constraints.

So, this example that I am going to talk about this is so called 8 queens problem. Probably, most of what this is so, we are given usual chess board. And we want to place queens on it. So, place 8 queens on usual which is an 8 by 8 chess board such that no 2 queens capture each other or no 2 queens are in the same row same column or same diagonal.

We remainder by this is a problem in the sense, we have described. Well, we can always modulate as problem on numbers an integer. So, we can think of this as find Q 1 Q 2 to Q 8. Let us say Q i represents where in column i the ith queen is going to be placed. So, find these Q such that Q of i is not equal to Q of j. So, interferic Q of i as equal to the row in which the queen in column i is placed.

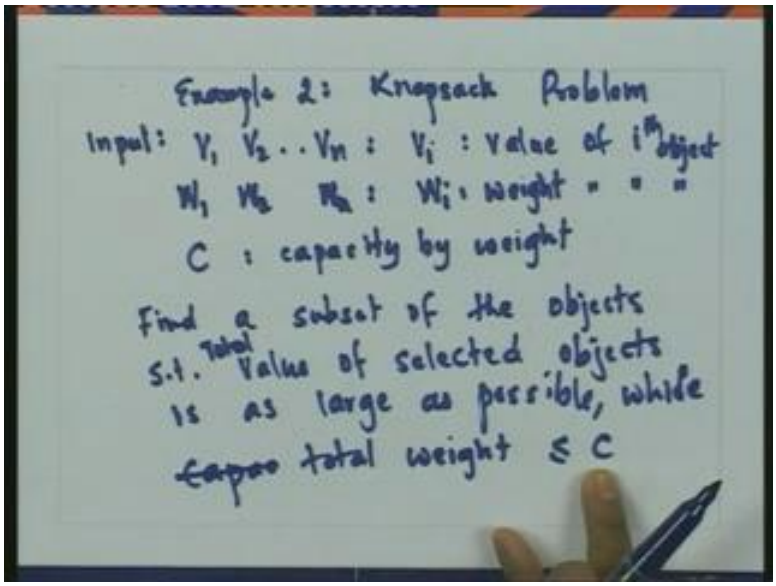
So, this condition simply says that the green which get placed in the row in column i. Does not have as it is row number value, which is the same as the value in which value which the queen in the jth column is placed. So, in other words the ith queen in the jth queen are not placed in the same row. And we will assume that the ith queen is placed in the ith column and the jth queen is placed in the jth column.

But this is not all. So, we have already taken care of the first condition that they are not placed in the same row. Well, they are not placed in the same column. Because, we are only defining 8 variables and we are saying that k of i is the position. The queen which is going to a fixed in that which is going to be placed in the ith column and we are only giving the row.

Now, we need to take care of the condition that they are not placed in the same diagonal. And this condition is easily taken care of by asserting that Q of i minus Q of j. This is simply the distance along the rows the vertical distance. The distance between the rows and if I take the absolute value it will actually be an unsigned and this should not be equal to i minus j. This is true for all i j distinct.

So, we have now formulated the 8 queens problem as the problem of finding a sequence of numbers Q 1 through Q 8 such that they satisfy these 2 conditions. Well actually these are not 2 conditions between these conditions which apply for all i and j. So, this is say, whatever 8 times 8 upon 2 conditions this is also 8 times 8 upon 2 conditions.

(Refer Slide Time: 09:15)



Of course, this is really a toy or game problem, but we can do we can have more serious problems as well. So, let me give a few of those. The second problem that we are going to look at is so called knapsack problem. In this case in fact, we are going to have an object function as well.

((Refer Time: 05:09)) So in this case, our question was to their exist such numbers. That is satisfy these conditions. But, it did not say that if there are several such 8 tuples of number which one we are interested in. We said the just find any such set or any such sequence.

Here, also going to have an objective function. So here as, we are going to have an input and the input consist of say numbers V 1 V 2 all the way to V n. And thing of V i as the value in rupees say of ith object. So implicit is that, there is a set of n objects over here each object has a value and that value is known to us.

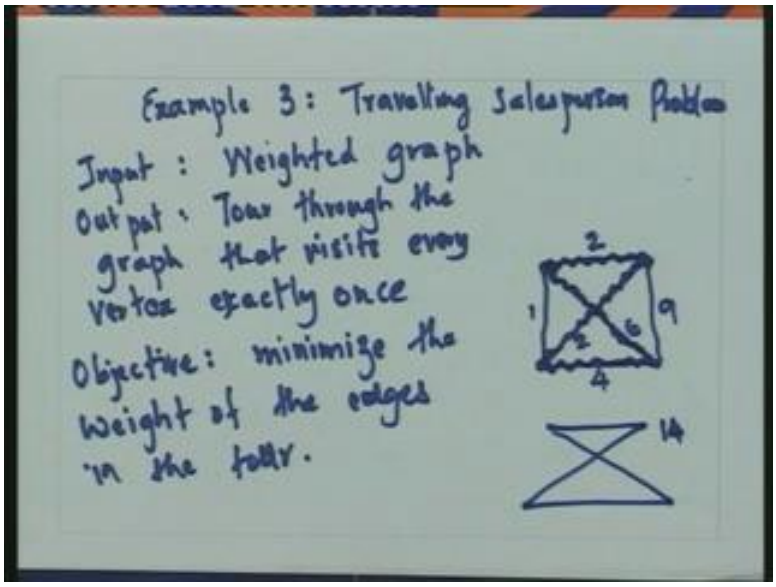
In addition each object also has a weight. So, say W 1 W 2 all the W8 W n and say such that W i is the weight saying in kilos in kilo grams of the ith object. And imagine that, we are given a knapsack which has capacity C, C is weight, capacity by weight. The problem should be this manner; we want to determine which of these objects. We should place in the knapsack such that we do not exceed the capacity of the knapsack and such that we are picking up objects of maximum value.

So, we want to find a subset of the object such that value of objects. The total value of selected objects is as large as possible, while capacity while total weight is less than or equal to 0. So although, this problem looks like it arises in the context of objects and values and weights it can also arise and other context. So maybe, there are jobs and their profits associated with the job. And we want to decide which job to select something like that.

So, again there is a constraint over here well first of all there is a combinatorial object that we need to take which is this, which is a subset of the given objects or crops call these objects items. So, we want to pick a subset of these given items. And there is a constraint on them that the weight of the selected items should not be bigger than C that the constraints. And then there is also an objective function.

The objective function is that there value should be as large as possible. So, this is a maximization problem and the objective function given to us can be thought of is a benefit function. Because after all, if we pick as many objects of large value. We are maximizing the benefit to ourselves.

(Refer Slide Time: 13:32)



Let me give you one more example, this is another classic example this is so called travelling sales person problem. The input over here is a weighted graph. And the output is a tour through the graph. That visits every vertex exactly once and objective we want to minimize the weight of the edges in the tour.

The names come from the association that we can think of this weighted graph as representing a map. Map of a city for example, I am sorry a map of a country or a map of the world vertices represent towns and edges represents roads and weight represent the length of the edges. So, imagine that there is a sales person who wants to visit every city for his for selling his products.

However, he wants to make sure that he travels as little as possible. So, this is a problem that he might he or she might want to solve she might. So, he would like to find the tour that visits every city exactly once such that the total distance covered along all the edges is as small as possible.

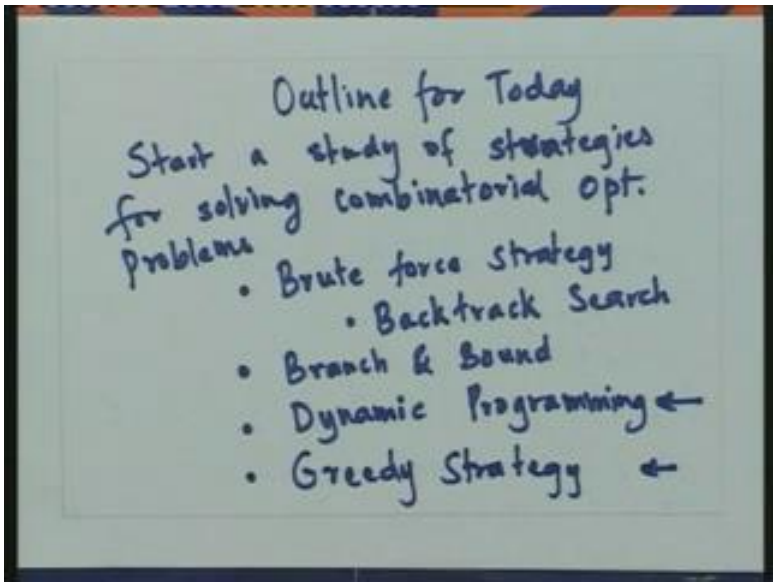
Again on one hand this may seem like a problem relating to sales persons and cities and roads. But, this problem arises in many many contexts. For example this problem arises in robotics; this problem arises in computation of biology and many other fields. Before carrying on let me do an example of this problem just to clarify what is meant over here.

So, let us say that we have four cities. These are our four cities and so this is our graph. So typically, we will have a complete graph over here the graph I am going to draw is going to be a undirected graph. But, this problem makes sense in the directed case as well, so this is the graph. And let us say there are weights over here. So for example, this weight could be 1 this could be 2 say 9. This could be 6 this could be 2 and say this could be 4.

So, there are various ways in which the sales man could tour through this cities. And so, for example, 1 way is this outer most this outer tour, what is the cost of the? Well the cost of that is 1 plus 2 plus 9 plus 4. So that is 16, here is a better tour. So for example, something like this. So, this figure of 8 this tour, what is the cost of that? That is 2 plus 6 plus 2 plus 4. So that is 14. So, this is in fact, a better to over than just the outer tour.

And in fact, you should be able to check that this is the best possible tour. In any case the pointers we want to we want to make sure that we have found the tour which has the minimum such cost at the minimum such length if you like. These are only three examples of the kinds of problems we have in mind. And in fact, later on in the course will be considering more will be considering more problems of a similar kind.

(Refer Slide Time: 18:10)



But let me now state the topic for today. The idea today is to look at we want to start a study of strategies for solving combinatorial optimization problems. Combinatorial optimization and search problems, I may drop the search occasionally, but that is what that is imply. Today I am going to talk about a few basic strategies one is so called well one is based so called backtrack search, but it really a sort of a Brute force search strategy.

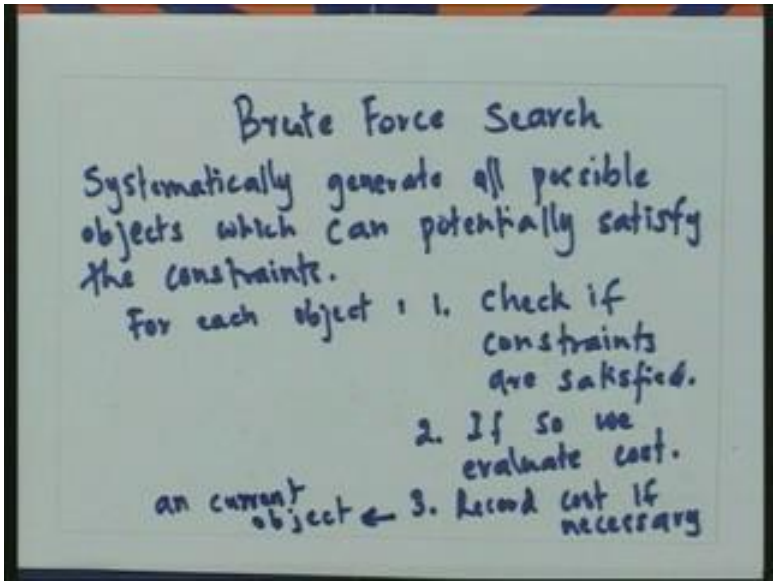
So, under this we will be looking at what is called backtrack search. Then, we look at the strategy called branch and bound. Will also be looking at a strategy called dynamic programming and in fact, it is this strategy that we want to study the most. Because, this is the strategy which we can analyze and very often it gives us very fast algorithms as well.

But, it requires as considerable about among the diff level analysis. But, I am wanted to start talking about this strategy today. There is also a strategy which is often called greedy strategy which is also going to be discussed on this course. Because, this is also gives us very fast algorithms for solving combinatorial optimization problems.

Let me, point out right away that this strategy the backtrack search strategy is inner most was general. This can be apply to almost any problem. This is the more restricted strategy, but it becomes little bit more efficient than backtrack search problem search strategies. Dynamic programming typically is more efficient than branch and bound. But,

it requires as to do a lot more analytical work. We need to actually exploit the problem and it is not always the case the dynamic programming ideas will be applicable.

(Refer Slide Time: 21:02)



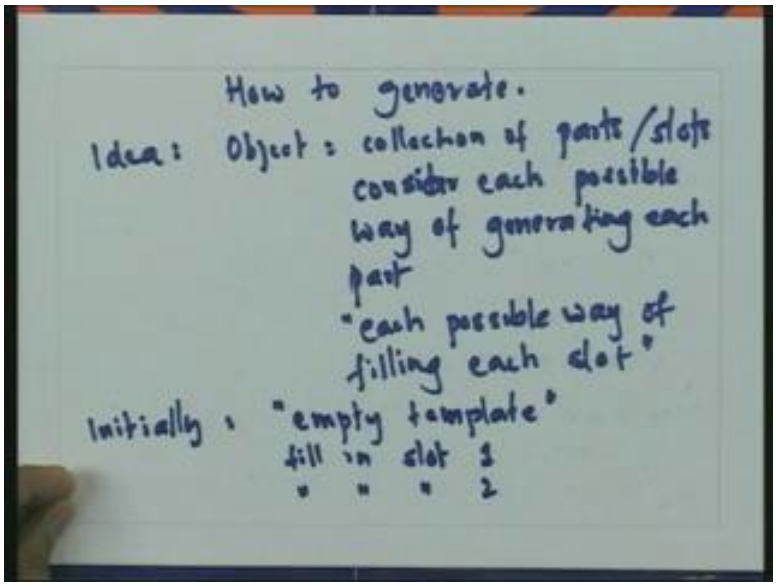
So, let us come to Brute force search strategies. The name should suggest to you that we are going to do something fairly simple minded. And in fact, let us just catch how you may do this simple minded; we might do something simple minded to make sure. However that we do get an answer correct we correctly.

So, the basic idea as you might guess or as in such as is to systematically somehow a numerate or generate all possible objects which can potentially satisfy the constraints. For each object here is, what we do? Then, we first check whether in fact the constraints are satisfied. If so, we evaluate the cost or benefit function. If the cost is better than the cost so for that is if the cost of this object is less than the cost of the best object so for then we storage.

So, we record cost if necessary. Similarly, if there is a benefit function given to us instead of the cost function. We evaluate the benefit function and check this benefit function with the best benefit function found so for. If this new object has a benefit function which is better than the best of best benefit function founds so for. Then, we record this benefit function and also the current object record cost, if necessary and current object.

So clearly, if we generate all possible objects and we check whether each of them satisfies the conditions this procedure must certainly work. This procedure must will be guaranteed to give us the correct answer. Just because, we are just going over we are doing everything that can possibly have to be done.

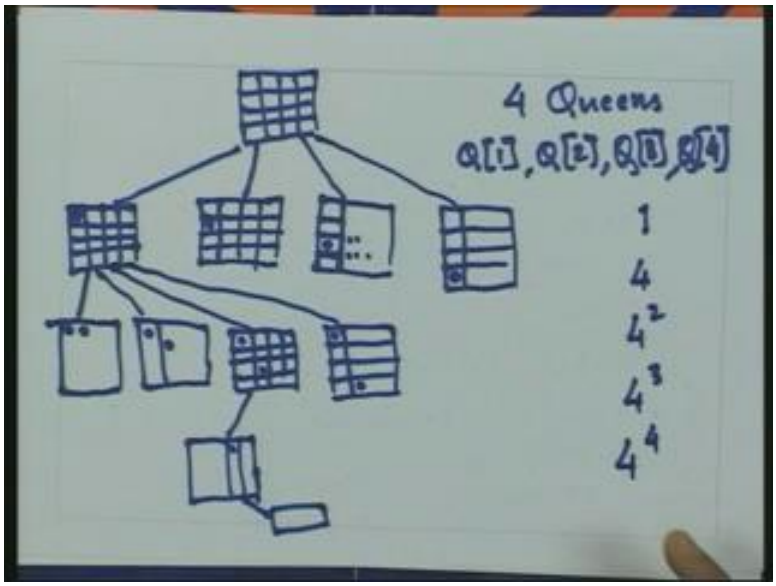
(Refer Slide Time: 24:04)



A question over is how do we systematically generate all possible objects. And that is where slight amount of cleverness comes in. So, generating all possible objects are how to generate, the basic idea is think of the object that we want as a collection of parts or a collection of slots. Then, consider each possible way of generating each part or if you want to think of this object as a template in which there are slots to be filled.

We want to think of each possible way possible way of filling each slot, how do we go about actually doing this? Well we go over the parts or the slots one after another. So, the basic idea is that initially the sort of have an empty template. So, all's slots are empty. Then, we fill in the first slot of course; this can be done in several possible ways. So here, we have to consider all the alternatives. Then, we fill in the next slot and again we consider all the alternatives over here and so on.

(Refer Slide Time: 26:17)



Let us, take an example and that will make this idea very clear. So, let us take the example of 8 queens, but sense this page is small, let us just make it 4 queens. So, what we have well we have very wells Q of 1 Q of 2 Q of 3 and Q of 4. So, we can think of these four as the smaller parts comprising our big object that we need to find or we can think of this as a template with four slots and it which is given to us and we want to fill in values into the slots.

Instead of looking at these numbers themselves. I am going to think of this I am going to interpret these numbers and pictorially draw the board that corresponds to this. So initially, we are going to start up with an empty template. So that corresponds to a board which is completely empty. Then, we are going to consider the ways in which the first slot in this template can be filled. So Q 1, we said must have a value between 1 and n where n is 4 over here.

And so, there are four ways in which this can be done. So for example, something like this. So, we can placed the first queen in this first position over here. But, this is not the only way in which this slot can be filled. There are many other ways in which the slot can be filled as well. And so, here is another. So from this empty template, we can come to this way this possible way of filling the first slot. That is that we placed the first queen in the second row.

Of course, it is not necessary that we placed in the second row, but we might place it in the third row as well. So, this is that corresponding position and of course, we can place it in the fourth row. I am not drawing the entire board. But, I am just telling you where the queen is going to be placed. So, these are all possible ways in which the first queen is placed first queen can be placed.

And we started out with the empty board and at the end of one way of one considering how the first slot can be filled. We have arrived at four possible positions four possible candidate objects each of edges only partially built. So, we started up with one partially built object candidate object and now we have four partially built candidate objects. But, they have been they have been extended.

So, we started up this object and these are object which are full arrange substance or more build up than this previous object. But, we can keep going in this manner. So from this point, we will again consider all possible ways of placing the second queen. The first queen is placed over here, the second queen could also be placed in the same row or it could be placed in the next row or the row after that or finally, in the last row.

So, when we come down to this level. We have our object even more fully constructed. So, we have placed 2 queens, we are not placed all of them here. But, this is at a higher level of construction higher level completion than this one which internal at higher level of completion and this one. And of course, each of these objects will have to be constructed further.

So, there will be something over here and meet everything which represents the ways of placing the third queen. And then and read, this there will be something which represents the ways of placing the fourth queen. In this case, we can see what is going to happen. So, we started with one node, then we have four nodes we have four bolts in this next level.

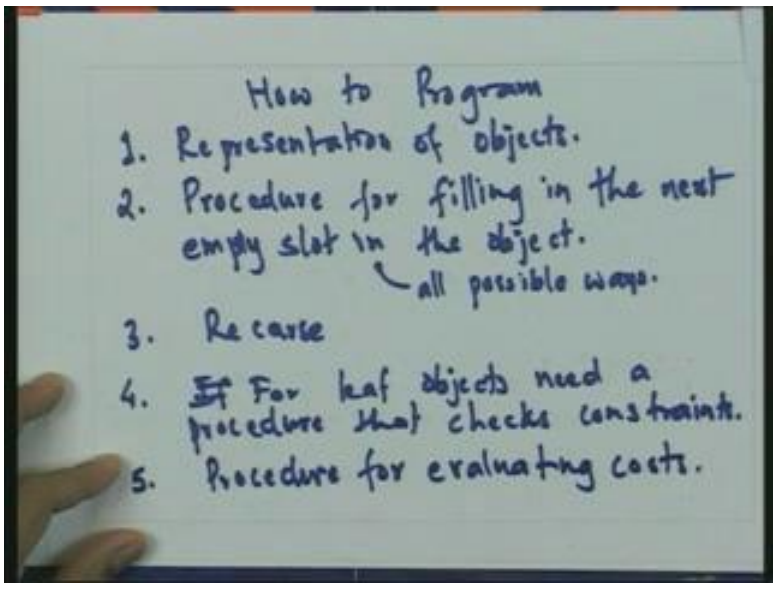
Then, we have a 4 square bolts to be consider at this level 4 square partially constructed objects. Then, we have a 4 cube partially constructed objects at the next level 4 times as many as the when the previous level. And finally, 4 to the 4 objects at the leaf level, in fact, I have already told you what this looks like in some sense. So, we have leaves at the bottom, we have root at the top and in fact, this is going to look like a tree.

In fact, in almost every in every exhaustive generation method we will be following something like this. So, we start of in the empty object, we keep on extending it. So, the extension will correspond to children of the parent node. Their extensions will correspond to object which are even fuller. And until we get to a point at which we cannot extend object any further. That is the point at which we have got to leaves and that is the point at which we can terminate our generation step.

Once, we get to a leaf in this case for example, we know that we have all the queens completely placed. So, what is the next step? Well at this point we are going to check if the queen position satisfy our conditions. If they do then in this case, we can just report we can just print out saying yes they do in fact satisfy all the conditions. If they do not well, then we have to consider the other leaves.

This is as for as considering problems in which there is no objective function. If there is an objective function then when we get down to this level, we need to not only consider whether the leaf the leaf objects satisfy the constraints. But, we also need to evaluate the cost function. So suppose in fact, we have a cost function rather than a benefit function. This for each leaves, we can think of the cost function. And we can evaluate the cost function and our problem is to find a leaf such that it is satisfies all the constraints and such that whose cost functions is as small as possible.

(Refer Slide Time: 33:07)



So, let me, said as summarize what I said in terms of a programming idea. So, first we need something so that we represent our objects properly. So for example, in the case of the 8 queens problem, where the four queens problem? We used a data structure of an array of four element array to represent the four queen positions.

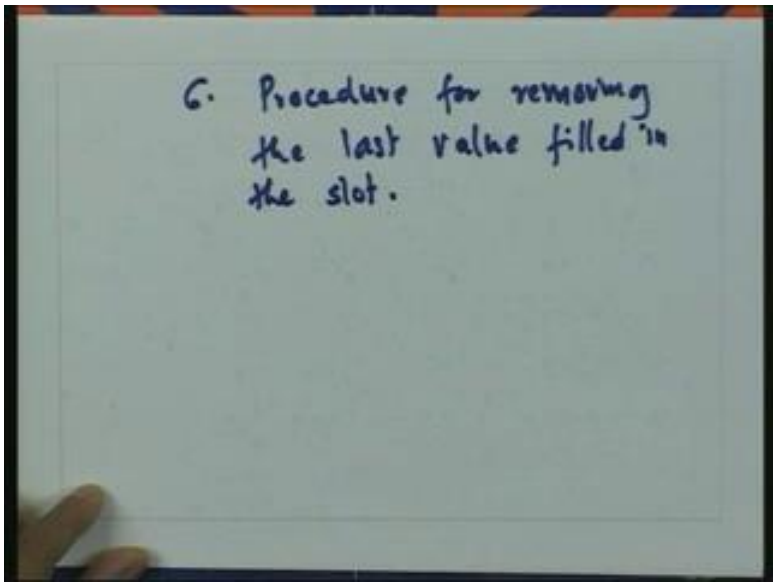
In case of the travelling salesman problem of the travelling salesperson problem, we want to represent the tour again if the graph as n vertices we could use an array of length n, where the ith element of the array denotes the ith city in the tour. In the case of the knapsack problem is simply need to say whether a certain object has been selected or not. So, corresponding to each object we could have a bit which is equal to 0 meaning that object is not replaced in the knapsack or if that bit is 1, then the object is replaced in the knapsack.

So some of the other, we need to have a representation of the object that we are looking for. After that, we need a procedure for filling in the next empty slot in the object. Of course, there will be many ways in which this can be done. And so, this procedure should allow us to provide all possible ways. At this point in the algorithm, we are simply going to recurse. So then, we will recurse and we will recursively fill in the sub sequential loss, what happens next? When we have recurse we will it is prevalent to starting at say the root.

And we filled in the object filled in the first slot and then when we recurse we filled in the next slot and so on. Eventually, we will get the leaves. So viral you need to have a check over here, just to see whether we have in the leaves. So for leaf objects, we need a procedure that checks constraints. I am it written true if all constraints are met it written false if the constraints are not met.

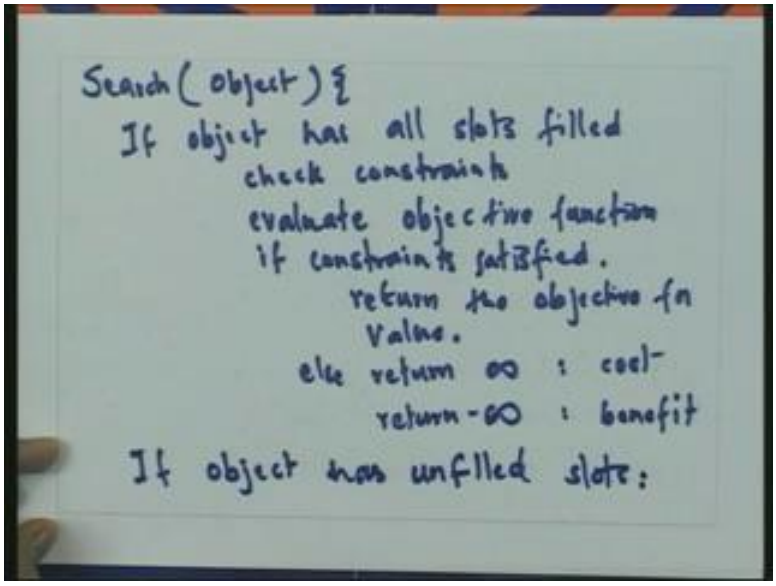
We also need a procedure for evaluating cost. That is not all however. So this way, we keep going down and we go down to a leaf. But, you also want to go back up, and go to the other leafs and that is why this whole procedure is called back track search. So, we get to the leaf and then we go back up again in the tree. And then, we go to the next leafs then from that leaf, we go back up again.

(Refer Slide Time: 37:10)



So over here, we said that we need a procedure for filling in the next empty slot in the object. But similarly, we need a procedure for removing the last value filled in the last slot. This is how we will go back up into the tree and make sure that all the objects will get generated.

(Refer Slide Time: 37:50)



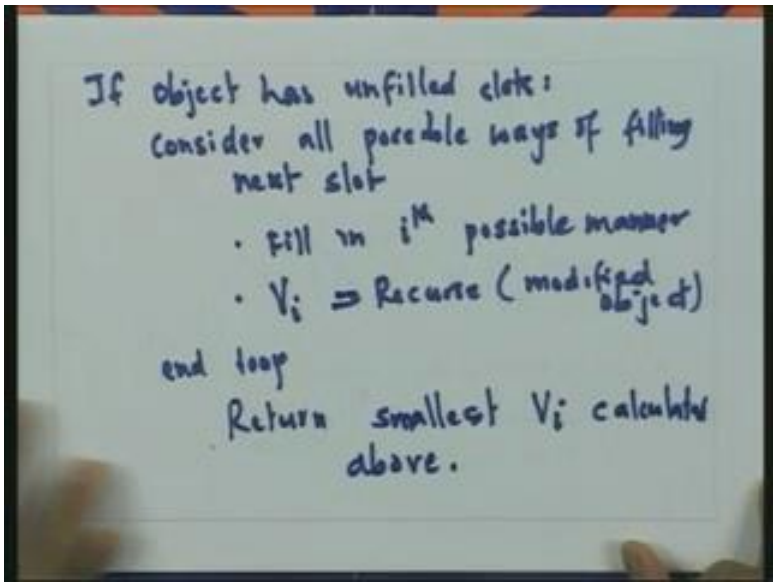
So, let me try to write this as a very vaguely as a procedure just give you the idea of what is going on. So, our search procedure is going to look something like this so it going to take this candidate objects as an argument. And let me, just note that this while have a

return while have return on object every here I really mean this template which could be partially filled.

So, what does this do? So, if object has all slots filled which is to say that it is a leaf. Then, we are going to check constraints. We are going to evaluate the objective function. If constraints satisfied and in fact we will return the value, if the constraints are not satisfied we will return. So, if the constraints are not satisfied, then will return infinity if we are given a cost function. That is to say that this leaf found is to be disorder. Infinitive will just say that this is a useless leaf.

If you return a minus infinitive or a zero if this is a benefit function. Assuming that the benefit is positive we can even return zero. So this is what we do at a leaf? If object has unfiled slots then what we need to do? Well we need to do this branching business.

(Refer Slide Time: 40:09)

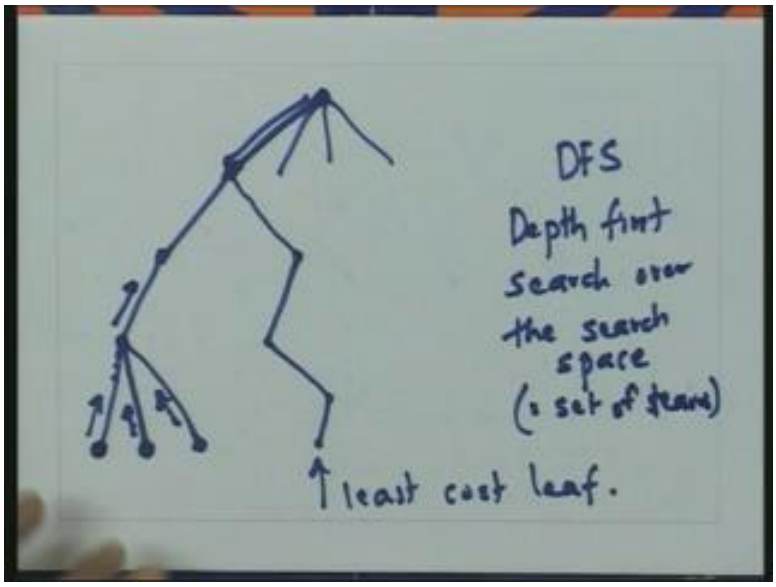


So, this has an unfilled slot. So, we need to filled for this slots, if object has unfilled slots. We are going to consider all possible ways of filling slots filling next slot. So, you will fill in ith possible manner. We will recurse on modified object. And this recursion will give as value. So, let me call this V sub i and this will be in the end of the loop. So, this think of this consider as being a loop.

And now the idea is something like this. So, we are going to be go over all possible leaves underneath. So, starting at this or starting at this we will be visiting this will be

visiting this will be visiting this will be visiting this will be visiting this. And we are going to associate with this node a cost function which is the value of the smallest cost function found anywhere over here. That way, we can written that and eventually will make sure that the smallest cost gets written to the top.

(Refer Slide Time: 42:13)



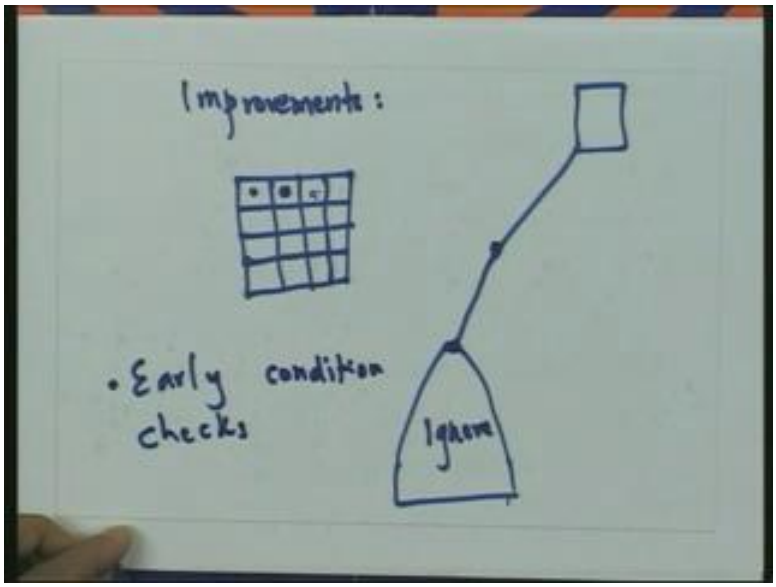
So, here we are going to be return in smallest V sub i calculated above. So let me, do this again. So, we start up with an empty object, then we consider all possible ways of doing it. But really in the first instance, we just consider we just go along this path. Then, we go along this path then we go along this path and so on until we get to a leaf. Say let us say we get to a leaf over here.

Then, we return the value of the cost as seen at this leaf. Then, at this point we consider other waste of generating it and we get to another leaf for example, then we return this cost. Then, we go back again then we return the cost over here. And let us say there are no more ways of filing in that last slot. Then at that point what gets return over here is the smallest of these cost values.

So, in general in this manner, what is going to get return to the root, is going to be the value generated at along some path in this graph. And that is going to be the least cost leaf the value of the least cost leaf. So, this complete at description of this search procedure. In fact, if you have studied this in the data structures scores, what you have done really is a depth first search over this search space. Search space is simply set of

leafs. We are looking at all the leaves somehow the other. And so that is of called search space. And what we are doing is? We are putting a tree on top of that search space and we are just doing a depth first search of it.

(Refer Slide Time: 44:04)

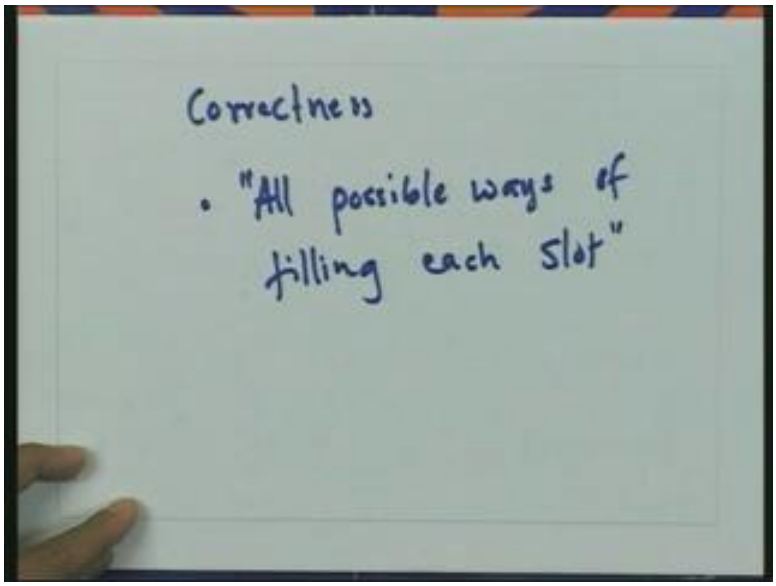


There are some improvements possible to procedure. We do not need to do our condition with checks our constraints checks at the very end. We can do them as early as possible. So for example, in case of the 4 queens problem. Suppose, we placed the first queen in this first column in this first row then when we placed the second queen in the first row as well. We could say look these 2 queens already capture in each other. Or the condition that Q of 1 equals to Q 2 to not equal to Q 2 to is not being made.

And therefore, we can say we do not really need to go and further check where condition where queens three and four placed. So, what happen to be here was that we started up with empty board. Then we placed the first wheel and then we placed the second wheel as well. And then, we can safely ignore this part of the search tree immediately.

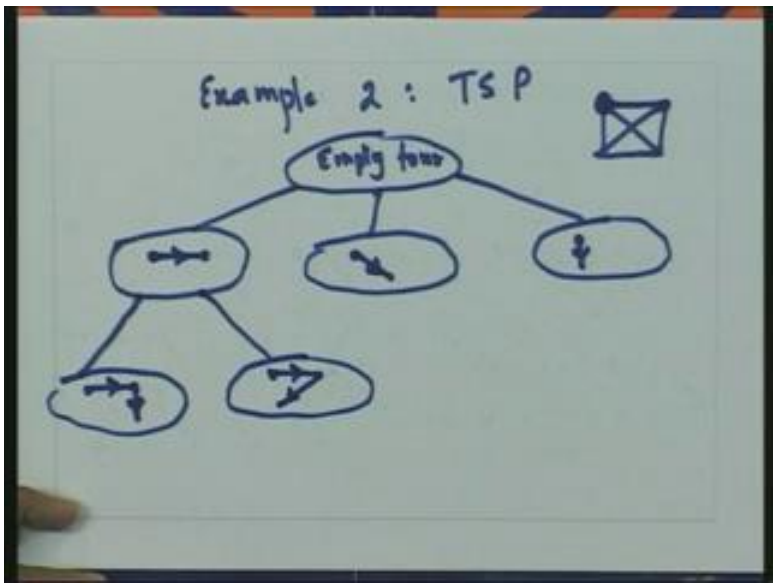
Because, we have checking the condition right here and this condition which we check should be something which will be valid no matter what will be get over here. And in fact, that is true, these 2 queens are not going to change their position at any of these leaves at these leaves only these queens will get placed. And therefore, this condition ((Refer Time: 45:18)). So, the possibility of early conditional checks condition checks can improve up on efficiency.

(Refer Slide Time: 45:36)



I would also like to note that this procedure is; obviously, correct. And that really follows from the fact that we are considering all possible ways of filling each slot.

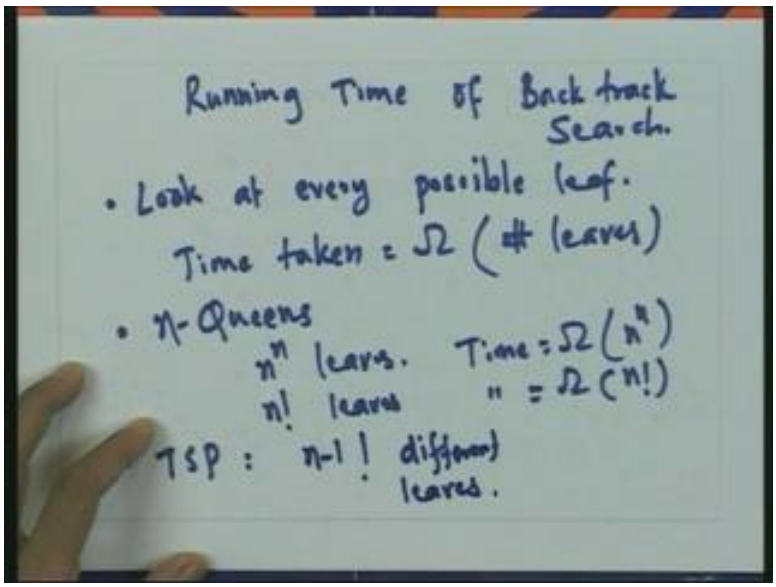
(Refer Slide Time: 45:56)



Let we go to our second example which is the travelling salesperson. So, I am just going to give a quick description of what this search tree is going to look like. So at the root, we are going to have the empty tour. So, let us see this is our graph. Then, we can say that let us say just designates this vertex of the starting vertex it does not really matters. Since, we are going to come back to vertex anyway. Since, the tour is cyclic.

But suppose, this is the starting vertex, then we can ask how many ways are there in which the tour we can go out of this vertex. And that gives us that is sort of closing in the first slot in the tour. So for example, we can come to this point at which the tour has proceeded along this edge or we can come to this point at which the tour has proceeded down diagonally or we can come to this point at which the tour has gone now. Then this tour can further we extended over here. So starting at this, we can go down directly or it can go back diagonally and so on. So, eventually we will get to a leaf where will be having complete tour.

(Refer Slide Time: 47:29)



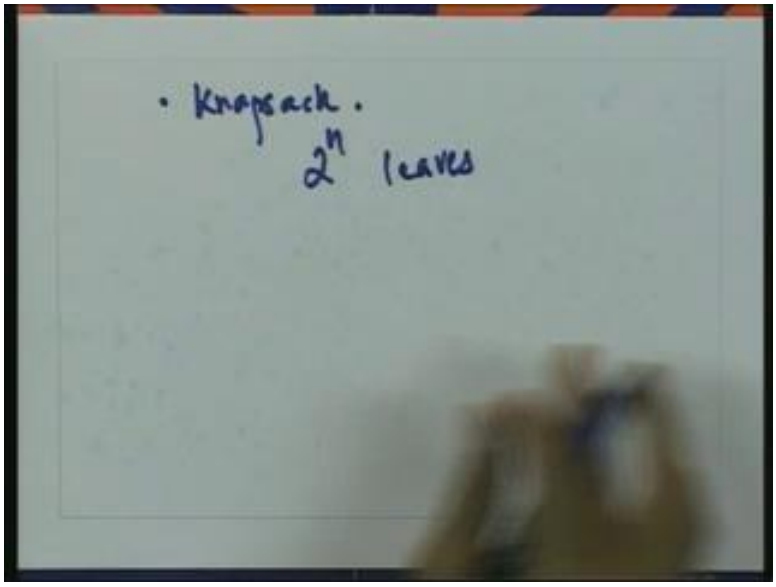
So, let me say something about the running time of backtrack search. So basically, the idea is that we are going to look at every possible leaf. So, the time taken will be will essentially have to be at least in the worst case have to be the number of leaves. So, let us consider this for the case of the n queens problem. Specialized generalized it 2 n queens rather than 8 queens or 4 queens.

So in this case, how many leaves to be have? Well the first queen can be placed in n ways the second can be placed in n ways and so on. And so, there are n to the power n leaves and so time is at least n to the power n. If you implement the early checks, then the early checks will improve the situation some part. And in fact, a simple very simple early checks is to say let me make sure that this new queen is placed in a different row from all the previous queens.

So here, our output will simply be a permutation on the numbers 1 through n. And so, here there will be n factorial leaves. So, the time is going to be omega n factorial. This is still to still quite large, what about the TSP, how many tours do you have, through n cities. Well starting at any city the first city you can be placed can be picked in n minus 1 factorial ways. Then, n minus 1 ways then the next n minus 2 ways and so on and therefore, we will have n minus 1 factorial different leaves in this search space.

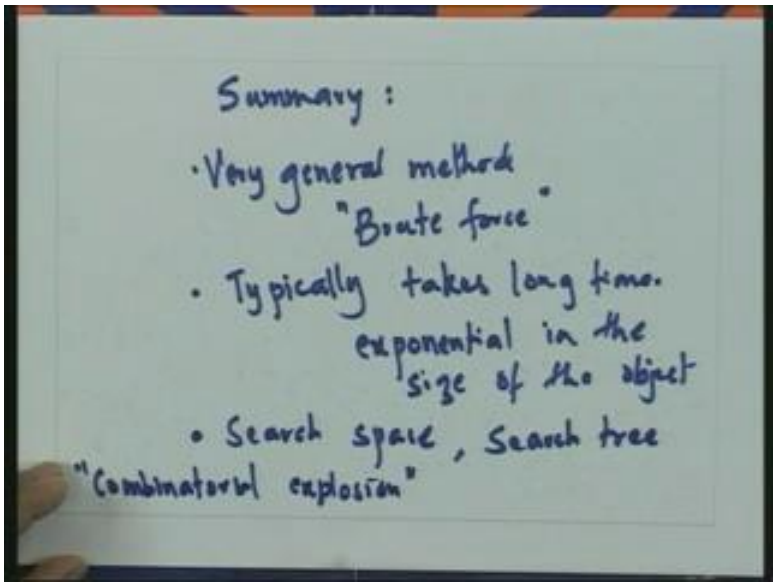
So simply put, there are three ways of getting to the first the vertex of the first level. The next there are two children and in this case there will be 1 child and so on. But in general, if you start you can n city tour there will be n minus 1 children of the first level n minus 2 children at the second level for each of the vertices in the previous level. So therefore, we will have n minus 1 differently used and again here the time will be omega n factorial.

(Refer Slide Time: 50:10)



For the knapsack problem, it should be possible for you to improve that there will be 2 to the n leaves corresponding to all possible ways for mean subsets of n objects.

(Refer Slide Time: 52:00)

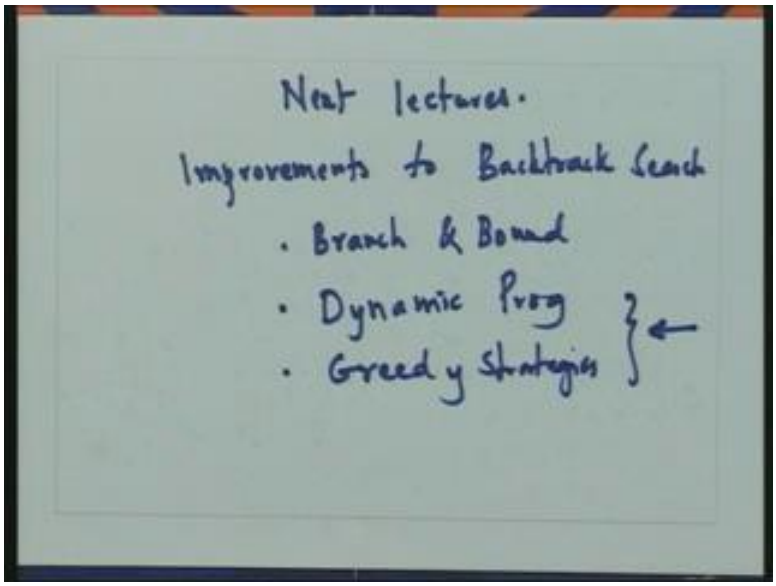


So, let me summarize this co situation. So, what have you done? So for, we have looked at very general method. It is Brute force in that it generates every possible object and it does not try to be two efficient or two clever in saying look I do not. I am not going to generate this class of objects, because this is not going to be useful to me.

And typically it takes long time. In fact, typically takes time exp1ntial in the size of the object. So either, due to again or n factorial or n to the n or something like that and we have defined terms likes search space and tree that set on it as a search tree. And let me end this lecture by mentioning one more term which is combinatorial optimization combinatorial explosion. This term is commonly used in connection the back track search.

The idea is that at the first level the tree might look manage level, because it has a small number of children. And the second level the number of children the grand children of the root will grow. At the third level the number of children will be a further. And eventually even you get believes you will have many, many, many children or many many, many vertices. And so, enzymes your tree will explore and this think and this phenomena is commonly called combinatorial explosion.

(Refer Slide Time: 52:40)



So let me, just look at what we are going to do next. We are going to study improvements to backtrack search. As mention before we are going to study a technique called branch and bound. And then we will look at dynamic programming and greedy strategies. And this is really the focus of this course, but it is important to view these as part of a grand picture in which sort of the simplest backtrack search ideas are present as well as these more sophisticated strategies are present.

We should also note that the backtrack search is a very, very general strategy, whereas these are very, very sophisticated. But very specialized strategies, these are likely to be fast as we will see in the next lectures. This is likely to be slow, but it is likely to ((Refer Time: 53:52)) It is going to very, very general and applicable in almost any combinatorial optimization problem.