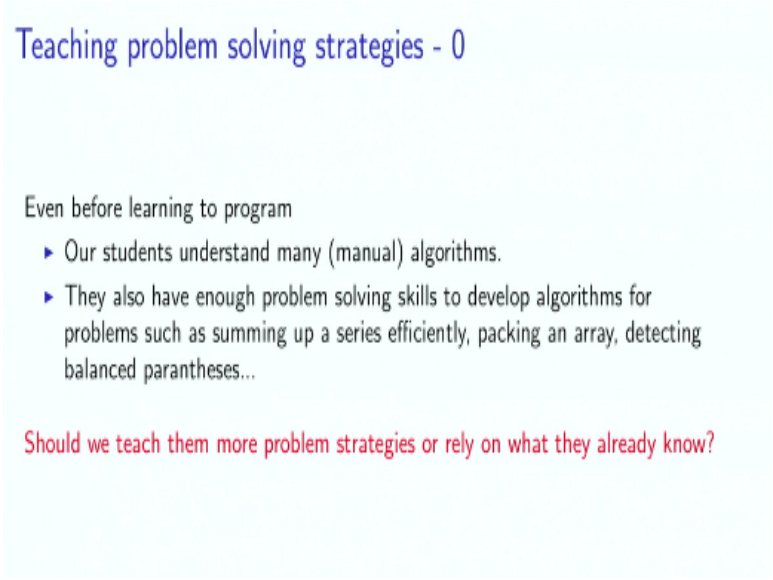
Design and Pedagogy of the Introductory Programming Course
Prof. Abhiram. G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology – Bombay

Lecture - 08
Basic Ideas in our Approach 3: Should We Teach Students (Manual) Program Solving Strategies

Hello and welcome back. The topic for this session is should we teach students problem solving strategies and in particular manual problem solving strategies.

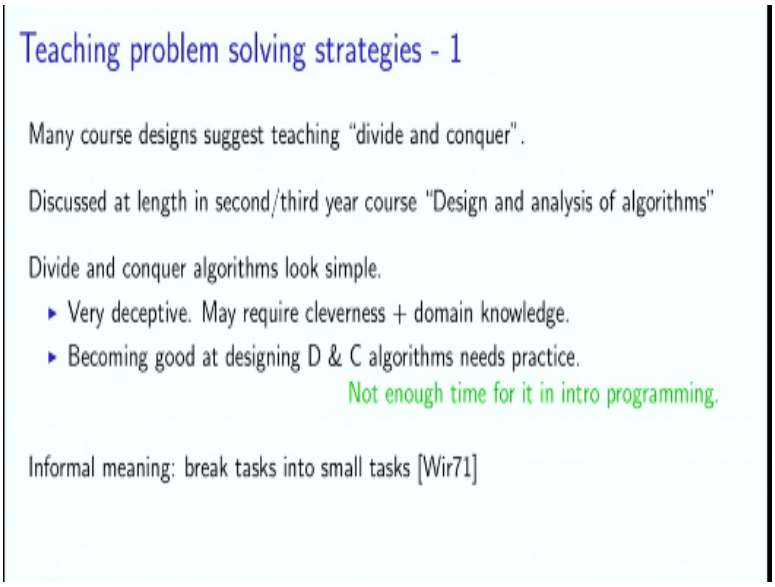
(Refer Slide Time: 00:34)



So let me put out the context. Even before learning to program our students understand many manual algorithms. These are algorithms that they have learned in primary and secondary schools and also in junior college. Now they also have enough problem solving skills to develop algorithms for problems such as summing up a series efficiently, packing an array, detecting balance parentheses this is what we just saw in the last lecture. Should we teach them more problem solving strategies or should we rely on what they already know.

That is the reason that we want to discuss in this lecture.

(Refer Slide Time: 01:18)



So many course designs suggest teaching a strategy which is loosely called divide and conquer. This strategy is often discussed at length in the second year or third year course on design and analysis of algorithms. This course is considered a difficult course actually and this strategy while it looks simple it is actually quite difficult. It requires some amount of cleverness, some amount of domain knowledge and it is a very powerful strategy, but to make it work it is not easy you need lot of practice.

So if you want to become good at the strategy in its technical sense. So be able to prove correctness and all of that requires a lot of practice and of course it has to be taught well and it also in some sense requires some maturity as well. In introductory programming course my feeling is that teaching is very powerful, but very precise divide and conquer strategy. The name (()) (02:38) very, very specific way of thinking that is probably not a good idea.

Probably there is not enough time for it in the introductory programming course. Now this term is also used in an informal sense and in the informal sense in day-to-to-day (()) (02:59) it means somehow or the other break tasks into small tasks. This is what Niklaus Wirth says in his 1971 paper and in that sense breaking up something into small pieces itself is a good idea.

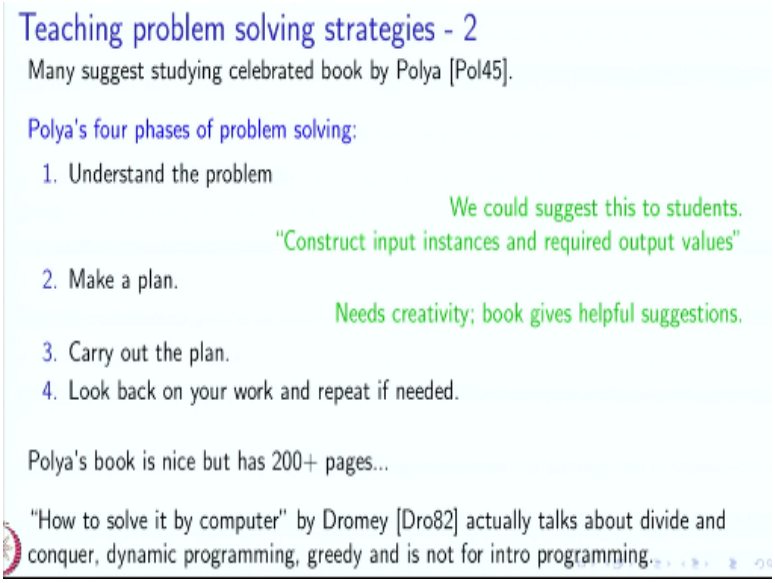
I mean if you want to do something large it is always a good idea to break it into small pieces. Of course just because it is obvious it does not mean that you should be telling people about it. You should tell students however there is a difficulty. How do you break it now that is not always all these? You may say you break it into pieces, but how do you break it that is

not at all clear.

Because breaking it up in one way does not cleanly separate things into sub problem. As an example consider the problem of placing 8 Queens. You cannot say place the first Queen, place the Second queen, place the Third queen because these things do not happen in isolation. The way you place the first Queen or the first few Queens affects what happens to the last Queens whether you can even place them at all.

So breaking up into small tasks is good advice, but it is not the end of it. It requires lot more insight probably to really make it work in general, but sometimes it can be adequate.

(Refer Slide Time: 04:41)



Now in a lot of the literature on programming there is a mention of really beautiful book by Polya it was written as early as 1945. So the title of the book is how to solve it. So it is a book about problem solving. I should say Mathematical problem solving. So the examples are substantially from Mathematical problem solving though there are other examples like solving crossword puzzles.

But Polya is a Mathematician and the power or the punch of the book is really in the mathematical problem that Polya deals with. Polya recommends a four phase problem solving strategy. First phase is understanding the problem. Now understanding the problem on one hand seems like a very obvious advice. You might say look I read it what more can I do, is there anything more in terms of understanding it.

Would not you understand it the moment you read it. Well, we should suggest this to students, but we should also suggest to them that not only should you read it and try to understand it whatever that means to you to check whether you have actually understood, construct input instances and the required output values. If you can construct input instances what kind of input does your program need to take and what output should it produce for that input.

Then we could say or you could say to yourself that okay now I understand the problem just by reading you do not really understand the problem and of course there are interesting input for instances and not so interesting input instances. So the additional advice could be construct input instances or construct tricky input instances. The next step in Polya's problem solving strategy is to make a plan.

Now this is a pretty difficult or let say this is a pretty mysterious phase. So Polya gives several helpful suggestions. So for example he says that think about whether you have seen problems like this before or think about what really is the problem is there an abstract version effect. So the problem maybe in terms of money, it maybe in terms of time, it maybe in terms of any other energy, but what are the relationships that are required to be adhere to.

And is there an objective to be optimized. So Polya says think about it and that will enable you to maybe make a plan. So by a plan Polya means some way of solving it or some possible way of solving it. It is a plan in the sense that the plan may fail, but unless you make a plan and this is an important point, unless you have a plan in mind you are not going to succeed. Even if you have a plan in mind you may not succeed.

But if you do not have a plan in mind the chances of succeeding are very slim that is what Polya says then you carry out the plan. Your plan may succeed or it may fail and in that case if it fails you look back on your work and say okay why did it fail can I try a different plan. So all of this seems fairly simple at least at this level, but Polya gives lots of good suggestions and his book is beautiful written I must say, but it has 200+ pages.

And it tackles difficult Mathematical problems as well. So it is not light reading. It is a book which you have to take up, you have to solve problems, you have to be patient. It is a subject in itself. So while you could say that problem solving is something that Polya has written up and has beautifully written up and you can learn from it. It is hard to think of it as a

supplementary reader for an introductory programming course.

There is not just enough time and let us face it I think it requires a certain kind of expertise in terms of the domain of the problem, problems being solved and that conveying all of that and conveying all the information about the programming languages and things like that which we have to do is going to be very difficult. Now there is another book called how to solve it by computer which at first glance might seem more promising.

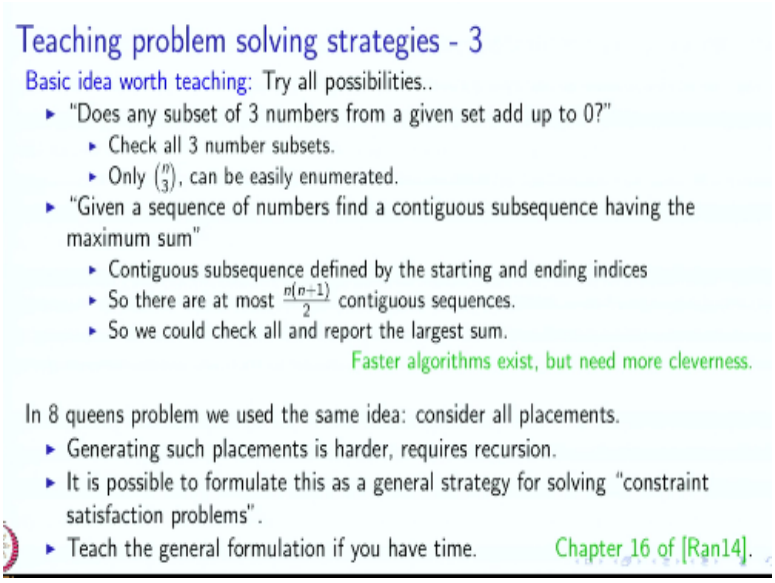
It is written by Dromey it is a slightly more recent book. It actually talks about divide and conquer, dynamic programming, greedy strategies and so on, but these are strategies which are taught in the design and analysis of algorithm course and again these are not strategies meant for light reading. They are strategies which you learn by example surely or by explicit instructions, but a key point is that you have to think about the problems that you solve.

You have to understand the strategies, the proof techniques which go along with them and again it is a lot of time and remember that this is considered to be a difficult design and analysis of algorithm is considered to be a difficult course. So sort of standard problem solving advice might probably not work in this course. Now we could say let us try shall we at least tell people about some commonsense things.

So that is probably a more workable idea. So what we should be telling students is that okay you have been doing problem solving so we expect you to continue along with your street smartness or whatever you have learned so far, but in addition to that remember that there are somewhat common sense ideas and you may forget them or they may not occur to you. So let me just mention to you.

So one such idea is try all possibilities. So let me give an examples. So does any subset of 3 numbers from a given set add up to 0. So what is the input? The input consists of some n numbers say in an array.

(Refer Slide Time: 12:00)



And I want to know whether any subset of 3 number from it adds up to 0. How do you do this? So one simple idea is to ask can I do it by Brute force. So well I will look for all 3 numbers subsets. So how many 3 numbers subsets are there of n numbers. Well that is simply n (()) (12:23) or n *n-1 * n-2 upon 6. So can we now generate all of those 3 numbers subsets. Yes, we can we can just write 3 nested loops and we should be able to generate all those 3 numbers subsets.

So we try out all 3 numbers subsets and choose 3 of them we enumerate them in the sense that we generate them systematically one after another and that is it. We print out subject if we find that subset adds up to 0. Here is another problem given a sequence of numbers find a contiguous subsequence having the maximum sum. So this might look like a more difficult problem.

But again if you ask the question how many possibilities are there the answer is actually fair (()) (13:15). So contiguous subsequence is defined by the starting and ending indices. So for example if I have a sequence of n numbers where the indices go from 0 to n-1 what are the subsequences that I can have. Well so the indices allowed are 0 to n-1. So I can say that the first index could be 0 and for that the next index the last index could also be 0.

So this might just be a sequence of length then maybe 1 then maybe I have index 0 and then 1, 0 and 2 and 0 all the way till n-1. Similarly, I can ask how many subsequences which are the subsequences which start at 1. So I have 1 going to 1, 1 going to 2 all the way till 1 going to n-1 and so on. So if you add up this then you will get something like n* n+1 upon 2

contiguous subsequences.

And these subsequences can also be generated fairly easily. So we could check all of these subsequences, add them up and report which one of them had the largest sum. So this is not a difficult piece of quote to write as I said it is going to consists of sum number of nested loops that is all. Of course this is not the best way of doing things, but the better way of doing things need more cleverness and we will come to this a little bit.

But at the level of the introductory programming course, the fastest possible algorithm which will run in linear time is probably too much to expect, but we will come to that nevertheless so that at some point during this course you will see that algorithm as well. And I will suggest later on a way in which you could pose that problem even to first year students by giving perhaps some hints.

Now we studied the 8 Queens problem and as you might recollect we used the same idea over there as well which said consider all placements of the 8 Queens. So generating such placements however is a little bit harder because we have 8 Queens. So one way to do it could be to write 8 nested loops. So (()) (15:55) means, but suppose somebody says so tell me what happens on (()) (16:02) where n is going to be given to you as an input.

Now in that case you are not going to be able to write n nested loops because you do not know n beforehand. So there you will need to do recursion. So it is a little bit of a harder proposition, but it can be made to work it is not terribly hard and it is possible to formulate this as a general strategy for solving what are so called constraint satisfaction problems. So in the 8 Queens problem there are some placement needed and the placements have some constraints that must be met.

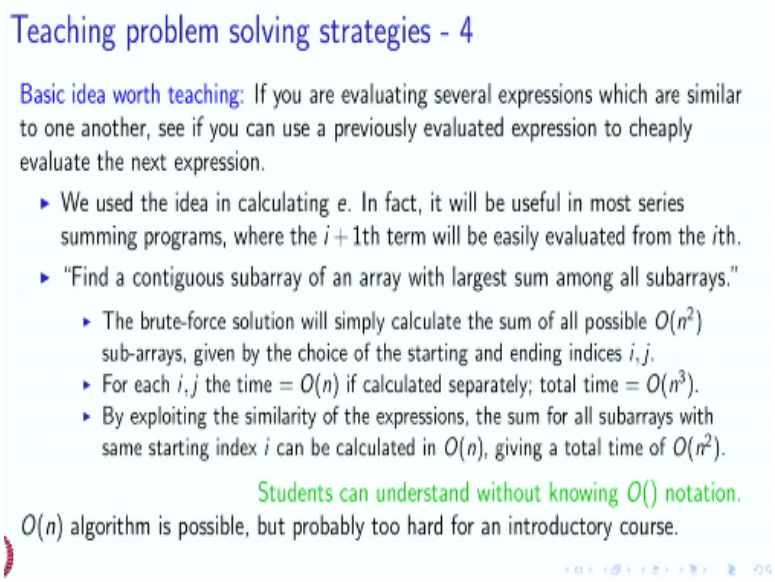
So you can actually generalize this and you can write a general purpose strategy for solving constraint satisfaction problems and it can be made to work for the 8 Queens problem as a special case. So if you have time, you can teach this general formulation as well. Although I would say that only if you have time it is not I would not rate it as a core part of the course. So later on we will see that we are going to have a notion of (()) (17:16) core or advance topic that you could potentially study.

And this last bit might be something like an advanced topic, but the first part just the idea of trying all possibilities and doing it for simple problems. I think you should mention, you need not mention it in a specific lecture, but probably the best way of doing this is that give those problems and that indicates that look why do not you think about trying all possibilities. So it is one sense obvious things, it is on another sense a powerful thing.

So rather than take up a lecture in which some students might wonder well is not this obvious. It is I think a better idea to perhaps teach it along with another problem that you are solving in class or maybe a problem that you have given as homework or something like that. And by the way this 8 Queens strategy or the constraints satisfaction general strategy has been discussed in our book in chapter 16.

So should you want to teach it you will have reading material to assign to your students. Now here is another idea which is again a sort of a commonsense idea, but may not occur to students.

(Refer Slide Time: 18:38)



And this is as follows. If you are evaluating several expressions and by this, I mean mathematical expressions and if these expressions are similar to one another see if you can use a previously evaluated expression to cheaply evaluate the next expression. So you have evaluated something already check whether the next thing that you want to evaluate is only slightly different.

And therefore you can evaluate that without doing it from scratch. You can just maybe (())

(19:10) modification maybe make an extra addition or an extra division or something like that and you get your new expression as well. So this is way to actually reduce your work. So we have already use this idea as perhaps you might guess already. We use the idea in calculating E and in fact it will be useful in most series summing programs where the i+1 term will be easily evaluated by the ith term.

So i+1th term should not be evaluated from scratch, but you could ask look if I already know the ith term how different what more do I need to get the i+1 term. We just saw this problem find a contiguous subarray of an array with largest sum amongst all subarrays. Now the Brute force solution will simply calculate the sum of all possible O of n square subarrays or say roughly n square subarrays given the choice of the starting and ending indices.

However, for each i, j the time is O of n. So if you calculate it separately the total time is O of n cube. By exploiting the similarity of the expression the sum of all subarrays with the same starting index can be evaluated in O of n time total not just for single subarrays, but all subarrays starting with common index. So for a single starting index you will take O of n time so for all starting indices you will take n time as much or overall you will take time O of n square.

So already you have a reduction from n cube to n square. Now I have written the O notation over here. You do not have to tell the students the O notations certainly not formerly. However, you can say and you should say in class that in time proportionate to n, time proportionate to n square. So you are indirectly hinting that look we are worried about whether the time is proportionate to n to n square or things like that.

I think I mentioned earlier, but let me say that again there is a very nice O of n time algorithm possible for this problem and it is a bit too hard for the introductory course, but I will discuss it because I think if you can do it. It is a good idea to assign it to the class with some additional hints. So they will get a sense of discovering some nice algorithm. So I am going to stop here and in the next lecture I am going to actually do the design of the course that we have set out to do. Thank you.