Problem Solving in Computer Science

Tim Capes

March 2, 2011

A3 announcement

I have decided to reduce A3 by one question and hand it out on Tuesday instead of Today. A lot of you have a large amount of midterms this week and having 2 different assignments to work on along with a midterm from this course is a little excessive.

Different Type of Topic

In this course so far you've seen a lot of programming. Programs are written in order to execute specific tasks which are often related to solving a problem. In this part of the course we are going to take a high level look at some of the theory of Computer Science and then get into some problem solving skills for computer science.

A Famous problem, P vs NP: Background

1. Computer Scientists classify problems by how hard they are

A Famous problem, P vs NP: Background

1. Computer Scientists classify problems by how hard they are
2. One measure of how hard a problem is, is based on the behavior of the worst case runtime as a function of the input length.

A Famous problem, P vs NP: Background

1. Computer Scientists classify problems by how hard they are
2. One measure of how hard a problem is, is based on the behavior of the worst case runtime as a function of the input length.
3. For example, if for every length x the worst input of size x takes time x then this problem has linear runtime.

A Famous problem, P vs NP: Background

1. Computer Scientists classify problems by how hard they are
2. One measure of how hard a problem is, is based on the behavior of the worst case runtime as a function of the input length.
3. For example, if for every length x the worst input of size x takes time x then this problem has linear runtime.
4. Every program you've written so far for assignments has worst case runtime x * x where x is the size of your input. Your only expensive operation was loops and your had at most two of them nested.

P vs NP: Further Background

1. Two of the most well-known classes are P and NP

P vs NP: Further Background

1. Two of the most well-known classes are P and NP
    1.1 P stands for Polynomial, and contains of all problems we can solve in polynomial time

P vs NP: Further Background

1. Two of the most well-known classes are P and NP
    1.1 P stands for Polynomial, and contains of all problems we can solve in polynomial time
    1.2 NP stands for Non-Deterministic Polynomial, and contains all problems we can solve in polynomial time if we are allowed to guess bits correctly one at a time.

P vs NP: Further Background

1. Two of the most well-known classes are P and NP
   1.1 P stands for Polynomial, and contains of all problems we can solve in polynomial time
   1.2 NP stands for Non-Deterministic Polynomial, and contains all problems we can solve in polynomial time if we are allowed to guess bits correctly one at a time.
2. The question is in some sense is it possible to write code efficient enough to guess well?

P vs NP : An open problem

1. The answer is that we don't know, this problem has been open (not shown to be either true or false) since the 1970's.

P vs NP : An open problem

1. The answer is that we don't know, this problem has been open (not shown to be either true or false) since the 1970's.
2. We suspect the answer is we cannot: Guessing right is very powerful.

P vs NP : An open problem

1. The answer is that we don't know, this problem has been open (not shown to be either true or false) since the 1970's.
2. We suspect the answer is we cannot: Guessing right is very powerful.
3. One can think about this in terms of a guessing game (but this analogy only goes so far)

A 3 peg guessing game

1. You have 3 pegs to guess, each of which is white or black, and you are only told if your entire guess is correct or incorrect.

A 3 peg guessing game

1. You have 3 pegs to guess, each of which is white or black, and you are only told if your entire guess is correct or incorrect.
2. In order to check your guess you have to have a guess for all 3 pegs.

A 3 peg guessing game

1. You have 3 pegs to guess, each of which is white or black, and you are only told if your entire guess is correct or incorrect.
2. In order to check your guess you have to have a guess for all 3 pegs.
3. This problem takes up to 2^3 steps (worst possible guess order) normally.

A 3 peg guessing game

1. You have 3 pegs to guess, each of which is white or black, and you are only told if your entire guess is correct or incorrect.
2. In order to check your guess you have to have a guess for all 3 pegs.
3. This problem takes up to 2^3 steps (worst possible guess order) normally.
4. This problem takes 3 steps if we are allowed to guess one bit at a time.

Connecting back to P vs NP

1. Consider the worst possible scenario for a guessing algorithm playing the n peg version of this game

Connecting back to P vs NP

1. Consider the worst possible scenario for a guessing algorithm playing the n peg version of this game
2. It will take 2^n steps (guessing all 2^n combinations last one correctly).

Connecting back to P vs NP

1. Consider the worst possible scenario for a guessing algorithm playing the n peg version of this game
2. It will take 2^n steps (guessing all 2^n combinations last one correctly).
3. If we are able to guess bits correctly it will take n steps.

Connecting back to P vs NP

1. Consider the worst possible scenario for a guessing algorithm playing the n peg version of this game
2. It will take 2^n steps (guessing all 2^n combinations last one correctly).
3. If we are able to guess bits correctly it will take n steps.
4. This is a massive difference linear vs exponential

So why isn't this solved?

1. The motivating example above isn't a program, it gets new information every time it makes a guess.

So why isn't this solved?

1. The motivating example above isn't a program, it gets new information every time it makes a guess.
2. If we restrict to programs it becomes much more technical, much more formal and uses much more advanced math.

What we will do in this section?

1. We've now seen a glimpse of a famous problem in computer science, and we don't even have the tools to formally define it. We obviously aren't going to attempt solving it either.

What we will do in this section?

1. We've now seen a glimpse of a famous problem in computer science, and we don't even have the tools to formally define it. We obviously aren't going to attempt solving it either.
2. We're going to do some problem solving Methods instead. We're interested in methods because computer science is fundamentally about process, and these methods tell us how problems are solved.

Polya

1. George Polya was a famous Hungarian Mathematician.

Polya

1. George Polya was a famous Hungarian Mathematician.
2. However, his most famous contribution to Mathematics was quite possibly to Mathematics Education

Polya

1. George Polya was a famous Hungarian Mathematician.
2. However, his most famous contribution to Mathematics was quite possibly to Mathematics Education
3. This contribution was a method of solving problems, which applies for more broadly than in Mathematics.

Polya's Method

Polya's Method breaks the problem into four key principles.

1. First Principle: Understand the problem

Polya's Method

Polya's Method breaks the problem into four key principles.

1. First Principle: Understand the problem
2. Second Principle: Make a Plan

Polya's Method

Polya's Method breaks the problem into four key principles.

1. First Principle: Understand the problem
2. Second Principle: Make a Plan
3. Third Principle: Carry out the plan

Polya's Method

Polya's Method breaks the problem into four key principles.

1. First Principle: Understand the problem
2. Second Principle: Make a Plan
3. Third Principle: Carry out the plan
4. Fourth Principle: Look back on your work

Understanding the Problem

1. What are you asked to find or show?

Understanding the Problem

1. What are you asked to find or show?
2. Can you restate the problem in your own words?

Understanding the Problem

1. What are you asked to find or show?
2. Can you restate the problem in your own words?
3. Can you think of useful pictures or diagrams?

Understanding the Problem

1. What are you asked to find or show?
2. Can you restate the problem in your own words?
3. Can you think of useful pictures or diagrams?
4. Is there enough information for you to find a solution?

Understanding the Problem

1. What are you asked to find or show?
2. Can you restate the problem in your own words?
3. Can you think of useful pictures or diagrams?
4. Is there enough information for you to find a solution?
5. Do you understand all the words used in stating the problem?

Understanding the Problem

1. What are you asked to find or show?
2. Can you restate the problem in your own words?
3. Can you think of useful pictures or diagrams?
4. Is there enough information for you to find a solution?
5. Do you understand all the words used in stating the problem?
6. Do you need to ask a question in order to get the answer?

Devise a Plan

Some common plans/methods:

1. Guess and Check

Devise a Plan

Some common plans/methods:

1. Guess and Check
2. Pruning or Elimination of Possibilities

Devise a Plan

Some common plans/methods:

1. Guess and Check
2. Pruning or Elimination of Possibilities
3. Use an ordered list

Devise a Plan

Some common plans/methods:

1. Guess and Check
2. Pruning or Elimination of Possibilities
3. Use an ordered list
4. Make use of symmetry

Devise a Plan

Some common plans/methods:

1. Guess and Check
2. Pruning or Elimination of Possibilities
3. Use an ordered list
4. Make use of symmetry
5. Divide problem into cases

Devise a Plan

Some common plans/methods:

1. Guess and Check
2. Pruning or Elimination of Possibilities
3. Use an ordered list
4. Make use of symmetry
5. Divide problem into cases
6. Use Direct Reasoning

Devise a Plan

Some common plans/methods:

1. Guess and Check
2. Pruning or Elimination of Possibilities
3. Use an ordered list
4. Make use of symmetry
5. Divide problem into cases
6. Use Direct Reasoning
7. Set up and solve an equation

Carry Out the Plan

This is usually easier than figuring out which plan to use. If you are not making progress then you can switch back to the second step and devise a different plan.

Review/Extend

This step is about looking over your solution and trying to understand what worked and what didn't. It's a step which is primarily taken to improve your work on future problems, but is also useful when your solution can be improved.

An interesting example

A couple has three children, Alice, Bob and Carol. Alice Bob and Carol's ages have a product of 36 and a sum of 13. This isn't enough information to solve what their ages are. The father of these children then tells you that the oldest child can play the piano. You now have enough information to solve this problem. What is the solution and why does this happen?

Understanding the Problem

I know that I'll use the sum and product of ages to figure out the ages of the children. I've also been told that that isn't enough information so there is likely to be more than one solution. Somehow knowing that the oldest child plays piano will eliminate all but one of these solutions.

Choosing a method

1. I know that there are a limited number of possible combinations for the ages. I can use the factors of 36 to make a list of all the possible tuples of ages (36 has only 4 factors so there won't be many combinations).

Choosing a method

1. I know that there are a limited number of possible combinations for the ages. I can use the factors of 36 to make a list of all the possible tuples of ages (36 has only 4 factors so there won't be many combinations).
2. I can then rule out any answers that don't sum to 13. This seems like a good approach because it ensures I don't overlook combinations, it doesn't seem like it will take a massive amount of work and since I know there are multiple solutions I don't want to miss any, which looking at all possibilities in a specific order helps with.

3rd Step: Sum and Product of Ages

Product of 36: $36 = 2 * 2 * 3 * 3$

3rd Step: Sum and Product of Ages

Product of 36: $36 = 2 * 2 * 3 * 3$ Since the sum is 13 I'll ignore possibilities with ages above that number. Age possibilites for this are:

3rd Step: Sum and Product of Ages

Product of 36: $36 = 2 * 2 * 3 * 3$ Since the sum is 13 I'll ignore possibilities with ages above that number. Age possibilites for this are: (9,4,1),

3rd Step: Sum and Product of Ages

Product of 36: $36 = 2 * 2 * 3 * 3$ Since the sum is 13 I'll ignore possibilities with ages above that number. Age possibilites for this are: (9,4,1), (9,2,2),

3rd Step: Sum and Product of Ages

Product of 36: $36 = 2 * 2 * 3 * 3$ Since the sum is 13 I'll ignore possibilities with ages above that number. Age possibilites for this are: (9,4,1), (9,2,2), (6,6,1),

3rd Step: Sum and Product of Ages

Product of 36: $36 = 2 * 2 * 3 * 3$ Since the sum is 13 I'll ignore possibilities with ages above that number. Age possibilites for this are: (9,4,1), (9,2,2), (6,6,1), (6,3,2),

3rd Step: Sum and Product of Ages

Product of 36: $36 = 2 * 2 * 3 * 3$ Since the sum is 13 I'll ignore possibilities with ages above that number. Age possibilites for this are: (9,4,1), (9,2,2), (6,6,1), (6,3,2),(4,3,3)

3rd Step: Sum and Product of Ages

Product of 36: $36 = 2 * 2 * 3 * 3$ Since the sum is 13 I'll ignore possibilities with ages above that number. Age possibilites for this are: (9,4,1), (9,2,2), (6,6,1), (6,3,2),(4,3,3)Since the sum is 13 the only possibilities are:

3rd Step: Sum and Product of Ages

Product of 36: $36 = 2 * 2 * 3 * 3$ Since the sum is 13 I'll ignore possibilities with ages above that number. Age possibilites for this are: (9,4,1), (9,2,2), (6,6,1), (6,3,2),(4,3,3)Since the sum is 13 the only possibilities are: (9,2,2)

3rd Step: Sum and Product of Ages

Product of 36: $36 = 2 * 2 * 3 * 3$ Since the sum is 13 I'll ignore possibilities with ages above that number. Age possibilites for this are: (9,4,1), (9,2,2), (6,6,1), (6,3,2),(4,3,3)Since the sum is 13 the only possibilities are: (9,2,2) and (6,6,1).

4th Step: Sum and Product of Ages continued

With these two possibilities (9,2,2) and (6,6,1) we don't know which is possible.

4th Step: Sum and Product of Ages continued

With these two possibilities (9,2,2) and (6,6,1) we don't know which is possible. But once we know the oldest child plays the piano and as such we know there is a unique oldest child. This rules out (6,6,1) and we now know that (9,2,2) is the solution.

4th Step: Sum and Product of Ages continued

With these two possibilities (9,2,2) and (6,6,1) we don't know which is possible. But once we know the oldest child plays the piano and as such we know there is a unique oldest child. This rules out (6,6,1) and we now know that (9,2,2) is the solution. So this seemingly irrelevant statement is actually quite useful.

Was it really 4th step?

That fourth step was really a bit of the 2nd step, we changed our plan to eliminate additional elements of the list using a bit of information that we didn't really understand until after doing step 3 and seeing what the possible answers were.

Sometimes we do things out of order

Sometimes we need to go back to step 2 once we have a better understanding of the problem and make tweaks or start over with a completely different method. Here we need to do this because we didn't initially understand how to use some information from the question. Note that this is rather different from not understanding ”what the solution is”.

Another Example

A computer scientist is working on an AI program in the lab. She has to collect a data set by running her program on test cases on the computers. There are 4 available computers, each of which should only be running one program at a time. There are 23 test cases which are expected to take an average of one hour each, and 1 test case which is expected to take 9 hours. In how long can she expect to finish the task?

Understanding the problem

With only one computer we'd have to schedule all the tasks one after the other and the time would simply be the number of hours taken in total $(23 + 9)$.

Understanding the problem

With only one computer we'd have to schedule all the tasks one after the other and the time would simply be the number of hours taken in total ($23 + 9$). With multiple computers it becomes more complicated. The theoretical best solution is $32/4 = 8$ hours but that isn't going to be possible because one task by itself can take 9 hours. We need to figure out a way to ensure that we find a best possible solution.

Picking a method

This problem naturally falls into a direct reasoning approach. We want to keep track of how many hours we have scheduled so far, and schedule the tasks one at a time, always scheduling the current task on the computer with the most time remaining.

Picking a method

This problem naturally falls into a direct reasoning approach. We want to keep track of how many hours we have scheduled so far, and schedule the tasks one at a time, always scheduling the current task on the computer with the most time remaining. We also need to think about the order in which we schedule the tasks. Lets try scheduling the smallest tasks first. This is wrong but we want to see an example of coming back and trying a new approach.

Step 3: Smallest first scheduling

We schedule the 23 1 hour tasks 1 at a time arriving at (6,6,6,5). for the number of hours each computer is in use $(6 + 6 + 6 + 5 = 23)$.

Step 3: Smallest first scheduling

We schedule the 23 1 hour tasks 1 at a time arriving at (6,6,6,5). for the number of hours each computer is in use $(6 + 6 + 6 + 5 = 23)$.We next schedule the 9 hour task on the computer with the least work so far. This results in (6,6,6,14).

Step 3: Smallest first scheduling

We schedule the 23 1 hour tasks 1 at a time arriving at (6,6,6,5). for the number of hours each computer is in use $(6 + 6 + 6 + 5 = 23)$.We next schedule the 9 hour task on the computer with the least work so far. This results in (6,6,6,14). We can clearly show this answer is wrong as we could have scheduled (7,6,6,4) for the 23 1 hour tasks and then would have (7,6,6,13) a better solution.

Step 3: Smallest first scheduling

We schedule the 23 1 hour tasks 1 at a time arriving at (6,6,6,5). for the number of hours each computer is in use $(6 + 6 + 6 + 5 = 23)$.We next schedule the 9 hour task on the computer with the least work so far. This results in (6,6,6,14). We can clearly show this answer is wrong as we could have scheduled (7,6,6,4) for the 23 1 hour tasks and then would have (7,6,6,13) a better solution. So something is wrong with our method.

Step 2: A new approach

We saw how scheduling the smallest task first caused all sorts of problems, and showing the answer was wrong even offered some insight into why the smallest task first approach didn't work.

Step 2: A new approach

We saw how scheduling the smallest task first caused all sorts of problems, and showing the answer was wrong even offered some insight into why the smallest task first approach didn't work. Ideally what we want to do is get the large tasks out of the way, and distribute the work as evenly as possible on all the computers, which means using a group of smaller tasks to build up towards the larger task. One way to do this is by scheduling the largest task first.

Step 2: A new approach

We saw how scheduling the smallest task first caused all sorts of problems, and showing the answer was wrong even offered some insight into why the smallest task first approach didn't work. Ideally what we want to do is get the large tasks out of the way, and distribute the work as evenly as possible on all the computers, which means using a group of smaller tasks to build up towards the larger task. One way to do this is by scheduling the largest task first. We now have a new approach to creating a good guess.

Step 3: Using the new approach

We start by scheduling the largest task on any of the computers, for covenience I'll use the first one: (9,0,0,0)

Step 3: Using the new approach

We start by scheduling the largest task on any of the computers, for covenience I'll use the first one: (9,0,0,0) Next, we schedule the first one hour task on any of the computer which has the earliest start time: (9,1,0,0)

Step 3: Using the new approach

We start by scheduling the largest task on any of the computers, for covenience I'll use the first one: (9,0,0,0) Next, we schedule the first one hour task on any of the computer which has the earliest start time: (9,1,0,0) Proceeding in this fashion we get to: (9,1,1,1)

Step 3: Using the new approach

We start by scheduling the largest task on any of the computers, for covenience I'll use the first one: (9,0,0,0) Next, we schedule the first one hour task on any of the computer which has the earliest start time: (9,1,0,0) Proceeding in this fashion we get to: (9,1,1,1) , (9,2,2,2)

Step 3: Using the new approach

We start by scheduling the largest task on any of the computers, for covenience I'll use the first one: (9,0,0,0) Next, we schedule the first one hour task on any of the computer which has the earliest start time: (9,1,0,0) Proceeding in this fashion we get to: (9,1,1,1) , (9,2,2,2) , (9,3,3,3)

Step 3: Using the new approach

We start by scheduling the largest task on any of the computers, for covenience I'll use the first one: (9,0,0,0) Next, we schedule the first one hour task on any of the computer which has the earliest start time: (9,1,0,0) Proceeding in this fashion we get to: (9,1,1,1) , (9,2,2,2) , (9,3,3,3) , (9,4,4,4)

Step 3: Using the new approach

We start by scheduling the largest task on any of the computers, for covenience I'll use the first one: (9,0,0,0) Next, we schedule the first one hour task on any of the computer which has the earliest start time: (9,1,0,0) Proceeding in this fashion we get to: (9,1,1,1) , (9,2,2,2) , (9,3,3,3) , (9,4,4,4) , (9,5,5,5)

Step 3: Using the new approach

We start by scheduling the largest task on any of the computers, for covenience I'll use the first one: (9,0,0,0) Next, we schedule the first one hour task on any of the computer which has the earliest start time: (9,1,0,0) Proceeding in this fashion we get to: (9,1,1,1) , (9,2,2,2) , (9,3,3,3) , (9,4,4,4) , (9,5,5,5) , (9,6,6,6) and (9,7,7,7). We've now scheduled the 9 hour task and 21 of the 1 hour tasks.

Step 3: Using the new approach continued

We have (9,7,7,7) and need to schedule the penultimate 1 hour task. It has to go on one of the computers with 7 hours of work as that is the lowest time remaining. For convenience we will put it on the 2nd computer.

Step 3: Using the new approach continued

We have (9,7,7,7) and need to schedule the penultimate 1 hour task. It has to go on one of the computers with 7 hours of work as that is the lowest time remaining. For convenience we will put it on the 2nd computer. This means we've scheduled (9,8,7,7) with just the last task remaining. We will again put it on a computer with 7 hours of work as this is the lowest remaining. This gives us a distribution of tasks of (9,8,8,7) for the 32 total hours of computer time.

Step 3: Using the new approach continued

We have (9,7,7,7) and need to schedule the penultimate 1 hour task. It has to go on one of the computers with 7 hours of work as that is the lowest time remaining. For convenience we will put it on the 2nd computer. This means we've scheduled (9,8,7,7) with just the last task remaining. We will again put it on a computer with 7 hours of work as this is the lowest remaining. This gives us a distribution of tasks of (9,8,8,7) for the 32 total hours of computer time. It will take 9 hours to complete the task as that is the most scheduled on one computer. This is optimal as there is a 9 hour task which can't be split, which means 9 hours is the best possible solution.

Step 4

The key idea was scheduling the larger tasks first so that we could build them up out of the smaller remaining tasks. While the specific numbers in this problem made it somewhat easy to check the work this method appears to be much more general.

Step 4

The key idea was scheduling the larger tasks first so that we could build them up out of the smaller remaining tasks. While the specific numbers in this problem made it somewhat easy to check the work this method appears to be much more general. In fact, in a 3rd year theory course, one often sees a variation of this problem where all the information is general (as variables), and one needs to show this method works for all possible number of computers, numbers of tasks and lengths of tasks. The way to do this is a formal method called an exchange argument, which we will not cover in this course.

Step 4

The key idea was scheduling the larger tasks first so that we could build them up out of the smaller remaining tasks. While the specific numbers in this problem made it somewhat easy to check the work this method appears to be much more general. In fact, in a 3rd year theory course, one often sees a variation of this problem where all the information is general (as variables), and one needs to show this method works for all possible number of computers, numbers of tasks and lengths of tasks. The way to do this is a formal method called an exchange argument, which we will not cover in this course.

Why do computer scientists care about this?

Consider the following code: listOfTasks contains the amount of time units each task takes, and is ordered from largest to smallest, listOfComputers has a length equal to the number of available computers and is initialized to all zeros.

The Code

def taskScheduler(listOfTasks, listOfComputers): for taskIndex in range(0,length(listOfTasks)): minimum = -1 minIndex = -1 for computerIndex in range(0,length(listOfComputers)): if $minimum == -1$: minimum = listOfComputers[computerIndex] minIndex = computerIndex if $minimum > listOfComputers[computerIndex]$: minimum = listOfComputer[computerIndex] minIndex = computerIndex listOfComputers[minIndex] = listOfComputers[minIndex] + listOfTasks[taskIndex]

Implementing a Method

The code above implements the method we described on our specific problem for the general problem.

Implementing a Method

The code above implements the method we described on our specific problem for the general problem. How do we know it works?

Implementing a Method

The code above implements the method we described on our specific problem for the general problem. How do we know it works? Would you want to implement such a method to handle a process for your business if you didn't know it worked?

Implementing a Method

The code above implements the method we described on our specific problem for the general problem. How do we know it works? Would you want to implement such a method to handle a process for your business if you didn't know it worked? In order to know it works we need to know this method is optimal! That means knowing how to prove it.

Implementing a Method

The code above implements the method we described on our specific problem for the general problem. How do we know it works? Would you want to implement such a method to handle a process for your business if you didn't know it worked? In order to know it works we need to know this method is optimal! That means knowing how to prove it. This is part of why theory matters to computer scientists.