CS 209
Data Structures and Mathematical Foundations

02 / 28 / 2024

Instructor: Michael Eckmann

Today's Topics

- Questions/Comments?
- Maximum Contiguous Subsequence problem
  - 3 algorithms to solve
  - Analyze them each for runtime

Algorithm Analysis

- Let's consider 1 problem and 3 ways to solve it (using 3 different algorithms) and we'll analyze the running times of each.
- The Maximum contiguous subsequence problem:
  - Given an integer sequence $A_1, A_2, \ldots, A_N$, find (and identify the sequence corresponding to) the maximum value of $\sum_{k=i}^{j} A_k$. The maximum contiguous subsequence sum is zero if all are negative. Therefore, an empty sequence may be maximum.
- Example input: { -2, **11, -4, 13**, -5, 2 } the answer is 20 and the sequence is { 11, -4, 13 }
- Another: { 1, -3, **4, -2, -1, 6** } the answer is 7, the sequence is { 4, -2, -1, 6 }

Algorithm Analysis

- The simplest is an exhaustive search (brute force algorithm.)
  - that is, simply consider every possible subsequence and compute its sum, keep doing this and save the greatest
  - so, we set the maxSum = 0 (b/c it is at least this big) and we start at the first element and consider every subsequence that begins with the first element and sum them up ... if any has a sum larger than maxSum, save this ...
  - then start at second element and do the same ... and so on until start at last element

- Advantages to this: easy to implement, easy to understand

- Disadvantages to this: slow

- Let's examine the algorithm.
  - decide what is a good thing to count
  - count that operation (in terms of the input size)

Algorithm Analysis

- With a bit of observation it should be apparent that the line that does the summing:

  seq_sum = seq_sum + seq[i]

- is the one that executes most and therefore is the good thing to count.
- The two outer loops are similar to ones we had analyzed in other contexts before to know that they execute n*(n+1)/2 times. The innermost loop iterates a different number of times for each of the outer loops iterations. It iterates n times, n-1 times twice, n-2 times sometimes, ...
- It turns out that the number of times the summing line executes is: n*(n+1)*(n+2)/6 which is a Theta(n^3) algorithm.

Algorithm Analysis

- The exhaustive search has many unnecessary computations.
- Notice that $\sum^{j}_{k=i} A_k = A_j + \sum^{j-1}_{k=i} A_k$
- That is, if we know the sum of the first j-1 elements, then the sum of the first j elements is found just by adding in the jth element.
- Knowing that, the problem can be solved more efficiently by the algorithm that we are about to write and analyze.
  - We won't need to keep adding up a sequence from scratch

Algorithm Analysis

- The second algorithm that we'll analyze uses the improvement just mentioned and the running time improves (goes down.)

Algorithm Analysis

- A further improvement can come if we realize that if a subsequence has a negative sum, it will not be the first part of the maximum subsequence.
  - Why?
- Also, all contiguous subsequences bordering a maximum contiguous subsequence must have negative or 0 sums.
  - why?

Algorithm Analysis

- A further improvement can come if we realize that if a subsequence has a negative sum, it will not be the first part of the maximum subsequence.
  - Why?
  - A negative value will only bring the total down
- Also, all contiguous subsequences bordering a maximum contiguous subsequence must have negative or 0 sums.
  - why?
  - If they were >0, they would be attached to the maximum sequence (thereby giving it a larger sum).

Algorithm Analysis

- While computing the sum of a subsequence, if at any time the sum becomes negative, we start considering sequences only **starting** at the next element.
- Let's write this algorithm and determine the running time of it.

Algorithm Analysis

- What's the point of that exercise:
  - 1) get a feel for how to count how much work is being done in an algorithm
  - 2) it is sometimes possible to create a reduced running time algorithm by exploiting facts about the problem.
  - 3) it is good to think about such things in a course that mainly deals with data structures. Any guesses as to why I say this?
  - 4) it takes some thinking to exploit some things about the problem to make a more efficient algorithm

Proof by induction

Prove that $2^0 + 2^1 + \ldots + 2^n = 2^{(n+1)} - 1$ (call this proposition: P(n))

Base case: when n=0 (show that P(0) is true)

$2^0 = 1$

And $2^{(0+1)} - 1 = 2 - 1 = 1$. These are equal so P(0) is true.

Induction step:

Assume that P(k) is true: $2^0 + 2^1 + \ldots + 2^k = 2^{(k+1)} - 1$

Try to show that P(k+1) is true.

$2^0 + 2^1 + \ldots + 2^k + 2^{k+1}$

$= 2^{(k+1)} - 1 + 2^{k+1}$

$= 2*2^{(k+1)} - 1 = 2^{(k+2)} - 1$ This shows that P(k+1) is true.