CS358: Applied Algorithms

Assignment 1: Two Towers Revisited (due 9/22/21)

Instructor: Sam McCauley

Instructions

All submissions are to be done through github. This process is detailed in the handout "Handing In Assignments" on the course website. Answers to the questions below should be submitted by editing this document. All places where you are expected to fill in solution are marked in comments with "FILL IN."

Please contact me at srm2@williams.edu if you have any questions or find any problems with the assignment materials.

Problem Description

INPUT: The input to the problem will be an array of at most 64 signed 64-bit integers (this array will be given as a pointer of type `int64_t*`), along with an integer representing how many elements there are in the array.

OUTPUT: The output is a single signed 64-bit integer whose bits represent a subset of the blocks in the array. For example, the subset consisting of only the first element in the array would be represented by the integer 1; the subset consisting of the first, third, and fourth elements of the array would be represented by 13.

GOAL: The input represents a set of blocks; the ith integer in the input represents the "area" of the ith block. The height of a block is the square root of its area.

The goal is to partition the blocks into two sets (which we call towers), such that the height of the towers is as close as possible. The value returned should be the blocks that make up the smaller of these two towers.

Testing Parameters

The `main()` method of the testing program (in `test.c`) takes two arguments, each of which is a file containing instances of the two towers problem. An instance of the problem is represented by a sequence of integers (separated by spaces) on one line, and the intended solution (represented as a decimal number) on the second line. You can test your program by first running `make`, and then running `./test.out testData.txt timeData.txt`.

- For all instances, the best solution is guaranteed to be at least .0001 larger than the second-best solution (i.e. the smaller tower in the optimal solution is .0001 larger than

any other tower smaller than half the height). This guarantee is to help rule out issues with floating point errors. This also means that the smaller tower is strictly smaller than the larger tower—you do not need to worry about tiebreaking.

- These instances were tested using 64-bit `double`s. Higher-precision numbers such as `long double`s are acceptable (and are available on the lab computers), but are not necessary to obtain a correct solution. 32-bit `float`s may not be sufficient.
- All input block areas will be between 1 and 9223372036854775807 (this is the largest number that fits in a signed 64 bit integer). For the timed testing case, these inputs were (essentially) randomly generated. This means that you can assume that the inputs are approximately randomly distributed. However, your algorithm should work for any input.
- Two solutions with running times within .1 seconds of each other will be considered tied for the purposes of this assignment.

Questions

Code (50 points). Implement the meet in the middle algorithm for the two towers algorithm. Please give a very brief (2-3 sentence) description of your approach below.

Solution.

Problem 1 (15 points)**.** What proportion of your algorithm's running time is spent on sorting?[1]

Describe specifically how you performed your experiment. How did you measure the time it took? What input did you use? Did the experiment entail any changes to the running of your implementation that might change its performance?

Solution.

Problem 2 (10 points)**.** Can meet in the middle be modified to efficiently return the k best solutions, rather than just the best? Describe how this modification could be made and analyze its running time (the running time should be in terms of n, the number of blocks, and k, the desired number of solutions—you should not assume that k is a constant). Make sure your methodology is correct and is clearly described.

Solution.

[1] It's OK if the version of your implementation that you use for this experiment does not exactly match the version you hand in. If your implementation does not use sorting at all, design and perform an experiment to estimate the time a sort would take compared to your implementation's total running time.

Problem 3 (25 points)**.** Consider a similar problem: again we are given a list of n blocks (let's call it S), but now we want to partition them into *three* towers such that the heights are as close as possible. In particular, the goal is to minimize

$$\text{the height of the tallest tower} - \text{the height of the smallest tower.}$$

In this problem you will design a meet in the middle approach to solve this problem. I have broken the question into several parts below to help you find a solution.

(a) First, give a simple algorithm to solve this problem in $O(n3^n)$ time. How much space does it require?

Solution.

(b) Now, let us consider two sets S_1 and S_2 that partition S (i.e. they satisfy $S_1 \cap S_2 = \emptyset$ and $S_1 \cup S_2 = S$). Consider an assignment of each element of S_1 to the small, medium, and large tower; let's call this assignment A_1. Given assignment A_1, what constraints are there on an assignment A_2 that assigns each element of S_2 to the small, medium, and large tower?[2] Of assignments that meet this constraint, which gives us the best overall solution?

In particular, let's say s_1, m_1, and ℓ_1 are the total heights of blocks that A_1 assigns to the small, medium, and large tower respectively; similarly, let's say s_2, m_2, and ℓ_2 are the assignments to the small, medium, and large tower for A_2. This question is asking for constraints on s_2, m_2, and ℓ_2 in terms of s_1, m_1, and ℓ_1—they'll help us in part (c).

Solution.

(c) Using your ideas from part (b), give an $O(n3^{n/2})$ space, $O(n2^{n/2})$ time method that can, for any assignment of elements of S_1 to the three towers, find an optimal assignment of elements from S_2 to the three towers.

Hint: Let's say a set of elements $S' \subseteq S_2$ is assigned to the smallest tower. How quickly can you find the best way to assign the remainder of S_2 to the middle and tallest tower? This is where you should be precomputing a lookup table.

Solution.

(d) Using part (c), give an $O(3^{n/2} \cdot n2^{n/2})$-time,[3] $O(n3^{n/2})$-space algorithm to find the optimal solution to a three towers instance.

Solution.

[2] In particular, given A_1, we'd better ensure that any assignment A_2 of elements of S_2 results in the "small tower" having the smallest total height, and the "large tower" having the largest total height. This is similar to how we, in the original two towers problem, assigned elements to the smaller of the two towers and capped its height at half the sum of all elements in S.

[3] This is approximately $O(n3^{.82n})$, quite a bit faster than $O(n3^n)$.

(e) Extra credit (5 points): Improve upon your result from part (d)—give an asymptotically faster algorithm. (The improvement needs to be exponential; improving by a $O(n)$ factor is not enough.)

Solution.

Tips and tricks

- Remember to create a correct program before worrying about creating a fast one!
- You need to iterate through all possible subsets. One way to do this is to store each subset as an integer, and increment it to obtain a new subset. If you are unfamiliar with this technique, it may be useful to read the version of this lab given in CSCI 136, which can be found here for example: https://williams-cs.github.io/cs136-s21-www/labs/two-towers.html (there's also now a video for that lab here: https://www.youtube.com/watch?v=MFU0hGOWvuk).
- Since the problem asks for the items that make up the solution, your meet in the middle approach will need to keep track of both the cost of a subset, and the items in the subset. There are many ways to do this. One easy way may be to keep track of a subset using a 64-bit integer, and to keep track of a partial solution using a struct containing both the cost of the subset and the integer representing its items. Other solutions are also possible, such as storing two arrays (one for costs and one for subsets), or storing a hash table keeping track of which solution corresponds to which cost.