Homework 5

CS 4104 (Fall 2021)

Assigned on October 28, 2021.
Submit PDF solutions on Canvas by the
11:59pm on November 4, 2021.

Instructions:

- The Honor Code applies to this homework with the following exception:
  - You can pair up with another student to solve the homework. Please form teams yourselves. Of course, you can ask the instructor for help if you cannot find a team-mate. You may choose to work alone.
  - You are allowed to discuss possible algorithms and bounce ideas with your team-mate. **Do not discuss proofs of correctness or running time in detail with your team-mate. You must write down your solution individually and indepedently. Do not send a written solution to your team-mate for any reason whatsoever.**
  - *In your solution, write down the name of the other member in your team. If you do not have a team-mate, please say so.*
  - Apart from your team-mate, you are not allowed to consult sources other than your textbook, the slides on the course web page, your own class notes, the TAs, and the instructor. In particular, do not use a search engine.
- Do not forget to typeset your solutions. *Every mathematical expression must be typeset as a mathematical expression, e.g., the square of n must appear as n^2 and not as "n^2".* You can use the LaTeX version of the homework problems to start entering your solutions.
- Do not make any assumptions not stated in the problem. If you do make any assumptions, state them clearly, and explain why the assumption does not decrease the generality of your solution.
- You must also provide a clear proof that your solution is correct (or a counter-example, where applicable). Type out all the statements you need to complete your proof. *You must convince us that you can write out the complete proof. You will lose points if you work out some details of the proof in your head but do not type them out in your solution.*
- If you are proposing an algorithm as the solution to a problem, keep the following in mind (the strategies are based on mistakes made by students over the years):
  - Describe your algorithms as clearly as possible. The style used in the book is fine, as long as your description is not ambiguous. Explain your algorithm in words. A step-wise description is fine. *However, if you submit detailed code or pseudo-code without an explanation, we will not grade your solutions.*
  - Do not describe your algorithms only for a specific example you may have worked out.
  - Make sure to state and prove the running time of your algorithm. You will only get partial credit if your analysis is not tight, i.e., if the bound you prove for your algorithm is not the best upper bound possible.
  - You will get partial credit if your algorithm is not the most efficient one that is possible to develop for the problem.
- In general for a graph problem, you may assume that the graph is stored in an adjacency list and that the input size is $m + n$, where n is the number of nodes and m is the number of edges in the graph. Therefore, a linear time graph algorithm will run in $O(m + n)$ time.

My team-mate is ______________________

Problem 1 (15 points) In this problem, you will analyse the worst-case running time of weighted interval scheduling *without* memoisation. Recall that we sorted the n jobs in increasing order of finish time and renumbered these jobs in this order, so that $f_i \leq f_{i+1}$, for all $1 \leq i < n$, where f_i is the finish time of job i. For every job j, we defined $p(j)$ to be the job with the largest index that finishes earlier than job j. Consider the input in Figure 6.4 on page 256 of your textbook. Here all jobs have weight 1 and $p(j) = j - 2$, for all $3 \leq j \leq n$ and $p(1) = p(2) = 0$. Let $T(n)$ be the running time of the dynamic programming algorithm *without memoisation* for this particular input. As we discussed in class, we can write down the following recurrence:

$$T(n) = T(n-1) + T(n-2), n > 2$$
$$T(2) = T(1) = 1$$

Prove an *exponential lower* bound on $T(n)$. Specifically, prove that $T(n) \geq 1.5^{n-2}$, for all $n \geq 1$.

Problem 2 (25 points) After graduation, you start a job at a company that is building a space elevator! People travelling to space in the elevator need to eat. Therefore, your company is planning to open a series of restaurants along the elevator. There are n potential locations for these restaurants. Starting from the beginning of the space elevator, these locations are in miles and in increasing order, $m_1, m_2, m_3, \ldots m_{n-1}, m_n$. Your company wants to minimize the costs of constructing and maintaining these restaurants as well as maximising its profits. It devises the following rules:

(i) At each location, it can open at most one restaurant.

(ii) For each $1 \leq i \leq n$, the expected profit from opening a restaurant at location i is $p_i > 0$.

(iii) Any two restaurants your company builds should be at least k miles apart, where k is a positive integer.

Since the company hired you because you have taken CS 4104, it asks you to devise an efficient algorithm to decide where to build the restaurants so as to compute the maximum total profit subject to the given rules. Use your advanced knowledge of algorithms to impress your bosses and peers and collect a hefty Christmas bonus!

Problem 3 (25 points) There are many ways to formulate the cost of a secondary structure for an RNA molecule. In class, we defined the cost as the number of matching base pairs in the secondary structure and sought to maximise this quantity. Here is another approach. Suppose S is a secondary structure for an RNA molecule $b_1 b_2 \ldots b_{n-1} b_n$ that obeys all the rules we described in class. We define a *gap* as a maximal, consecutive sequences of indices that do not appear in S. More formally, a gap is a subsequence $b_k b_{k+1} \ldots b_{m-1} b_m$ where for every $k \leq l \leq m$ b_l, is not matched with any other base in S but both b_{k-1} and b_{m+1} are matched in S (not necessarily with each other). The *length* of the gap is $m - k + 1$. Define the cost of S as the sum of the lengths of the gaps. See Figure 1 for an illustration. Devise a recurrence relation to minimise this quantity. If you can derive the recurrence clearly from first principles (as we did in class), this derivation will serve as the proof of correctness. Note that I am not asking you to devise an algorithm to evaluate this recurrence. *Hint:* One recurrence may not be sufficient.

Problem 4 (35 points) Many object-oriented programming language implement a class for manipulating strings. A primitive operation supported by such languages is to split a string into two pieces. This operation usually involves copying the original string. Hence, it takes n units of time to split a string of length n into two pieces, *regardless of the location of the split.* However, if we want to split a string into many pieces, the order in which we make the splits can affect the total running time of all the splits.

For example, suppose we want to split a 20-character string at positions 3 and 10. If we make the first cut at position 3, the cost of the first cut is the length of the string, which is 20. Now the cut

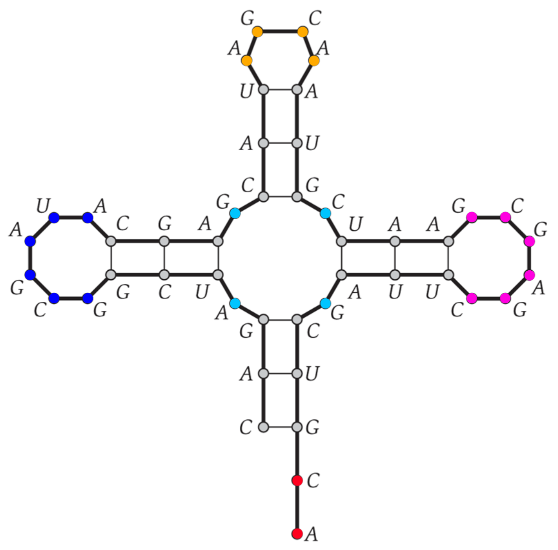

Figure 1: An illustration of gaps in an RNA secondary structure. There are four gaps of length one (light blue, in the centre) , one gap of length two (red, at the bottom), one gap of length four (orange, at the top), and two gaps of length six (dark blue and purple, on the left and right, respectively.) To understand what we mean by a gap being maximal, consider the orange sequence AGC at the top. It is not a gap since it is a part of the sequence AGCA, which is maximal and matches our definition of a gap, consequently.

at position 10 falls within the second string, whose length is 17, so the cost of the second cut is 17. Therefore, the total cost is 20 + 17 = 37. Instead, if we make the first cut at position 10, the cost of this cut is still 20. However, the second cut at position 3 falls within the first string, which has length 10. Therefore, the cost of the second cut is 10, implying a total cost of 20 + 10 = 30.

Design an algorithm that, given the locations of m cuts in a string of length n, finds the minimum total cost of breaking the string into $m + 1$ pieces at the given locations, minimised over all possible ways of breaking the string at the m locations. You can assume that if you make a cut at location i in a string of length n, you get two sub-strings, one of length i (from position 1 to position i) and the other of length $n - i$ (from position $i + 1$ to position n).

Let us define some notation to help develop the solution. Sort the locations of the m cuts in increasing order along the length of the string. Let c_i be the location of the ith cut, $1 \leq i \leq m$. Set $c_0 = 1$, and $c_{m+1} = n$, locations at the beginning and the end of the string, respectively. Note that we can assume, without loss of generality, that no cut has been specified at location n, since making this cut incurs no cost.

Hint: Suppose you make the first cut at some location c_j, where $1 \leq j \leq m$. This cut will split the string into two sub-strings. Consider where you may make the next cut in each sub-string. What types of sub-problems are you creating? In other words, each sub-problem is a sub-string: how do you denote or characterise this sub-string? Does it look like a sub-problem in weighted interval scheduling, segmented least squares, or RNA secondary structure? Figuring out the right set of sub-problems will go a long way towards helping you solve the problem. It will also help to define some notation for each sub-problem and for the cost of solving it.