CPSC 320 2016W2: Assignment #5

March 16, 2017

Please submit this assignment via GradeScope at https://gradescope.com. Detailed instructions about how to do that are pinned to the top of our Piazza board: https://piazza.com/ubc.ca/winterterm22016/cpsc320/. Briefly, your GradeScope account **must** use the "GradeScope Student #" we distributed in your Connect gradebook so that we can link your account with you!

Submit by the deadline **Friday 24 Mar at 10PM**. For credit, your group must make a **single** submission via one group member's account, marking all other group members in that submission **using GradeScope's interface**. Your group's submission **must**:

- Be on time.
- Consist of a single, clearly legible file uploadable to GradeScope with clearly indicated solutions to the problems. (PDFs produced via LaTeX, Word, Google Docs, or other editing software work well. Scanned documents will likely work well. **High-quality** photographs are OK if we agree they're legible.)
- Include prominent numbering that corresponds to the numbering used in this assignment handout (not the individual quiz postings). Put these **in order** starting each problem on a new page, ideally. If not, **very clearly** and prominently indicate which problem is answered where!
- Include at the start of the document the **GradeScope Student #s** of each member of your team. (No names are necessary.)
- Include at the start of the document the statement: "All group members have read and followed the guidelines for academic conduct in CPSC 320. As part of those rules, when collaborating with anyone outside my group, (1) I and my collaborators took no record but names (and GradeScope information) away, and (2) after a suitable break, my group created the assignment I am submitting without help from anyone other than the course staff." (Go read those guidelines!)
- Include at the start of the document your outside-group collaborators' GradeScope student #s or account names. (Be sure to get those when you collaborate!)

1 Ruler of My Non-Domain

There's no domain information for this problem. Amazing but true! :)

1.1 Do Be So Naïve

Consider the following recurrence:

$$A(i,j) = \begin{cases} 1 & \text{if } i = 1 \\ A(i-1, j+1) + A(i-1, j) & \text{otherwise} \end{cases}$$

CLARIFICATION: It **is** possible to substantially simplify this recurrence. Do **not** do so. Work with the recurrence as given instead.

1. Give naïve recursive code that computes the value of $A(n, 1)$ for $n \geq 1$.

2. Give an asymptotic bound on the **runtime and memory use** of a memoized version of this algorithm. (Assume that storing one value of A takes constant space.)

3. Write clear (pseudocode) nested loops to specify an order could solve the subproblems to convert this to a dynamic programming solution. (There is no need to write the initialization code for the function or the body that would actually solve the problem.)

1.2 Take a Memo!

Assume that you have an array of integers C with indexes $1 \ldots n$.

Consider the following recurrence (**updated** slightly so $j < i$):

$$A(i) = \begin{cases} 1 & \text{if } i \text{ is a power of 2} \\ \min_{\lceil \frac{i}{2} \rceil \leq j < i}(A(j) + C[i] - C[j]) & \text{otherwise} \end{cases}$$

Give pseudocode for a memoized version of this algorithm. You may not use global variables (although you may assume you have a language with nested scopes/functions if you like). Your solution should take C (and n if desired) as a parameter.

UPDATES: Feel free to assume you have a helper function `IsPow2(x)` that returns true if x is a power of 2. The bound on j has also been updated to be $< i$ rather than $\leq i$.

1.3 Be a Dynamo!

Assume that you have an array of integers C with indexes $1 \ldots n$.

Consider the following recurrence:

$$A(i) = \begin{cases} 1 & \text{if } i < 5 \text{ or } i > n - 5 \\ \min(C[i-1] + A(i-1), C[i-2] + A(i-2)) & \text{if } 5 \leq i < \lfloor \frac{n}{2} \rfloor \\ \min(C[i-1] + A(i-1), C[i+1] + A(i+1)) & \text{if } i = \lfloor \frac{n}{2} \rfloor \\ \min(C[i+1] + A(i+1), C[i+2] + A(i+2)) & \text{otherwise} \end{cases}$$

Convert this to an efficient dynamic programming solution that computes $A(\lfloor \frac{n}{2} \rfloor)$. Your solution should take C (and n if desired) as a parameter.

2 Parking in Wonderland

RECALL the parking optimization problem from quiz 3:

> A company called Wonderland offers several types of parking permits to its employees, with different durations and prices. The co-op student Alice will work in Wonderland for n consecutive days. She wants to figure out the cheapest collection of parking permits that would cover all days she needs to be present at work. Alice can buy as many permits of a given type as she likes. Let's assume there are k type of permits $1, \ldots, k$: the price of permit type t is p_t dollars and duration d_t. Alice needs to stay at work on n consecutive days. Our goal is to help Alice figure out how to park for all n days as cheaply as possible.

2.1 Permission Accomplished

Assume we're only interested in the optimal cost for parking over n days. Assume the durations of each of the k permits are stored in the array D and the prices are stored in the array P.

1. Give a recurrence $C(n)$ that describes the optimal cost to park for $n > 0$ days, in terms of the optimal cost to park for some smaller number(s) of days.
2. Give the corresponding base case(s) for your recurrence.
3. Assuming k (the number of types of permits) is a constant, design a linear-time algorithm to compute the optimal parking cost by converting your recurrence relation above into a memoized or dynamic programming solution.

2.2 Where Did We Park? (assignment only)

Give pseudo-code for a function that takes the table from your memoized or dynamic programming solution to the previous problem and computes the actual set of parking permits to purchase. (I.e., an "explain" function for this problem.)

3 Pwner of All I Survey

You're managing an online survey. You have a training set of responses to the survey. From this you have extracted the average time spent on each of the survey's n questions (numbered 1 through n).

You're using this to optimize later offerings of the same survey. In particular, you are "paginating" the survey. You are leaving the questions in the same order but breaking them up into pages. (So, for example, questions 1–4 may be on page 1, in which case question 5 will start page 2. Each page will have at least one question.)

You're particularly interested in the average time a page takes (i.e., the total of the average times of the questions on that page) compared to the maximum time that any one question takes on average, which we call m. You have decided that no page should take more than $2m$ time (on average).

3.1 A Profound Dis-Likert for Greedy

You're considering the following greedy approach:

> Put as many problems as possible without going over the time limit of $2m$ on the first page. Then, recursively solve the problem for any remaining questions.

With the exception of the last page, you **want** each page to take as close as possible to $2m$ time. Further, no page is allowed to go over $2m$ time.

You decide to "score" a pagination in the following way. Let the amount of time a page i takes be t_i. Pages before the last each score $(2m - t_i)^2$ points. (The last page always scores 0 points.) You want to **minimize the total score across all pages**.

Problem: Give an instance that shows that the greedy approach is not optimal. Be sure to indicate (1) the solution the greedy approach produces, (2) the score the greedy approach earns, (3) the optimal solution, and (4) the score of the optimal solution. *Suggestion:* putting the longest question last makes it easier to deal with its impact on the scores of the pages.

3.2 A Fair and Balanced Survey

With the exception of the last page, you **want** each page to take as close as possible to $2m$ time. Further, no page is allowed to go over $2m$ time.

You decide to "score" a pagination in the following way. Let the amount of time a page i takes be t_i. Pages before the last each score $(2m - t_i)^2$ points. (The last page always scores 0 points.) You want to **minimize the total score across all pages**.

Problem: Give a naïve recursive (brute force) algorithm—suitable for conversion to a memoized or dynamic programming form—to compute the score of the optimal solution to this problem.

4 Seam Carving

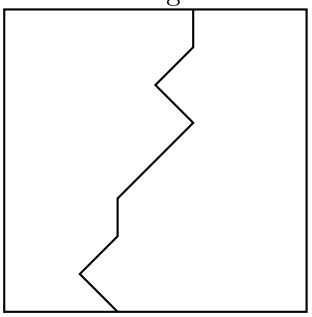
You can resize an image by scaling or cropping it, but what if the pieces of the image that you want are not all in one rectangular area, and you don't want to make those parts of the image smaller by scaling?[1]

In that case, you might instead choose to eliminate one pixel from each row (to make the image one pixel narrower) or one pixel from each column (to make the image shorter) while somehow optimizing for the "best" pixels to remove. In this problem, we focus on removing one pixel from each row.

We'll assume an image is an n row by m column array of pixels $A[1 \ldots n][1 \ldots m]$, where each pixel is an "energy" rather than a color. Energies are non-negative numbers representing the importance of the pixel. (**CORRECTION:** the words "row" and "column" were swapped on the quiz version.)

A legal seam must include one pixel from every row. Each pair of seam pixels in neighbouring rows must be either in the same column or one column apart (i.e., on a diagonal). The cost of a seam is the total energy of all the pixels in the seam. The best seam is the one with lowest cost.

So, a seam of pixels to remove often looks a little like a "lightning bolt" moving down, down-and-left, and down-and-right from the top to the bottom of the image, such as this:



4.1 Seamingly Simple

1. Circle two non-overlapping seams in this diagram that have different costs. Indicate their costs and which seam is better:

```
1 8 7 5 6 2 4
9 5 1 2 8 8 7
6 6 2 1 9 5 4
```

2. Give the cost for a partial seam that only has a pixel in the very first row, $i = 1$. (This is our base case.)

```
C(1, j) =
```

[1] This method was developed by Avidan and Shamir.

4.2 Seamy Details

Give a recurrence for the cost of the best partial seam that has one pixel in each of rows 1 down to i and ends at the pixel in row i and column j. Your recurrence should be in terms of the seams ending at pixels in the row above, row $i-1$. Assume $i > 1$.

`C(i, j) =`