CPSC 320 2016W1: Assignment 5

November 11, 2016

Submit this assignment via handin (see the syllabus for more information) to the target `assn5` by the deadline **Thursday 24 Nov at 10PM**. For credit, your group must make a **single** submission via one group member's account. Your group's submission **must**:

- Be on time.
- Consist of a single, clearly legible PDF file named `solution.pdf` with clearly indicated solutions to the problems. (Directly produced via LaTeX, Word, Google Docs, or other editing software is best. Scanned is fine. **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**, ideally. If not, **very clearly** and prominently indicate which problem is answered where!
- Include at the start of the document the names and ugrad.cs.ubc.ca account IDs of each member of your team.
- 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 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 acknowledgment of collaborators and references (with the exceptions listed in the conduct guidelines).

1 Lowest-Cost (Not So) Simple Path

Imagine a weighted, directed graph G where edge weights may be positive, negative, or zero. We will consider the problem of finding the lowest-cost simple path between a source node s and terminal node t in such a graph. We'll call this problem GENSHORT for "general shortest path". (Recall that a **simple** path is a path with no vertex repeated, i.e., with no cycles.)

(Recall that the Bellman-Ford Algorithm—as presented in our text—finds the shortest path from any start vertex in the graph to a single terminal vertex t. It proceeds using dynamic programming using a table parameterized by which node is being considered as s and the maximum number of edges in the path from s to t. The first column (where the maximum number of edges is 0) has ∞ for all nodes except t itself and 0 for t. On each iteration, it updates each row s in the next column based on the lowest-cost path of all those that go from s to some node u (in one edge) and then from u to t using the already-computed value in the previous column.)

1. **Very briefly** explain why the Bellman-Ford algorithm cannot in general be used to solve GENSHORT.

This work is licensed under a Creative Commons Attribution 4.0 International License.
For license purposes, the author is the University of British Columbia.

2. Give a small instance of GENSHORT on which the Bellman-Ford algorithm **will** find the lowest-cost simple path from s to t. Be sure to indicate what that lowest-cost simple path is.

3. Here is a proposed reduction from GENSHORT to the problem of finding the lowest-cost simple path between a source node s and terminal node t in a weighted, directed graph with **only non-negative edge weights**:

   Reduction: Given the graph G that may contain negative edge weights, find the edge with minimum weight w_{min} (by scanning through all edges) and subtract w_{min} from the weight of every edge to create graph G'. In G' the minimum weight edge has weight 0, and no edge has negative weight. Find the lowest-cost simple path between s and t in G' (i.e., call on the solution to the underlying problem), and then return this list of vertices as the lowest-cost simple path in the original graph. (Of course, the edges connecting the vertices have different weights in G, but it's still the same path.)

   Give a small instance of GENSHORT on which this reduction does **not** produce the optimal solution. Indicate the solution produced by the reduction and the optimal solution.

1.1 NP-Completeness

In this part, we will consider a decision-variant of GENSHORT. In this variant, we add a number k to the format of an instance. The solution to the instance is YES if a simple path from s to t exists with cost less than or equal to k; otherwise, the solution is NO.

1. Prove—by reducing from the HAMPATH problem to GENSHORT—that GENSHORT is NP-hard. (Note: HAMPATH is NP-complete.) *Hint:* it may help to add a couple of nodes to be s and t. When thinking about edges to and from those nodes, consider that you can have zero-weight edges.

2. Prove that the decision version of GENSHORT is in NP by showing it is "efficiently certifiable". First, select a certificate. (Think of how you would describe the solution to the **original** version of GENSHORT.) Then, show how to prove in time polynomial in the size of the decision-variant GENSHORT instance that the answer to the decision problem is YES given such a certificate. (A decision-variant GENSHORT instance is a graph plus one extra number; think of its size as $O(n+m)$ as usual for graphs.)

(This isn't required, but you might want to work through how you could solve the original variant of GENSHORT using a polynomial number of calls to the decision-variant.)

2 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.

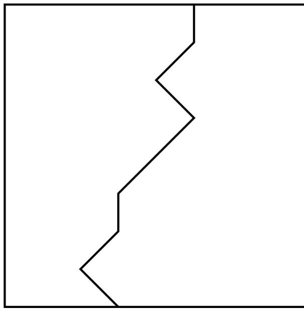
We'll assume an image is an n column by m row array of pixels $A[1 \dots n][1 \dots m]$, where each pixel is an "energy" rather than a color. Energies are non-negative numbers representing the importance of the pixel.

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:

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

This work is licensed under a Creative Commons Attribution 4.0 International License.
For license purposes, the author is the University of British Columbia.



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 a recurrence for the cost of the best partial seam that has pixels only from row 0 up 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 $i > 0$.

```
C(i, j) =
```

3. Give the cost for a partial seam that only has a pixel in the very first row, $i = 0$. (This is our base case.)

```
C(0, j) =
```

2.1 Dynamic Programming

1. Give a pseudocode algorithm that finds the cost of the best seam in an energy array using dynamic programming.
2. Give a pseudocode algorithm that takes the dynamic programming table and produces the column numbers of the pixels in the best seam. (So, if the best seam has a pixel at column 3 in the first row and column 4 in the second, then your solution should give the list $[3, 4]$.)

2.2 Bonus: Implement Awesomely

Look up seam carving and design an implementation. Maybe extend it to apply to video!

3 Transformers

In the ELEC problem, you're given a network of electrical wires which can be represented as a directed, acyclic graph (DAG) with three types of nodes:

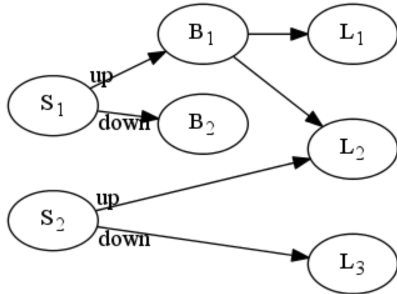
- "Switch" nodes supply power. They have **no** wires coming in and two wires going out labeled "up" and "down". They also have a switch. If the switch is in the up position, then power (electricity) flows into the up wire. If the switch is in the down position, then power flows into the down wire.
- "Branch" nodes can have one wire coming in (which may or may not carry power) and any number of wires going out. If the wire coming in carries power, then all wires going out also carry power. Otherwise, none of the wires carries power.

This work is licensed under a Creative Commons Attribution 4.0 International License.
For license purposes, the author is the University of British Columbia.

- "Load" nodes represent electrical devices that must be powered. They have one or more wires coming in and none going out. If any wire coming in carries power, the load is powered. Otherwise, it is not.

The solution to an ELEC instance is YES if some configuration of the switches powers all the loads; otherwise, it's NO.

1. Indicate a configuration of the switches in the following network that powers all the loads by writing "up" or "down" on each switch node. (Switch nodes are labeled S, branch nodes B, and load nodes L.)



2. Give a reduction from SAT to ELEC. *Hint:* Consider that a variable can be positive or negated, the positive (or negated) literal can appear in many clauses, and each clause needs at least one true literal in it.

3.1 NP-Completeness

You have already shown that ELEC is NP-hard with your reduction from SAT to ELEC above. Now, you will show that ELEC is in NP. (Together, the fact that it is NP-hard and in NP makes it NP-complete.)

1. Give a (polynomial-length) certificate for ELEC instances where all loads can be powered. *Hint:* We already asked you for your "solution" to an ELEC problem above. What form does such a solution take?

2. Give an algorithm that takes polynomial time in the size of an ELEC instance to determine whether all loads can be powered in that instance, given a certificate like the the one you describe above.

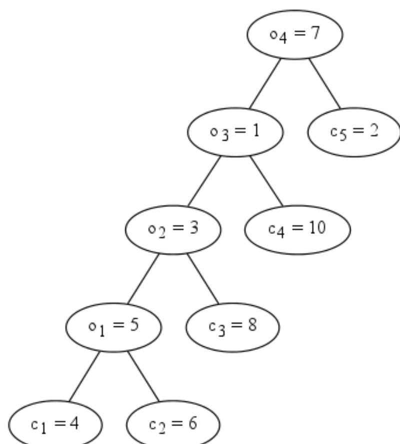
4 The Benefits of Preparallelation?

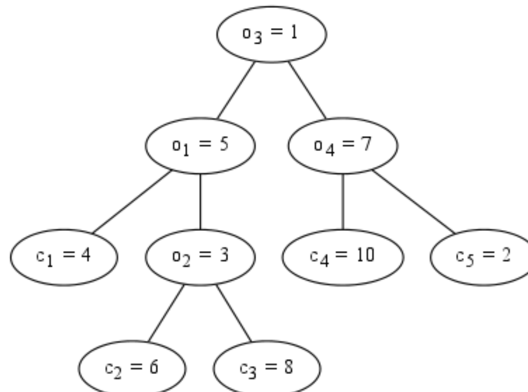
You're setting up a process to run in parallel on a huge array of processors.[2] The process is a chain of $n \geq 1$ computations $c_1, \ldots, c_n$ connected by $n-1$ operations $\circ_1, \ldots, \circ_{n-1}$ of the form $(c_1 \circ_1 c_2 \circ_2 \ldots \circ_{n-2} c_{n-1} \circ_{n-1} c_n)$. You receive an array $C[1 \ldots n]$ of costs of each computation c_i and an array $D[1 \ldots n-1]$ of the cost of each operator $\circ_i$. The process runs on n processors, each running a single operation c_i in time $C[i]$. Then, one of two processors with results from neighbouring computations collects results from the other processor and combines them in time $D[j]$ using the appropriate operator $\circ_j$.

The way the process is split across processors and then combined by $\circ$ operators forms a binary tree. For example, with $n = 5$ processes with costs $C = [4, 6, 8, 10, 2]$ and $D = [5, 3, 1, 7]$, we might combine the results in the fashion described by one of these two trees:

[2] Skippable note: "Huge" means big enough for asymptotics to matter. For example, the Sunway TaihuLight supercomputer in China has over 10 million cores, which is 3 times as many as last year's biggest.

This work is licensed under a Creative Commons Attribution 4.0 International License.
For license purposes, the author is the University of British Columbia.





The left-hand tree combines the results from c_1 with the results from c_2 first, adds in c_3's results, then c_4's, and then c_5's. The right-hand combines the result from c_1 with the combination of c_2's and c_3's results, and combines that result with the combination of c_4's and c_5's results.

The overall cost of a way to split up the process (i.e., a tree) is the maximum cost path from the root to a leaf (because that is the longest-running series of non-parallelizable steps). The left tree's cost is $7 + 1 + 3 + 5 + 6 = 22$. The right tree's cost is lower: $1 + 7 + 10 = 18$. So, the right tree is a better way to split up the process.

1. Give a pseudocode algorithm that—given a binary tree like the diagrams above (i.e., a pointer to its root)—computes the overall cost of that tree in linear time.

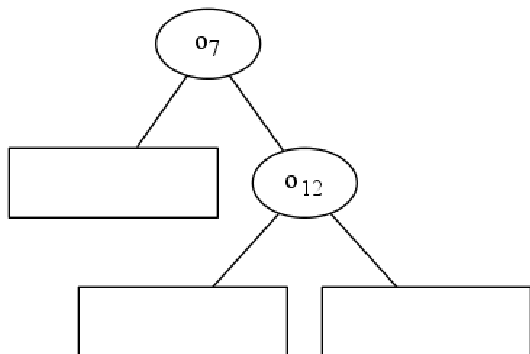
2. Here is a proposed greedy algorithm to choose the best binary tree (i.e., way to split up the process):

   Algorithm: If only a single computation remains, produce a single node for that computation. Otherwise, choose the least expensive operation o_i, make it the root of the tree, and produce its left and right subtrees by recursively applying the greedy algorithm to the parts of the process to the left and right of o_i. (For example, it would start by splitting the sample problem above into $(c_1 \circ_1 c_2 \circ_2 c_3)$ and $(c_4 \circ_4 c_5)$ because $\circ_3$ is the lowest-cost operation.)

   Give the smallest possible counterexample to the optimality of this algorithm. All operations $\circ_i$ in your counterexample must have distinct cost (so that "least expensive operation" is unique).

4.1 A Dynamic Programming Approach

1. Here is a portion of a tree representing a way to split a process with $n = 15$ computations up, with three boxes where the tree has not been finished.



   Fill in the boxes above with the computations still left to perform. Note: we do **not** want subtrees but computations like $(c_1 \circ_1 c_2 \circ_2 c_3 \circ_3 \ldots \circ_{n-2} c_{n-1} \circ_{n-1} c_n)$.

2. Give a succinct (short and clear) way to specify which computations go in each box.

This work is licensed under a Creative Commons Attribution 4.0 International License.
For license purposes, the author is the University of British Columbia.

3. Give a recurrence to describe the cost of the optimal way to split up the process. (Note that your algorithm for computing the overall cost of a tree also describes the cost of a particular split choice, while your succinct way to specify what goes in the boxes parameterizes the problem.)
4. Write a dynamic programming solution to the problem of finding the cost of the optimal way to split up a process.
5. Write a function that takes the table from your dynamic programming solution and produces the binary tree representing the split. (Each node in the tree should be labeled with the index of its operator—for internal nodes—or computation—for leaf nodes.)

This work is licensed under a Creative Commons Attribution 4.0 International License.
For license purposes, the author is the University of British Columbia.