CPSC 320 2019S2: Assignment 5

Please submit this assignment via GradeScope at `https://gradescope.com`. Be sure to identify everyone in your group if you're making a group submission (which we encourage!).

Submit by the deadline **Tuesday August 6 at 11PM**. 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. 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 **ugrad.cs.ubc.ca e-mail addresses** of each member of your team. Please **do not** include names on the assignment. If you don't mind private information being stored outside of Canada and want an extra double-check on your identity, include your student number rather than your name.
- 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' ugrad.cs.ubc.ca IDs, but **not** their names. (Be sure to get those IDs when you collaborate!)

1 Night of the Undead Recurrences

Consider the following recurrence – which, incidentally, can be used to describe the probability that a living army of size i will defeat an undead army of size j.[1]

$$P(i,j) = \begin{cases} 0 & \text{if } i = 0 \\ 1 & \text{if } j = 0 \\ 0.5 \cdot P(i-1, j+1) + 0.5 \cdot P(i, j-1) & \text{for } i, j > 0. \end{cases}$$

Assume that $P(0,0)$ is undefined.

[1] Check out the problem on "Riddler Classic":
`https://fivethirtyeight.com/features/how-many-soldiers-do-you-need-to-beat-the-night-king/`

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

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

3. Write clear (pseudocode) nested loops to specify an order for solving the subproblems to convert this to a dynamic programming solution to compute $P(m, n)$. There is no need to write the initialization code for the function or the body that would actually solve the problem: we just want the order of indices for the loops.

Take a Memo!

The following recurrence has been resurrected from the ashes of a previous CPSC 320 offering. Spooky!

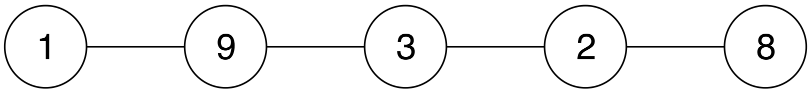
Assume you have an array of integers C with indexes $1 \ldots n$ and consider the following recurrence:

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

4. Give pseudocode for a memoized version of this algorithm. You may assume that you have a helper function `IsPow2(x)` that returns true if x is a power of 2. Your solution should take C (and n if desired) as a parameter.

2 The Path to Victory

Suppose we have an undirected graph G that is a *path*: namely, its nodes can be written as $v_1, v_2, \ldots v_n$ with an edge between v_i and v_j if and only if i and j are consecutive numbers. Each node v_i has a positive weight w_i. For example, in the following path, the weights are shown as numbers drawn inside the nodes:



In this problem, we want to find the *maximum independent set*. An independent set is a subset of nodes such that no two of them share and edge, and the maximum independent set is the independent set with largest total weight. For example, in the graph above, the largest-weight independent set is v_2 and v_5, with total weight 17.

Assume you're given an n-dimensional array V, where $V[i]$ contains the weight of the node v_i (note that we are assuming 1-based indexing).

1. Give a recurrence $M(i)$ (for $0 \leq i \leq n$) that defines the weight of the maximum independent set of the first i nodes in G.

2. Design a linear-time algorithm to compute the weight of the maximum independent set by converting your recurrence relation above into a dynamic programming solution.

3. Write a function that takes the table from your dynamic programming solution and the array V and returns the indices (in 1-based indexing) of the actual nodes in the maximum independent set (i.e., write an "explain" function like we've done in class).

4. The problem of finding the maximum independent set in a graph is known to be NP-complete. So why were we able to solve our problem in linear time? (Hint: it's not because P = NP.)

3 Menu-jay (Sequel to The Hungry Games)

Recall our favourite affordable buffet restaurant in Assignment 3, which sold foods in slurry form, from which you could dispense any fractional amount in grams you wanted into compostable paper cups. In that assignment, we showed that an optimal greedy strategy for getting the most calories from a given amount of spending money was to take as much as we could of the highest calorie-per-dollar menu items.

In a bid to attract a more sophisticated clientele, the restaurant has decided to switch to an à la carte system: that is, instead of instead of deciding how much weight of each item you can pick, you have to place an order for each dish off the menu. Assume that the mass (and number of calories) per dollar of each dish has remained the same. For example, if the restaurant used to have sweet-and-sour pork slurry for \$2 per 100 grams, and you came with \$11, you could spend \$11 on 550 grams of slurry. But now, they only sell a 400 gram dish of sweet-and-sour pork for \$8, which means you could spend \$8 on one order of pork, and have \$3 left for other purchases.

You know the cost of each dish on the menu and the number of calories the dish contains. Moreover, the restaurant has a limited quantity of each dish, and has said you can't place more than a single order for any dish.

As before, you come to a restaurant with some amount of money, and your goal is to maximize the number of calories you consume during your visit.

1. Construct a small instance with **no more than two different dishes** proving that the greedy approach we derived earlier is no longer optimal.
2. Give a recurrence to describe the maximum number of calories you can consume, given an amount of money M you have to spend (assume this is an integer), the price p_i of dish i (assume all prices are also integers), and the calories c_i of dish i, for $1 \leq i \leq n$.
3. Write pseudocode for a dynamic programming or memoized solution to compute the maximum number of consumable calories.
4. Give and briefly justify a good asymptotic bound for the runtime and memory usage of your algorithm.
5. **[BONUS, WORTH 1 COURSE BONUS POINT]** Suppose that the restaurant removed the restriction that you could place no more than one order per dish and said you could order a dish as many times as you wanted. Adapt your algorithm from question 3 to solve this problem, and give a good asymptotic bound for the runtime and memory usage of the new algorithm.

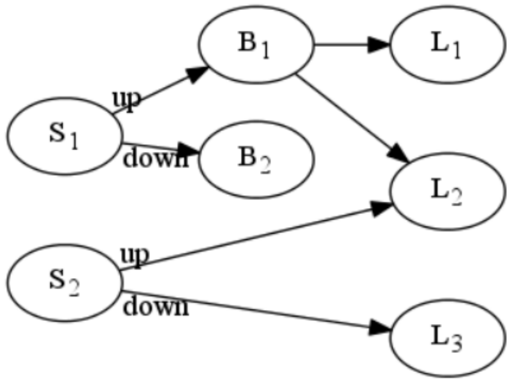
4 Transformers

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.
- "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. Prove that ELEC is NP-hard by giving 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. Try to think of features of the ELEC problem that are similar in some way.

Now, you will show that ELEC is in NP. This, together with the fact that it is in NP-hard, means that ELEC is NP-complete.

3. Give a (polynomial-length) certificate for ELEC instances where all loads can be powered. **Hint**: recall that a certificate is the *form* of a solution that we're actually *looking* for in a YES answer. What's the form of a solution to ELEC?

4. Give an algorithm that takes polynomial time in the size of an ELEC instance to check whether a certificate (of the kind that you describe above) is actually a solution that will power all loads in that instance.