CPSC 320 2017W2: Assignment 2

January 27, 2018

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. (Reminder: groups can include a maximum of three students; we strongly encourage groups of two.)

Remember that this assignment is based on the collected quizzes and quiz solutions. You will likely want to refer to those where you need more details on assignment questions.

Submit by the deadline **Monday 5 Feb at 10PM**. For credit, your group must make a **single** submission via one group member's account, marking all other group members and the pages associated with each problem 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 quizzes). 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 your name on the assignment, however.[1])
- 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 Wall-E Loman

... the air doesn't smell so foul here. If in doubt ... always follow your nose. — Gandalf

The *Traveling Salesperson Optimization Problem* (TSP) can be defined as follows: you are given a set $\{C_1, C_2, \ldots, C_n\}$ of cities. For every two cities C_i, C_j, there is a cost $c_{i,j}$ for traveling from city C_i to city C_j (equal to the cost of travelling from C_j to C_i). Your job is to find the lowest cost path that starts at C_1, travels through every other city, and returns to C_1. You are not allowed to go through a city more than once (except for C_1 which appears both at the start and at the end of your trip).

[1] If you don't mind private information being stored outside Canada and want an extra double-check on your identity, include your student number rather than your name.

1. Briefly but clearly justify why the number of valid solutions to an instance of TSP with n cities is $(n-1)!$. **Note:** a "brief justification" is not a proof but should contain the key ideas that would enable a correct proof.

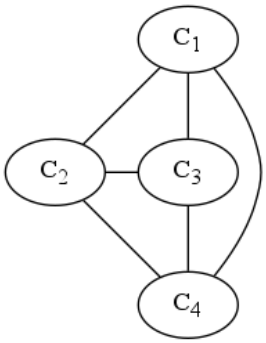
2. Imagine that you have a brute force algorithm that builds up a partial path step-by-step as a candidate solution to TSP. Each time it has visited all other nodes, it returns to C_1 and measures the cost of the solution.

   Complete the following small instance of TSP by labelling each edge with a *distinct* cost so that it can act as a counterexample to the correctness of the following heuristic for the algorithm: "always add as the next city in the partial path the cheapest not-yet-visited city to reach".

   Briefly describe how the algorithm could run and produce an incorrect solution.

   (E.g., "the algorithm starts at C_1 then the heuristic guides it to select C_2 then C_3 then C_4 as the "next closest" city at each step, but this produces a total of XXX, whereas the following solution produces a lower total . . . ".)

   Instance:



3. The correctness of the following heuristic relies on a (very sensible) assumption about the costs of travelling from one city to the next. What is the assumption, and why is it necessary?

   Heuristic: "discard any partial path that is more expensive than the cheapest complete path found so far".

4. Here is another heuristic for the algorithm for instances with at least 3 cities: "never include city C_3 in a partial path until after city C_2 has been included".

   Sketch the key points in a proof that this heuristic **is** correct. I.e., the algorithm run with this modification **will** produce an optimal solution.

2 The Antepenultimate Jedi

You want to hire for a movie the best stars to draw the most people to the theaters. For each star, you have a positive number indicating their "draw".

You can hire as many stars as you like. Certain actors consider themselves "big fish". If you include a big fish with draw x, you may not include any other star with draw k such that $\frac{x}{2} < k \leq x$.

1. For this question only, **every actor** considers themselves a "big fish". Give and briefly justify the correctness of an efficient greedy algorithm to solve this problem. (Your algorithm is likely to be simple enough that a clear English explanation is best, but clear and short pseudocode is also acceptable.)

2. We now alter the problem for this and subsequent parts. In the new version, the actor with the highest draw overall is automatically a big fish, whether or not they're in your movie. Additionally,

any *other* actor with draw x is a big fish if there is no actor at all (whether or not they're in your movie) with draw y such that $x < y < 2x$.[2]

So, for example, if the available stars have draw $[8, 10, 22, 3, 2, 30, 4]$ then 30 (as the highest draw actor) is a "big fish". 22 is not (because 30 is between 22 and 44). 10 is a "big fish" (because no other actor has draw between 10 and 20). 8 is not. 4 is a big fish. 3 and 2 are not.

Give an instance that includes actors with draw 11 and 13 in which the optimal solution involves hiring both of these actors.

3. Give an instance that includes actors with draw 11 and 13 in which no valid solution (whether optimal or not) involves hiring both actors.

4. Give and briefly justify the correctness of an efficient, greedy algorithm to solve this problem. (Your algorithm may be simple enough that a clear English explanation is best, but clear and short pseudocode may work better.)

5. Below we propose (suboptimal) algorithms to solve this problem. Give a **single** instance with three actors including one with draw 10 that serves as a counterexample to **all** the algorithms' optimality. That is, each algorithm will either fail to produce any valid solution or produce a suboptimal solution.

   Be sure to state: **the list of draws in your instance**, **the optimal solution and its total draw** and for each algorithm **the solution it produces and its total draw**.

   (a) Iterate through the stars in the order given. For each star, hire them if doing so would not invalidate the so-far-hired set of stars.

   (b) Sort the stars in decreasing order of draw. Iterate through the sorted stars. For each star, hire them if doing so would not invalidate the so-far-hired set of stars.

   (c) Sort the stars in decreasing order of draw. Iterate through the sorted stars. Hire the first (biggest) star. Then, for each subsequent star, hire them **unless** they are a big fish (in which case, skip them).

3 Blue, Gold, and Green

For its Blue-and-Gold fundraising campaign, UBC has found *anchors*—major donors, each with a *limit* (maximum amount they will donate)—and *causes* to which the anchors will donate.[3] Each cause has a *goal*, the maximum money that will go to the cause. **An anchor donates to at most one cause**, but **one cause may receive donations from many anchors**, and the total amount of the donations of anchors to a cause cannot exceed that cause's goal. The Blue-and-Gold Problem, or BGP, consists in determining how much money each anchor will give to each cause, subject to the constraints stated above. A BGP instance's solution is the maximum dollar amount that can be raised. (Assume limits are distinct, as are goals.)

1. An instance of BGP can be represented clearly and cleanly as a weighted, undirected, bipartite graph (where each edge has a **single** number as its weight). Draw the instance with limits $10, 100, 30, 40$ for anchors a_1, a_2, a_3, a_4, and goals $50, 80, 10$ for causes c_1, c_2, c_3.

2. Below we propose (suboptimal) algorithms to solve this problem. Give a **single** instance with three limits $[100, 50, 200]$ and two goals of your choice that serves as a counterexample to **all** the algorithms' optimality. That is, each algorithm will either fail to produce any valid solution or produce a suboptimal solution.

[2] They're a big fish in a little pond.

[3] And which are then named for them, like the Belleville Urban Beautification Project or Wolfman Lunar Research Fund.

Be sure to state: **the list of goals for your instance, the optimal solution and the anchor/cause pairings that generate it** and for each algorithm **the solution it produces and the anchor/cause pairings that generate it**.

Notes: (1) We use "limit" and "anchor" interchangeably. (2) All of these algorithms do the "right" thing when matching an anchor to a goal: They let the anchor donate the minimum of their limit and the money remaining before the goal is met and then update the money remaining for that goal.

(a) Iterate through the anchors in the order given. For each anchor, assign them to the cause with the most money still left before reaching its goal.

(b) Sort the goals in decreasing order. Iterate through the goals, matching each with the largest anchor who has not already been assigned a goal.

(c) Sort each of the limits and the goals in decreasing order. Iterate through the anchors, matching each with the first remaining goal. (Remove a goal when its fundraising target has been met.)

3. **PROBLEM CUT FROM ASSIGNMENT**

4 Oh Oh Oh

1. Determine the worst case running time of the `colorize` algorithm as a function of both n and m:

```
// G = (V, E) is an undirected graph, represented as an adjacency list
function colorize(V, E)
    n = |V|
    m = |E|
    let Colors be a list of at least n distinct colors
    for i = 0 to n - 1:
        V[i].setvisited(False)

    for i = 0 to n - 1:
        DFS(V[i], (function(v): v.setcolor(Colors[i])))
```

where DFS is defined as:

```
function DFS(v, visitfunction)
    if not v.getvisited():
        visitfunction(v)
        v.setvisited(True)
        for w in neighbours(v):
            DFS(w, visitfunction)
```

2. Briefly justify how your bound applies to a complete graph, including how and why each piece of the code contributes to the bound.

3. Describe in English what this algorithm does. Your description should be brief, clear, and precise.

5 O'd to a Pair of Runtimes

For this problem, we consider a dense, undirected graph and a simple path in that graph of length k. (A *dense* graph has $m \in \Theta(n^2)$. A *simple path* of length k is a list of $k+1$ vertices where each subsequent pair of vertices is connected by an edge and no vertex appears more than once in the list.) **ASSUME $k > 0$.**

1. Give exact upper- and lower-bounds on k in terms of n. (Note that "expressing y in terms of x" means you *can but need not* use x in a function you provide to describe y.)
2. Give an exact upper-bound on m in terms of n.
3. Briefly justify why it is not possible to give an exact lower-bound on m in terms of n using the information above.
4. Give a good Θ-bound on each of the following functions from above. Express your bound in terms of as few as possible of the variables k, n, and m.
   (a) $n \lg(knm)$
   (b) $(k+m)(n+k)$
   (c) $k^3 + m \lg m$

6 eXtreme True And/Or False

Each of the following problems presents a scenario in a graph and a statement about that scenario. For each one, exactly one of the following is correct:

1. The statement is **ALWAYS** true, i.e., true in *every* graph matching the scenario.
2. The statement is **SOMETIMES** true, i.e., true in some graph matching the scenario but also false in some such instance.
3. The statement is **NEVER** true, i.e., true in *none* of the graphs matching the scenario.

Briefly justify the correct answer to each one.

- Justify an **ALWAYS** answer by giving a small instance that fits the scenario for which the statement is true and then briefly sketching the key points in a proof that the statement is true for all instances that fit the scenario.
- Justify a **NEVER** answer by giving a small instance that fits the scenario for which the statement is **false** and then briefly sketching the key points in a proof that the statement is **false** for all instances that fit the scenario.
- Justify a **SOMETIMES** answer by giving **two** small instances that fit the scenario: one for which the statement is true and one for which the statement is false. (Indicate which is which!)

Recall: $|V| = n, |E| = m$. A vertex's *degree* is the number of edges incident on it. For directed graphs, a vertex's *in-degree* is the number of edges ending at the vertex and the *out-degree* is the number starting from it. A *path* of length k is a list of $k+1$ vertices with an edge between each consecutive pair of vertices. A *simple path* repeats no vertex. A *cycle* is a path that starts and ends at the same vertex. A *simple cycle* repeats no vertex other than the starting/end vertex (which only appears twice). Unless stated otherwise, we assume graphs have no self-loops (edges from a vertex to itself).

1. **Scenario:** An undirected graph with $n \geq 2$ and at least $\frac{n^2}{4}$ edges. **Statement:** The graph is connected.
2. **Scenario:** A tree (undirected) found by DFS run on a connected, undirected graph with $n \geq 2$. **Statement:** The degree of every node in the tree is two or less.
3. **Scenario:** A weakly-connected but **not** strongly-connected, directed graph with $n \geq 2$. **Statement:** A simple cycle of length $n+1$ exists.
4. **Scenario:** A connected, undirected graph with $n \geq 3$. **Statement:** The graph includes at least 2^n different simple cycles.

7 BONUS!

This is worth only one course bonus point (for a clearly on-track answer that makes significant progress) or two course bonus points (for a beautiful, complete answer), which is way too little for how much work it is. On the other hand, how fun is it?! (On a correct group submission, each group member earns the number of bonus points noted above.)

Give and prove the correctness of an achievable, asymptotic upper-bound (in terms of n) on the number of optimal solutions to an interval scheduling problem instance with n intervals. *Hint:* it isn't an answer in online quiz #4!

Note that you'll need to give your bound, give a way to construct an instance of an arbitrary size n that has a number of solutions in your bound, and show that no higher bound is possible.