CPSC 320 2017W1: Assignment 2

September 27, 2017

WARNING: We reserve the right to add some questions to this assignment after the midterm exam.

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 **Friday 13 Oct 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 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. (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' ugrad.cs.ubc.ca e-mail addresses. (Be sure to get those when you collaborate!)

1 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, indicate by filling in the appropriate circle whether:

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.

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 other vertex.

1. **Scenario:** A directed graph with $n \geq 2$. **Statement:** The out-degree of each vertex is larger than its in-degree.
   ◯ ALWAYS
   ◯ SOMETIMES
   ◯ NEVER

2. **Scenario:** A non-empty DAG with a single, unique topological ordering. **Statement:** Any two different vertices in the graph have exactly one edge between them (i.e., an edge going one direction or the other, but not both edges).
   ◯ ALWAYS
   ◯ SOMETIMES
   ◯ NEVER

3. **Scenario:** A strongly-connected, directed graph with $n \geq 2$. **Statement:** A simple cycle of length n exists.
   ◯ ALWAYS
   ◯ SOMETIMES
   ◯ NEVER

4. **Scenario:** A connected, undirected graph with $n \geq 2$. **Statement:** A path—not necessarily simple—of length $2n$ exists that visits every node.
   ◯ ALWAYS
   ◯ SOMETIMES
   ◯ NEVER

1.1 Quiz Solution

1. **SOLUTION:** NEVER
2. **SOLUTION:** SOMETIMES
3. **SOLUTION:** SOMETIMES
4. **SOLUTION:** ALWAYS

1.2 Assignment

On the assignment, for each of the problems above, please **briefly** justify the answer.

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.

2 O'd to a Pair of Runtimes

The pairs of functions below represent algorithm runtimes on an **undirected, connected** graph with n vertices and m edges. **ASSUME** $m > 0$**.** For each pair, fill in the circle next to the best choice of:

LEFT: the left function is big-O of the right, i.e., left $\in$ O(right)

RIGHT: the right function is big-O of the left, i.e., right $\in$ O(left)

SAME: the two functions are Θ of each other, i.e., left $\in \Theta$(right)

INCOMPARABLE: none of the previous relationships holds for all allowed values of n and m.

Do not choose **LEFT** or **RIGHT** if **SAME** is true. The first one is filled in for you.

Left Function	Right Function	Answer
n	n^2	**LEFT**
$n+3$	m	◯ LEFT ◯ RIGHT ◯ SAME ◯ INCOMPARABLE
$nlgn$	m	◯ LEFT ◯ RIGHT ◯ SAME ◯ INCOMPARABLE
n^2	$3m + \lg n$	◯ LEFT ◯ RIGHT ◯ SAME ◯ INCOMPARABLE
$m^2 \lg n$	2^n	◯ LEFT ◯ RIGHT ◯ SAME ◯ INCOMPARABLE
$2^{\log_{10} m}$	2^n	◯ LEFT ◯ RIGHT ◯ SAME ◯ INCOMPARABLE
$\frac{m}{\lg n}$	1	◯ LEFT ◯ RIGHT ◯ SAME ◯ INCOMPARABLE

2.1 Quiz Solution

Left Function	Right Function	Answer
n	n^2	**LEFT**
$n+3$	m	**LEFT**
$nlgn$	m	**INCOMPARABLE**
n^2	$3m + \lg n$	**RIGHT**
$m^2 \lg n$	2^n	**LEFT**
$2^{\log_{10} m}$	2^n	**LEFT**
$\frac{m}{\lg n}$	1	**RIGHT**

2.2 Assignment

On the assignment:

- For your own reference (not graded, but **crucial**), indicate the maximum and minimum values of m in terms of n for the scenario described in the problem.
- Give a good Θ-bound on each of the following functions from above:
  1. $3m + \lg n$
  2. $m^2 \lg n$
  3. $2^{\log_{10} m}$
  4. $\frac{m}{\lg n}$

  Notes:
  - Where possible, use logs base 2 (i.e., lg).
  - If a decimal number like 4.48 is important, round to the tenths place, like 4.5. (Technically, that will make your Θ bound incorrect, but we won't worry about that for this problem.)
  - If possible, express your bound in terms of a single variable m or n rather than both m and n.
- For the one problem whose answer is **INCOMPARABLE**, **briefly** justify why neither function is big-O of the other.

3 Slapping on a Bandwidth-Aid

Recall: EDP's input is an undirected, unweighted graph $G = (V, E)$ plus a set of distribution points $D = \{d_1, d_2, \ldots, d_k\}$ each a vertex in V and a single aid location $a \in V$ that is not in D. The output is the number of non-overlapping paths leading from some d_i to a. Paths may lead from different distribution points.

You have an efficient algorithm to solve the "network bandwidth problem" (NBP). NBP's input is a **weighted**, **directed** graph $G = (V, E)$ (where the tuple $(u, v, w) \in E$ represents a **directed** edge from u to v with **integer weight** w) and designated source and target vertices s and t. A node in the graph is a server and an edge is a network link between servers, weighted by its bandwidth—the maximum number of bytes per second the link can carry. (A weight of ∞ is also allowed, indicating unlimited bandwidth.)

NBP's output is the maximum bandwidth that can be carried from s to t.

Notes: The bandwidth on any link cannot exceed that link's weight. The bandwidth coming **out of** s is unlimited but none can go in, while unlimited bandwidth can go **into** t but none can come out. Otherwise, for any node v, the bandwidth coming into the node must equal the bandwidth coming out. **Assume only integral** (or infinite for links with weight ∞) **amounts of bandwidth can be used on each edge.**

Fill in the boxes to complete this reduction that solves EDP using NBP:

Convert an instance of EDP to an instance of NBP:

1. For each node $v \in V_{EDP}$, generate [ ] in V_{NBP}.
2. For each undirected edge $(u, v) \in E_{EDP}$, generate [ ] in E_{NBP}.
3. Generate one additional vertex v' in V_{NBP}.
4. For each distribution point $d \in D$, generate [ ].
5. Finally, let $s =$ [ ] and $t =$ [ ].

Convert an instance of NBP to EDP:

Let the solution to EDP be [ ].

3.1 Quiz Solution

First, two **clarifications**:

1. Paths being "non-overlapping" means they do not share **edges**. (They must share at least one vertex, the aid location to reach. They may also share other vertices, but they must use different edges between the vertices.)
2. A single distribution point can be the source of many disjoint paths to the aid location.

If you made different assumptions that are **reasonable in the given context of the problem** and led you to an alternative, correct solution, we intend to give you full credit for your quiz solution. (As always, you can object to our grading *once we return it* if you have concerns. Please don't object before you see your grade! ☺)

Now, **on to our solution**, which respects these clarifications.

Convert an instance of EDP to an instance of NBP:

1. For each node $v \in V_{EDP}$, generate a corresponding node v in V_{NBP}.
2. For each undirected edge $(u, v) \in E_{EDP}$, generate two edges: $(v, u, 1), (u, v, 1)$ in E_{NBP}.
3. Generate one additional vertex v' in V_{NBP}.
4. For each distribution point $d \in D$, generate an edge $(v', d, \infty) \in E_{NBP}$.

5. Finally, let $s = v'$ and $t = a$.

Convert an instance of NBP to EDP:

Let the solution to EDP be the solution to NBP. (Where a solution of ∞ means that the target is also a distribution point, in which case, it's not clear what a correct solution is anyway.)

3.2 Assignment

(Be sure to read clarifications in the quiz section above.) Now, consider this **incorrect** reduction from EDP to NBP.

Convert an instance of EDP to an instance of NBP:

1. For each node $v \in V_{EDP}$, generate a corresponding node v in V_{NBP}.
2. For each undirected edge $(u, v) \in E_{EDP}$, generate two edges: $(v, u, 1), (u, v, 1)$ in E_{NBP}.
3. Generate one additional vertex v' in V_{NBP}.
4. For each distribution point $d \in D$, generate an edge $(v', d, 1) \in E_{NBP}$.
5. Finally, let $s = v'$ and $t = a$.

Convert an instance of NBP to EDP:

Let the solution to EDP be the solution to NBP.

Now, do the following:

1. Give a good **counterexample** to the correctness of this reduction. (A drawing of the counterexample instance is an excellent way to express it.)
2. Give the instance of NBP produced by the reduction from your counterexample. (Again, a drawing works well.)
3. Give and **very briefly** explain the incorrect solution to the original instance produced by the reduction (paired with an arbitrary solution to the underlying problem).
4. Give and **very briefly** explain the correct solution to the original instance.

4 Milking Your Territory

Suppose the leaders of a town have decided that they should hire some ice cream vendors to ride bicycles around town, selling their goods to the children of the community. Your task is to help the leaders decide how many vendors to hire. To help us solve this problem, we have the following definitions:

- The town consists of a set of homes and a set of sidewalks, on which the vendors will ride.
- Each sidewalk connects a pair of homes.
- Two homes h_1 and h_2 are in the same market if there is a way to get from h_1 to h_2 riding only on sidewalks whose lengths are less than 100m each. (Bicycle vendors don't like to ride very far before they take a break.)
- Each market will be served by exactly one vendor.

Now, answer the following questions:

1. The number of vendors can be found most effciently by a modification of which of the following algorithms? Fill in the boxes next to **ALL** correct answers. (Several could be the foundation for a solution; select only those that solve the problem the fastest and with the least unnecessary functionality.)

   ☐ HeapSort
   ☐ Dijkstra's
   ☐ Kruskal's Minimum Spanning Tree algorithm
   ☐ Prim's Minimum Spanning Tree algorithm
   ☐ BFS
   ☐ DFS

2. **Briefly** describe using pseudocode (or simple C++/Java- or Racket-style code) two changes you would make to the algorithm:

   - change 1:

   - change 2:

3. Suppose each house is connected to at most 5 other homes by sidewalks. What is the running time of your algorithm in terms of the number of houses H?

4.1 Quiz Solution

1. DFS and BFS
2. Here are the changes conceptually:
   (a) An (unweighted) edge exists between two houses exactly when there is an edge (sidewalk) between them in the original graph of weight less than 100 (distance less than 100m).
   (b) Initialize the value L to 1. Then, repeat until all nodes are labelled in the following way:
      i. Pick an arbitrary unlabeled vertex.
      ii. Run DFS/BFS from that vertex, labeling all reachable nodes with L.
      iii. Increment L.
3. $O(h)$ (assuming a good implementation)

4.2 Assignment

On the assignment, you'll solve a related but **different** problem.

The algorithm above determined the appropriate *number* of vendors. With small modifications, it can also label each home with an identifier that indicates the market to which it belongs. **Assume for this problem** that such a labelling already exists. Furthermore, **assume** that in the town from the original problem, there is **at least one path along sidewalks from each house to each other house** in the town, although some of the sidewalks on the path may be longer than 100m.

After hiring the appropriate number of vendors (i.e., one for each market), each of whom lives in some house in the town, the leaders have to decide which vendor will sell in which market. The goal is to minimize the total commuting distance of the vendors, where the commuting distance for a vendor to a market is defined to be: the length of the shortest route from the vendor's home to some home in a market. The town leaders want to know which vendor should serve each market. (Note that some vendors' home-to-market routes may include sidewalks longer than the 100m limit, but we assume that vendors are OK with that on their commute. They'll walk their bikes if needed.)

We call this the Vendor Assignment Problem (VAP).

The following two problems will be instrumental in designing an algorithm to solve VAP. You should assume that efficient solutions to each exist (and they do!).

All Pairs Shortest Path (APSP): takes as input a weighted graph $G = (V, E)$ and returns a $|V| \times |V|$ matrix whose entries are 1) $dist(u, v)$, the shortest distance from vertex u to vertex v, and 2) $pred(u, v)$, the second to last vertex on the shortest path from u to v.

Minimum Cost Bipartite Matching (MCBM): takes as input a bipartite graph $G = (V \cup W, E)$ with costs $c_e, \forall e \in E$, and returns a perfect matching of least total cost if such a matching exists. Note that sets V and W are disjoint, costs are non-negative, and $|V| = |W| = n$.

Now solve the following problems:

1. Briefly explain why any scenario in which all vendors live in different markets might be considered a "trivial instance".

2. Make a sketch that you believe illustrates a **small but not trivial** instance of the problem, and augment the sketch with an optimal solution.

3. Give symbolic names to the quantities that matter in VAP, and use those names to describe **both** the **general form of an instance** (including any constraints on what makes it valid) of the problem, and the **general form of a solution** (including what makes a solution valid and good).

   If you're looking for a format, consider the various problems (including APSP and MCBM) that we've represented using symbolic names in this quiz/assignment (plus your textbook reading and our work in class, of course!).

4. Now use clear pseudocode (or **very very clear** C++, Java, or Racket) to give an algorithm to solve VAP given an instance of the original ice cream vendor problem that has been augmented with labels for each house indicating its market.

   You **must** make use of the APSP and MCBM algorithms in your solution, And they should make your solution fairly compact and simple. (You can think of this as reducing VAP to a combination of APSP and MCBM.)

For further reading on APSP and MCBM, see `http://jeffe.cs.illinois.edu/teaching/algorithms/notes/22-apsp.pdf`, and `http://theory.stanford.edu/~tim/w16/l/l5.pdf`.