CPSC 320 2017W1: Midterm 1

January 26, 2018

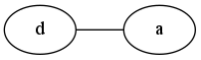
The next page are a pair of problem descriptions we will use on the exam.

Problem reminders:

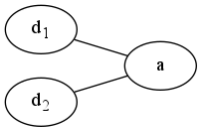
EMERGENCY DISTRIBUTION PROBLEM (EDP)

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 (edge-wise distinct) paths leading from some d_i to a. (Multiple paths may lead from a single distribution point, and paths may lead from different distribution points.)

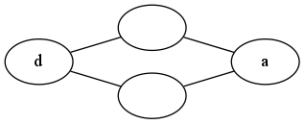
Here are some small sample instances with their solutions, where d, d_1, d_2 are distribution points and a is the aid vertex:



Solution: 1.



Solution: 2.



Solution: 2.

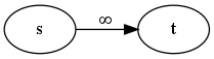
NETWORK BANDWIDTH PROBLEM (NBP)

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.)

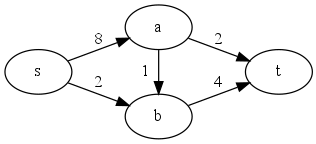
NBP's output is the maximum bandwidth that can be carried from s to t.

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.**

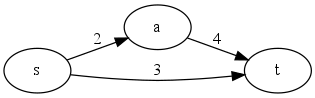
Here are some small sample instances with their solutions and a brief description of how to send the solution bandwidth from s to t. (Note: solving small instances by hand may be helpful, but you do **not** need to know or understand any algorithm to solve this problem.)



Solution: ∞.
∞ on $s \to t$



Solution: 5.
3 on $s \to a$
2 on $s \to b$
1 on $a \to b$
2 on $a \to t$
3 on $b \to t$



Solution: 5.
3 on $s \to t$
2 on $s \to a$
2 on $a \to t$

This page intentionally left (almost) blank.
If you write answers here, you must CLEARLY indicate on this page what question they belong with AND on the problem's page that you have answers here.

WRITE YOUR UGRAD ID: [ ] @ugrad.cs.ubc.ca (**-1 mark** if missing)

1 Differential Treatment [13 marks]

1. Consider the execution of the Gale-Shapley algorithm with **women proposing**, and imagine that it maintains a "marker" for every proposing person w, denoting the ranking of the person to whom she will next propose (if she makes another proposal). (I.e. her most preferred match is rank 1, second most preferred is rank 2, etc.)

   Fill in the circles next to the correct choices in order to complete the following narrative which justifies a bound on the worst-case running time of Gale-Shapley. **[5 marks]**

   Every iteration of Gale-Shapley moves one proposer's marker to a ◯ more / ◯ less preferred rank.

   Assuming constant time access to the preferences of all participants, each iteration (proposal and acceptance/rejection) requires time ◯ $O(1)$ / ◯ $O(\lg n)$ / ◯ $O(n)$ in the worst case.

   Since there are only ◯ $O(1)$ / ◯ $O(n)$ / ◯ $O(n^2)$ preferences total, for all proposers, ◯ $O(n)$ / ◯ $O(n \lg n)$ / ◯ $O(n^2)$ / ◯ $O(n^3)$ is a(n)

   ◯ upper / ◯ lower bound on the total running time of Gale-Shapley.

2. Consider this problem: Find and return a pair of any two **different** numbers in an array of n numbers. The array may contain duplicates but **does** contain at least two distinct values. **[3 marks]**

   Fill the circle next to the **best big-O bound** for the worst-case performance of an efficient algorithm to solve this problem if the array is...

   (a) ...unordered:
   - ◯ $O(1)$
   - ◯ $O(\lg n)$
   - ◯ $O(n)$
   - ◯ $O(n \lg n)$
   - ◯ $O(n^2)$

   (b) ...known to be sorted:
   - ◯ $O(1)$
   - ◯ $O(\lg n)$
   - ◯ $O(n)$
   - ◯ $O(n \lg n)$
   - ◯ $O(n^2)$

3. **Choose the data structure** for each problem below that most efficiently supports a solution. Choose the **best** answer. If there are multiple best answers, just pick one. **[3 marks]**

   (a) Given a string with $2n$ characters, determine if it is a palindrome (i.e., reads the same forward and backward).

   Chosen data structure: **[1 mark]**
   - ◯ stack
   - ◯ queue
   - ◯ priority queue (implemented as a binary heap)
   - ◯ map/dictionary (implemented as a hash table)

   (b) Given an odd query integer q, determine if a sequence of n integers contains two integers that sum to q

   Chosen data structure: **[2 marks]**
   - ◯ stack
   - ◯ queue
   - ◯ priority queue (implemented as a binary heap)
   - ◯ map/dictionary (implemented as a hash table)

4. DFS and BFS can produce different trees on the same (undirected, connected) graph depending on which node they are run on and what order children are visited. Which of these stays the same for a given graph regardless of these choices? Fill in the boxes next to **all** that apply: **[2 marks]**
   - ☐ The height of a DFS tree
   - ☐ The height of a BFS tree
   - ☐ The number of dashed edges in a DFS tree
   - ☐ The number of dashed edges in a BFS tree

2 eXtreme True And/Or False [15 marks]

Each of the following problems presents a scenario and a statement about that scenario. For each one, fill the circle by the best of these choices:

- The statement is **ALWAYS** true, i.e., true in *every* instance matching the scenario.
- The statement is **SOMETIMES** true, i.e., true in some instance matching the scenario but also false in some such instance.
- The statement is **NEVER** true, i.e., true in *none* of the instances matching the scenario.

Then, **justify** your answer as follows:

ALWAYS answer: give a small instance that fits the scenario for which the statement is true and briefly sketch the key point(s) in a proof that the statement is true for all instances that fit the scenario.

SOMETIMES answer: give a small instance that fits the scenario for which the statement is true and a small instance that fits the scenario for which the statement is false.

NEVER answer: give a small instance that fits the scenario for which the statement is false and briefly sketch the key point(s) in a proof that the statement is false for all instances that fit the scenario.

Here are the problems:

1. **Scenario:** Any SMP instance with $n \geq 2$ in which two men share the same preference list. **Statement:** The Gale-Shapley algorithm run with men proposing terminates after exactly n iterations. **[5 marks]**

   ◯ **ALWAYS**
   ◯ **SOMETIMES**
   ◯ **NEVER**

   True instance (always/sometimes) *or* **proof that statement is false in all instances** (never):

   False instance (sometimes/never) *or* **proof that statement is true in all instances** (always):

2. **Scenario:** A simple, connected, undirected, unweighted graph with $n \geq 2$. **Statement:** The minimum distance among all longest paths between pairs of vertices in the graph is equal to the maximum distance among all shortest paths between pairs of vertices in the graph.

   ◯ **ALWAYS**
   ◯ **SOMETIMES**
   ◯ **NEVER**

   True instance (always/sometimes) *or* **proof that statement is false in all instances** (never):

   False instance (sometimes/never) *or* **proof that statement is true in all instances** (always):

3. **Scenario:** A simple, undirected graph (with no self-loops) with $n \geq 2$ and $m \geq \frac{n^2}{5}$. **Statement:** The graph is connected. **[5 marks]**

   ◯ **ALWAYS**
   ◯ **SOMETIMES**
   ◯ **NEVER**

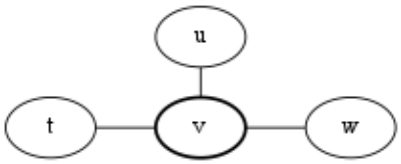
   True instance (always/sometimes) *or* **proof that statement is false in all instances** (never):

   False instance (sometimes/never) *or* **proof that statement is true in all instances** (always):

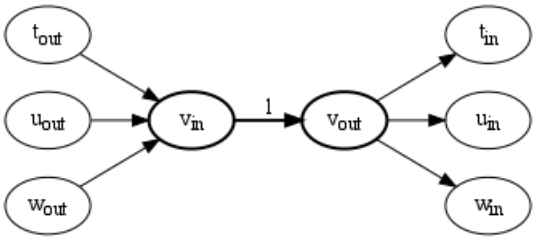
3 Slapping on a Bandwidth-Aid [12 marks]

Consider a variant of EDP that we call VDP (vertex-disjoint path problem). The input is the same, but the output is the maximum number of distinct, **vertex-disjoint** paths from distribution vertices to the aid vertex. Specifically, no two paths from distribution vertices to aid vertex can share any vertex **except** their end point (the aid vertex) and potentially their start point (if they begin at the same distribution point but travel different paths to the aid vertex).

We now describe a solution to VDP via reduction to NBP. The core of the reduction is to limit travel "through" a vertex by transforming a vertex like v in VDP (shown with only its immediate neighbors):



into a pair v_{in} and v_{out} in NBP:



Here is our (on track but broken) reduction:

Convert an instance of VDP to an instance of NBP:

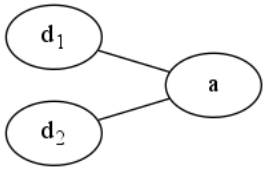
1. Generate new vertices in V_{NBP} as follows:
   (a) For the aid vertex a, generate a vertex $a_{in} \in V_{NBP}$.
   (b) For each distribution vertex $d \in D$, generate a vertex $d_{out} \in V_{NBP}$.
   (c) For each other vertex $v \in V_{VDP}$, generate two vertices $v_{in}, v_{out} \in V_{NBP}$ **and** an edge $(v_{in}, v_{out}, 1) \in E_{NBP}$.
2. For each undirected edge $(u, v) \in E_{VDP}$, if possible generate two edges: $(u_{out}, v_{in}, \infty)$ and $(v_{out}, u_{in}, \infty)$ in E_{NBP}, skipping cases where a corresponding vertex does not exist.
   (E.g., an edge (v, a) with the aid vertex can produce only an edge from v_{out} to a_{in}, since a_{out} does not exist.)
3. Generate one additional vertex v' in V_{NBP}.
4. For each distribution point $d \in D$, generate an edge $(v', d_{out}, \infty) \in E_{NBP}$.
5. Finally, let $s = v'$ and $t = a_{in}$.

Convert a solution to NBP to VDP:

Let the solution to VDP be the solution to NBP.

On the next pages, you consider and comment on small VDP instances and this reduction. In each instance, the vertices labelled d_1 or d_2 are **distribution points**, the vertex labelled a is the **aid vertex**, and others are "regular" vertices.

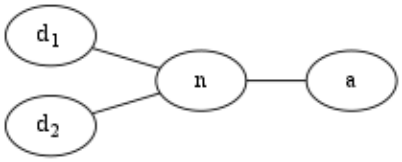
1. Consider this VDP instance, for which the correct solution is 2:



   (a) Draw the NBP instance created by the reduction from this VDP instance. **[3 marks]**

   (b) Give the solution to this instance produced by the reduction: ☐. **[1 mark]**

2. Consider this VDP instance, for which the correct solution is 1:



   (a) Draw the NBP instance created by the reduction from this VDP instance. **[4 marks]**

   (b) Give the solution to this instance produced by the reduction: ☐. **[1 mark]**

3. The instances above highlight a problem with the reduction. Fill in the blanks below to describe the **very small** change needed to fix the reduction on the previous page. **[3 marks]**

   Change the ☐ in step ☐ to ☐.