CPSC 320 2023S2: Quiz 1 Solutions

2 Stable Matching (4 points)

1. Are there instances of the Stable Matching Problem (SMP) that have more than one stable solution?

   SOLUTION: ● Yes ○ No

2. In an SMP instance with n employers, suppose that every employer has a *different* most-preferred applicant. For such an instance, which of these is a valid big-O bound on the number of offers made by the Gale-Shapley algorithm? **Check all that apply.**

   SOLUTION:

   ○ $O(1)$ ● $O(n \log n)$
   ○ $O(\log n)$ ● $O(n^2)$
   ● $O(n)$ ● $O(n^3)$

3. Let $T(n)$ be the worst-case runtime of SMP-Brute-Force, shown below (as in the worksheet). Recall that $T(n) = \Theta(n!\, n^2)$.

   function SMP-BRUTE-FORCE(n, P_E, P_A)
   ▷ $n \geq 1$ is the number of employers and also the number of applicants
   ▷ P_E is the collection of complete preference lists ($>_e$) of the employers
   ▷ P_A is the collection of complete preference lists ($>_a$) of the applicants

   for each perfect matching M (i.e., potential solution) **do**
   if M is stable (i.e., a good solution) **then**
   return M
   return "no stable matching"

   Is it the case that $T(n) = \Theta((n+2)!)$?

   SOLUTION: ● Yes ○ No

3 Data Structures for Linear Time Sorts (2 points)

Suppose that the entries stored in an unsorted array A of length n are all in the range $[1, \ldots, n]$. Array A may have duplicate entries, i.e., some entries may have the same value. Briefly describe a *linear time* algorithm that sorts the array A. You can provide an English description or a few lines of pseudocode. It's fine to use a second array if it's helpful. Hint: If you knew how many times each number in the range $[1, \ldots, n]$ appears in the array, could you then generate the array in sorted order?

SOLUTION: Create an array $B[1..n]$, and initialize all of its entries to 0. Then, scan the entries of array A from left to right. When an entry with value i is scanned, add 1 to $B[i]$. Once the scan of A is complete, scan B from left to right, and replace entries of A as follows: when the ith entry of B is scanned, if it has value j, then write i into the next j cells of A.

4 2-Means Custering (4 points)

Partitioning a set S of n data points $x_1, x_2, \ldots, x_n$ into a small number of related groups is a central problem in data science. One approach, called k-means clustering, aims to partition the points into k groups so that the clusters are as compact as possible, in a sense explained below. In this problem you'll analyze the runtime of a brute force implementation of 2-means clustering. Throughout, assume that each data point is a d-dimensional vector $x_i = (x_{i1}, x_{i2}, \ldots, x_{id})$.

1. The *mean* $\mu(S)$ of a set S of n d-dimensional points is the d-dimensional vector

$$\mu(S) = (\mu(S)_1, \mu(S)_2, \ldots, \mu(S)_d) = (\sum_{x_i \in S} x_{i1}/n, \sum_{x_i \in S} x_{i2}/n, \ldots, \sum_{x_i \in S} x_{id}/n).$$

   For example, the mean of the set $S = \{(1,3,2), (5,5,4)\}$, which has $n = 2$ points of dimension $d = 3$, is

$$((1+5)/2, (3+5)/2, (2+4/2)) = (3,4,3).$$

   How many arithmetic options (addition, subtraction, multiplication or division) operations are needed to calculate $\mu(S)$? Choose one option.

   ○ $\Theta(d \log n)$ ○ $\Theta(n \log d)$ ○ $\Theta(n + d)$ ● $\Theta(nd)$

2. The *spread* $\text{spr}(S)$ is the sum of the squares of the distances between the points and the mean $\mu(S)$:

$$\text{spr}(S) = \sum_{x_i \in S} \sum_{j=1}^{d} (x_{ij} - \mu(S)_j)^2.$$

   How many arithmetic options operations are needed to calculate the spread? Choose one option.

   ○ $\Theta(d \log n)$ ○ $\Theta(n \log d)$ ○ $\Theta(n + d)$ ● $\Theta(nd)$

3. A k-means clustering algorithm aims to partition a set of points into k clusters, so as to minimize the spread. Specifically, when $k = 2$, the points are partitioned into two (non-empty) clusters, say C_1 and C_2, so as to minimize

$$\text{spr}(C_1) + \text{spr}(C_2).$$

   A brute force algorithm for 2-means clustering considers each possible partition (C_1, C_2) of the points, and chooses the partition with the minimum spread. How many partitions must be considered by the algorithm?

   ○ $\Theta(2^d)$ ● $\Theta(2^n)$ ○ $\Theta(2^{n+d})$ ○ $\Theta(2^{nd})$

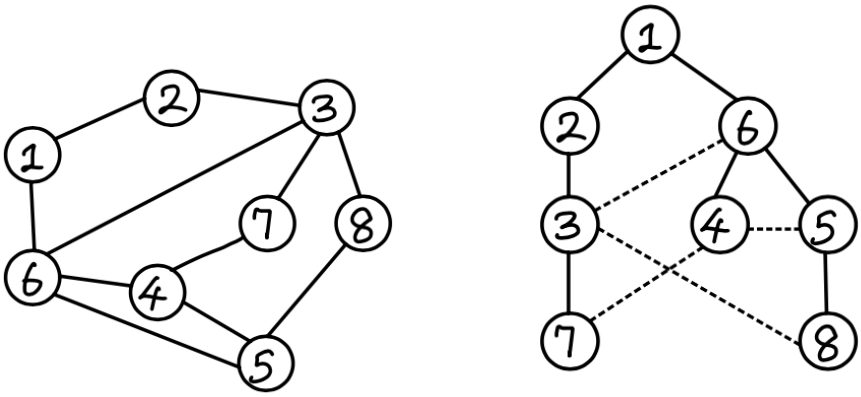
4. Let $f(n,d), g(n,d)$, and $h(n,d)$ be your choices for parts 1, 2, and 3 of this problem respectively. What is the overall runtime of the brute force algorithm?

   ○ $\Theta(f(n) + g(n) + h(n))$ ● $\Theta((f(n) + g(n))h(n))$
   ○ $\Theta(f(n)(g(n) + h(n)))$ ○ $\Theta(f(n)g(n)h(n))$

5 Counting Shortest Paths (5 points)

Let $G = (V, E)$ be an undirected, unweighted, connected graph with n nodes and m edges. For any pair of distinct nodes x and v, let $c(x, v)$ be the total number of shortest paths from x to v.

Example: The following graph has one shortest path from 1 to 6. Also there are three shortest paths from 1 to 7. Two of these, namely path 1,2,3,7, and path 1,6,3,7, go through node 3, while one, namely path 1,6,4,7, goes through node 4.



1. Draw a breadth first search tree rooted at 1 for the graph above. You can draw it to the right of the graph. Include all dashed edges as well as tree edges.

2. How many shortest paths are there from node 1 to node 8? That is, what is the value of $c(1, 8)$?

SOLUTION:

○ 1 ● 2 ○ 3 ○ 4

3. Choose which code to add in the space indicated below, to obtain an algorithm that computes $c(x, v)$ for all pairs of distinct nodes. For each node x, this code first initializes $c(x, v)$ for all v, and then calls a version of breadth first search that your selection will modify.

SOLUTION:

The solution here is based on an inductive definition for $c(x, v)$. If $x = v$ then there is exactly one shortest path from x to x, so $c(x, x)$ is initialized to 1. Otherwise, if $d_s[v]$ is d, we count the number of shortest paths to nodes u at level $d - 1$, such that we can then get from u to v via edge (u, v).

That is,

$$c(x, v) = \sum_{\substack{u \mid (u,v) \in E, \\ d_s[v] = d_s[u] + 1}} c(x, u)$$

The sum on the right is taken over all nodes u that are neighbours of v and additionally where v is one level deeper in the tree than u. In the code, term $c(x, u)$ of this sum is added to $c(x, v)$ when edge (u, v) is examined during the call to MODIFIED-BFS(x).

```
procedure COUNT-SHORTEST-PATHS(G)
    ▷ compute c(x,v), the number of shortest paths, from x to v, for all pairs of notes (x,v)
    for all x ∈ V do
        initialize c(x,x) to 1
        for all v ∈ V, v ≠ x do
            initialize c(x,v) to 0          ▷ no shortest paths to v found yet
        call MODIFIED-BFS(x)
procedure MODIFIED-BFS(s)
    for each v ∈ [1..n] do
        d_s[v] ← ∞
    L_0 ← {s}; d_s[s] ← 0; d ← 1
    while L_<d ≠ V do
        L_d ← ∅
        for each u ∈ L_{d-1} do
            for each v adjacent to u do
                if d_s[v] == ∞ then
                    add v to L_d; d_s[v] ← d;
                    p[v] ← u
                ▷ Choose which lines of code should go here, from the choices below:
                if d[v] == d then
                    c(s,v) ← c(s,v) + c(s,u)
        d ← d + 1
```

- ○ **if** $d_s[v] == d$ **then** $c(s, v) \leftarrow c(s, v) + 1$
- ○ $c(s, v) \leftarrow c(s, v) + 1$
- ● **if** $d_s[v] == d$ **then** $c(s, v) \leftarrow c(s, v) + c(s, u)$
- ○ $c(s, v) \leftarrow c(s, v) + c(s, u)$

4. What is the (worst-case) runtime of Algorithm COUNT-SHORTEST-PATHS from part 3? **Check all that apply.**

SOLUTION:

○ $\Theta(n+m)$

● $\Theta(n(n+m))$

● $\Theta(nm)$

○ $\Theta(n^2m)$

Extra page for scratch work