Final Exam

Quiz, 10 questions

1 point

1.

Consider a connected undirected graph with distinct edge costs. Which of the following are true? [Check all that apply.]

□ Suppose the edge e is the most expensive edge contained in the cycle C. Then e does not belong to any minimum spanning tree.

□ Suppose the edge e is not the cheapest edge that crosses the cut (A, B). Then e does not belong to any minimum spanning tree.

□ Suppose the edge e is the cheapest edge that crosses the cut (A, B). Then e belongs to every minimum spanning tree.

□ The minimum spanning tree is unique.

1 point

2.

You are given a connected undirected graph G with distinct edge costs, in adjacency list representation. You are also given the edges of a minimum spanning tree T of G. This question asks how quickly you can recompute the MST if we change the cost of a single edge. Which of the following are true? [RECALL: It is not known how to deterministically compute an MST from scratch in $O(m)$ time, where m is the number of edges of G.] [Check all that apply.]

□ Suppose $e \notin T$ and we increase the cost of e. Then, the new MST can be recomputed in $O(m)$ deterministic time.

□ Suppose $e \in T$ and we decrease the cost of e. Then, the new MST can be recomputed in $O(m)$ deterministic time.

□ Suppose $e \notin T$ and we decrease the cost of e. Then, the new MST can be recomputed in $O(m)$ deterministic time.

□

□ Suppose $e \in T$ and we increase the cost of e. Then, the new MST can be recomputed in $O(m)$ deterministic time.

1 point

3.

Which of the following graph algorithms can be sped up using the heap data structure?

□ Prim's minimum-spanning tree algorithm.

□ Kruskal's minimum-spanning tree algorithm.

□ Our dynamic programming algorithm for computing a maximum-weight independent set of a path graph.

□ Dijkstra's single-source shortest-path algorithm (from Part 2).

1 point

4.

Which of the following problems reduce, in a straightforward way, to the minimum spanning tree problem? [Check all that apply.]

□ The maximum-cost spanning tree problem. That is, among all spanning trees of a connected graph with edge costs, compute one with the maximum-possible sum of edge costs.

□ The single-source shortest-path problem.

□ Given a connected undirected graph $G = (V, E)$ with positive edge costs, compute a minimum-cost set $F \subseteq E$ such that the graph $(V, E - F)$ is acyclic.

□ The minimum bottleneck spanning tree problem. That is, among all spanning trees of a connected graph with edge costs, compute one with the minimum-possible maximum edge cost.

1 point

5.

Recall the greedy clustering algorithm from lecture and the max-spacing objective function. Which of the following are true? [Check all that apply.]

□ If the greedy algorithm produces a k-clustering with spacing S, then every other k-clustering has spacing at most S.

□ If the greedy algorithm produces a k-clustering with spacing S, then the distance between every pair of points chosen by the greedy algorithm (one pair per iteration) is at most S.

□ Suppose the greedy algorithm produces a k-clustering with spacing S. Then, if x, y are two points in a common cluster of this k-clustering, the distance between x and y is at most S.

□ This greedy clustering algorithm can be viewed as Prim's minimum spanning tree algorithm, stopped early.

1
point

6.

We are given as input a set of n jobs, where job j has a processing time p_j and a deadline d_j. Recall the definition of *completion times* C_j from the video lectures. Given a schedule (i.e., an ordering of the jobs), we define the *lateness* l_j of job j as the amount of time $C_j - d_j$ after its deadline that the job completes, or as 0 if $C_j \leq d_j$.

Our goal is to minimize the total lateness,

$\sum_j l_j$.

Which of the following greedy rules produces an ordering that minimizes the total lateness?

You can assume that all processing times and deadlines are distinct.

WARNING: This is similar to but *not* identical to a problem from Problem Set #1 (the objective function is different).

□ Schedule the requests in increasing order of deadline d_j

□ Schedule the requests in increasing order of the product $d_j \cdot p_j$

□ None of the other options are correct

□ Schedule the requests in increasing order of processing time p_j

1
point

7.

Consider an alphabet with five letters, $\{a, b, c, d, e\}$, and suppose we know the frequencies $f_a = 0.28$, $f_b = 0.27$, $f_c = 0.2$, $f_d = 0.15$, and $f_e = 0.1$. What is the expected number of bits used by Huffman's coding scheme to encode a 1000-letter document?

○ 2520

○ 2450

○ 2250

○ 2230

1
point

8.

Which of the following extensions of the Knapsack problem can be solved in time polynomial in n, the number of items, and M, the largest number that appears in the input? [Check all that apply.]

□ You are given n items with positive integer values and sizes, and a positive integer capacity W, as usual. You are also given a budget $k \leq n$ on the number of items that you can use in a feasible solution. The problem is to compute the max-value set of at most k items with total size at most W.

□ You are given n items with positive integer values and sizes, and a positive integer capacity W, as usual. The problem is to compute the max-value set of items with total size *exactly* W. If no such set exists, the algorithm should correctly detect that fact.

□ You are given n items with positive integer values and sizes, as usual, and *two* positive integer capacities, W_1 and W_2. The problem is to pack items into these two knapsacks (of capacities W_1 and W_2) to maximize the total value of the packed items. You are not allowed to split a single item between the two knapsacks.

□ You are given n items with positive integer values and sizes, as usual, and m positive integer capacities, $W_1, W_2, \ldots, W_m$. These denote the capacities of m different Knapsacks, where

m could be as large as $\Theta(n)$. The problem is to pack items into these knapsacks to maximize the total value of the packed items. You are not allowed to split a single item between two of the knapsacks.

1
point

9.

The following problems all take as input two strings X and Y, of length m and n, over some alphabet Σ. Which of them can be solved in $O(mn)$ time? [Check all that apply.]

□ Compute the length of a longest common subsequence of X and Y. (Recall a subsequence need not be consecutive. For example, the longest common subsequence of "abcdef" and "afebcd" is "abcd".)

□ Assume that X and Y have the same length n. Does there exist a permutation f, mapping each $i \in \{1,2,\ldots,n\}$ to a distinct $f(i) \in \{1,2,\ldots,n\}$, such that $X_i = Y_{f(i)}$ for every $i = 1,2,\ldots,n$?

□ Consider the following variation of sequence alignment. Instead of a single gap penalty α_{gap}, you're given two numbers a and b. The penalty of inserting k gaps in a row is now defined as $ak+b$, rather than $k\alpha_{gap}$. Other penalties (for matching two non-gaps) are defined as before. The goal is to compute the minimum-possible penalty of an alignment under this new cost model.

□ Compute the length of a longest common substring of X and Y. (A substring is a consecutive subsequence of a string. So "bcd" is a substring of "abcdef", whereas "bdf" is not.)

1
point

10.

Consider an instance of the optimal binary search tree problem with 7 keys (say 1,2,3,4,5,6,7 in sorted order) and frequencies $w_1 = .2, w_2 = .05, w_3 = .17, w_4 = .1, w_5 = .2, w_6 = .03, w_7 = .25$. What is the minimum-possible average search time of a binary search tree with these keys?

○ 2.18

○

○ 2.33

○ 2.29

○ 2.23

□ I, **Thanh Chi Doan**, understand that submitting work that isn't my own may result in permanent failure of this course or deactivation of my Coursera account.

Learn more about Coursera's Honor Code

Submit Quiz

👍 👎 ⚑