Final Exam Solutions

Question	Points
True/False	15
Sort and Dijkstra	6
Search Trees	9
Graph Consulting	25
Cooking Online	35
EXTRA CREDIT: Suspicious Traveler	0
Total:	90

Problem 1. [15 points] **True/False** (5 parts)

(a) [3 points] **T F** Inserting n elements into an empty dynamic array (via table doubling) takes worst case $O(n)$ time.

Solution: True. The expected time only happens when you are dividing by n to get constant time. The overall time is worst case.

(b) [3 points] **T F** You are given a hash table implemented using a universal hash function where collisions are resolved via chaining. If you insert n elements into the empty hash table, the length of any chain can be at most $O(\log n)$.

Solution: False, the length of a chain might be linear in the worst case.

(c) [3 points] **T F** Johnson's and Floyd-Warshall algorithms have the same asymptotic running time when applied to dense graphs.

Solution: True, $O(|V|^3) = O(|V|^2 \log |V| + |V||V|^2)$.

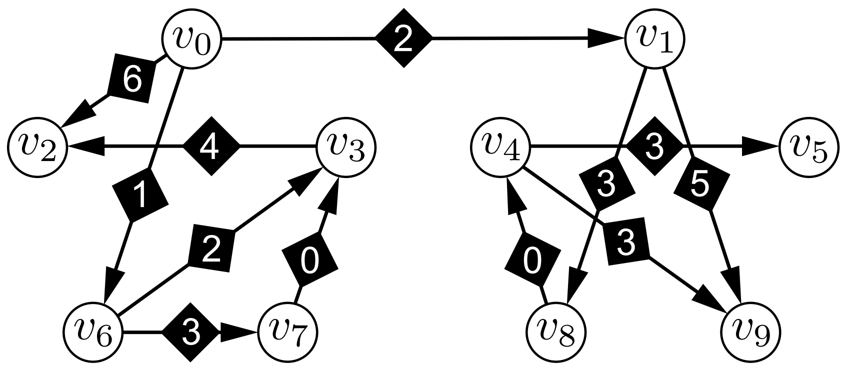
(d) [3 points] **T F** If a graph contains no negative-weight cycles, there exists an ordering of edges such that Bellman-Ford computes shortest paths by relaxing each edge at most once.

Solution: True, if there are no negative-weight cycles, there exists a directed shortest path tree. Relaxing in a topological sort order of that tree will only relax each edge once.

(e) [3 points] **T F** Taking a sorted array and converting it into a min-heap takes $O(N)$ time; you should simply ignore the fact that the array has some a-priori order and run heapify.

Solution: False. It takes no time; it is already a min heap!

Problem 2. [6 points] **Sort and Dijkstra** (3 parts)



Fill in the following tables based on the weighted directed graph above.

(a) [2 points] List the vertices in a topologically sorted order (many solutions exist).

v_0									

Solution: $(v_0, \{(v_6, v_7, v_3, v_2), (v_1, v_8, v_4, \{v_5, v_9\})\})$
(vertices inside $\{\cdot\}$ may be interleaved in any order)

(b) [2 points] List the shortest path distance to each vertex from vertex v_0.

$\delta_0(v_0)$	$\delta_0(v_1)$	$\delta_0(v_2)$	$\delta_0(v_3)$	$\delta_0(v_4)$	$\delta_0(v_5)$	$\delta_0(v_6)$	$\delta_0(v_7)$	$\delta_0(v_8)$	$\delta_0(v_9)$
0									

Solution: $(0, 2, 6, 3, 5, 8, 1, 4, 5, 7)$

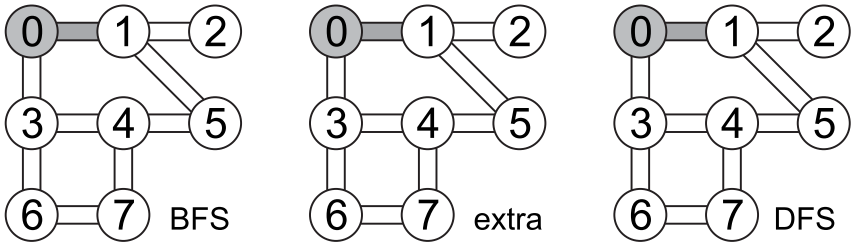
(c) [2 points] List the vertices in the order in which Dijkstra would remove them from the priority queue in a search starting at v_0.

v_0									

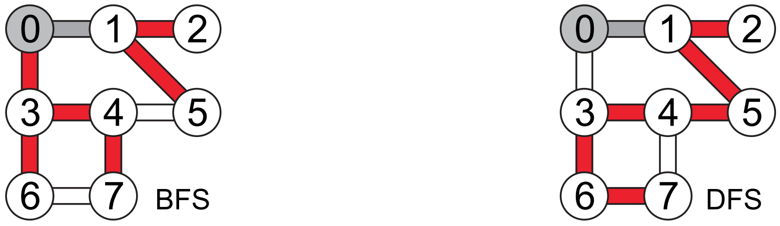
Solution: $(v_0, v_6, v_1, v_3, v_7, v_8, v_4, v_2, v_9, v_5)$

Problem 3. [9 points] **Search Trees** (2 parts)

(a) [5 points] Below are three drawings of the same undirected graph. Color in edges corresponding to the tree of parent pointers constructed by a search of the graph, using breadth-first search on the left, and depth-first search on the right (there's a third in the middle in case you mess up). Begin your search starting at the filled in node **prioritizing vertices with lower indices** whenever ambiguity exists. The first edge has been colored in for you.



Solution:



(b) [4 points] Describe a graph on n vertices for which the order of vertices first visited by both breadth-first and depth-first search could be the same.

Solution: There are many possible solutions to this problem. The two most obvious solutions would be a chain of length n with search starting at an end, and a star containing $n - 1$ leaves emanating from a central hub with search starting from the hub.

Problem 4. [25 points] **Graph Consulting** (4 parts)
For each of the following four scenarios, choose a graph algorithm that best solves the problem. Specifically: describe a graph related to the problem, name an algorithm to apply to the graph, justify your choice of algorithm, and state the algorithm's running time. You do not need to explain algorithms; so you can just say "topological sort" or "Dijkstra" without explaining further.

(a) [6 points] A cell phone carrier transfers calls through a connected network of cell towers. Each tower can transfer calls to at most five other towers. Sending a call from one tower to another decreases the quality of the call by a known multiplicative factor between 0 and 1; different tower to tower transfers may decrease quality by a different factor. Help the carrier connect a call from one tower to another to maximize the quality of the connected call. Hint: What weights should you use? Consider the fact that logarithms convert products into sums.

Solution: Construct a directed graph of connections between cell towers, weighted by the negative logarithm of the quality decrease factor for each connection, each of which will be positive. Run Dijkstra to find the shortest weighted path from one tower to another. If a binary or Fibonacci heap is used for the priority queue in Dijkstra, the running time is $O(|V| \log |V|)$, which is asymptotically faster than Bellman-Ford. Neither BFS nor topological sort relaxation apply as the graph has multiple weights and may contain cycles. While not what the problem was asking, applying Johnson to solve the all pairs problem in $O(|V|^2 \log |V|)$ is also a reasonable approach.

- 2 points for description of graph
- 1 point for choice of a correct algorithm
- 2 points for justification
  - Give 0 points if algorithm not best choice
  - Give 1 point if justification incomplete
- 1 point for running time

(b) [7 points] Carah Sonnor is flying a drone in a park with her young son. After a while, the drone runs out of batteries and crash lands in a two foot tall hedge maze, whose walls are a subset of edges from a square grid. From the entrance of the maze, Carah can see where the drone landed and the layout of the maze. Help Carah direct her son along a fastest route through the maze to retrieve the drone.

Solution: Construct an undirected graph connecting an edge between adjacent cells of the square grid if a wall is not between them. Because each edge has the same length, run BFS from the maze entrance to the drone to construct a shortest path. This algorithm runs in linear time with respect to the size of the maze, and is chosen because it is faster than other algorithms. Dijkstra could also be used if the priority queue is implemented with a first-in-first-out queue on a graph with all equal edge weights, but that is simply BFS.

- 2 points for description of graph
- 2 point for choice of a correct algorithm
- 2 points for justification
  - Give 0 points if algorithm not best choice
  - Give 1 point if justification incomplete
- 1 point for running time

(c) [6 points] San Holo is the debt-ridden pilot of a freight transport spaceship. He can fly directly between any two planets in the galaxy, though each one-way flight route demands its own fixed transport expenses. San can offset expenses by buying goods on one planet and selling them on another. His cargo hold can store at most one standard shipping container full of a single good, and local regulations require the sale of all cargo upon reaching any destination. Each planet buys or sells only a small set of possible goods at fixed prices. Find out if there is a way for San to **continually** make money in this trade network in order to pay off his debts.

Solution: Construct the complete directed graph on the planets in the galaxy. Weight each directed edge by the expenses of flying that route minus the maximum possible revenue that could be gained on the route; the maximum possible revenue can be computed in linear time with respect to the number of goods purchasable at the starting planet and sellable at the destination. This graph could have positive or negative weights, and a trade route that can continually make money represents a negative weight cycle in the graph. Run Bellman-Ford to find such a cycle, which will run in cubic time in the number of planets, as all routes between planets are present. Bellman-Ford detects negative weight cycles, while faster algorithms we know can not be used for that purpose. Because the graph is dense, Floyd-Warshall could also be used to detect negative weight cycles in cubic time.

- 2 points for description of graph
- 1 point for choice of a correct algorithm
- 2 points for justification
  - Give 0 points if algorithm not best choice
  - Give 1 point if justification incomplete
- 1 point for running time

(d) [6 points] A group of IIT-Delhi students decide to transfer to Ashoka. They want to drive to Ashoka along a shortest possible route measured by km driven, never looking back: that is, while driving, their straight-line distance to IIT-D should always **strictly increase**. Being IIT-D students, they are really bad at planning and math unless things are given to them in an MCQ format. Help them plan their trip, assuming there exists at least one route fitting their criteria.

Solution: Construct a directed graph of roads along which you can travel strictly away from IIT-D, with each road weighted by road length. This graph is acyclic, so topologically sort intersections according to their distance from IIT-D using DFS, and relax edges in topological sort order to find a shortest path. This algorithm runs in linear time with respect to graph size which is faster than the other algorithms.

- 2 points for description of graph
- 1 point for choice of a correct algorithm
- 2 points for justification
  - Give 0 points if algorithm not best choice
  - Give 1 point if justification incomplete
- 1 point for running time

Problem 5. [35 points] **Cooking Online** (2 parts)
Chulia Jild wants to make a responsive website where people can find and share recipes. Let R represent the number of recipes listed on the website at any given time. Each recipe has a name and contains a constant length description of ingredients, cooking instructions, and unit serving price for the meal.

(a) [15 points] Describe in detail a database that supports each of the following operations, describe how it supports each operation, argue that each operation achieves the stated running times. **State** whether **each running time** is a worst-case, expected, and/or amortized bound.

- `Add_Recipe`: add a recipe to the database in $O(\log R)$ time.
- `Find_Recipe`: return a recipe given its name in $O(1)$ time.
- `Similar_Price`: given a recipe name, return a list of 10 recipes closest in price to the given recipe in $O(\log R)$ time.

Solution: Construct a balanced binary search tree (like an AVL tree) with recipes stored at nodes, sorted by price. Additionally, store a hash table mapping a recipe name to its respective node in the tree.

- `Add_Recipe` will add the recipe as a tree node in worst case $O(\log R)$ time, and then add its name and a pointer to its tree node in expected $O(1)$ time, leading to amortized and expected $O(\log R)$ running time.
- `Find_Recipe` will search the hash table for the recipe name and return the recipe found at the linked tree node. This will run in expected $O(1)$ time.
- `Similar_Price` will find the recipe in the tree using the hash table in expected $O(1)$ time and then call `predecessor` and/or `successor` on the node a constant number of times to find 10 recipes on the website that are closest in price. Each of these calls takes worst case $O(\log R)$ time, so the operation takes expected $O(\log R)$ time.
- 3 points for balanced binary search tree keyed by price (or other valid structure)
- 2 points for hash table keyed by name (or other valid structure)
- 1 points for mentioning link between them
- 2 points for description of `Add_Recipe`
- 2 points for description of `Find_Recipe`
- 3 points for description of `Similar_Price`
- 1 point for stating `Add_Recipe` is **both** expected and amortized
- 1 point for stating `Find_Recipe` is expected
- 1 point for stating `Similar_Price` is expected
- Partial credit may be awarded for rubric items worth more than 1 point

(b) [20 points] Each individual recipe contains at most 20 ingredients, though there may be many more ingredients contained in all recipes listed on the website. Chulia wants users to be able to search the website for ingredients.

Modify your database from part (a) so it supports **all three** operations from part (a), **in addition** to the following three. Describe how the new database supports each operation, and argue that each operation achieves the stated running times. **State** whether **each running time** is a worst-case, expected, and/or amortized bound.

- `Ingredient_Recipe`: return a list of recipes that use a given ingredient in $O(R')$ time, where R' is the number of recipes returned.
- `Best_Recipe`: given a list of N distinct ingredients, return the names of two recipes with highest and lowest price, that can be made using any subset of those ingredients in $O(R + N)$ time.
- `Saffron_Count`: return the number of recipes having saffron as an ingredient and price less than or equal to a queried price, in $O(\log R)$ time.

Solution: Augment the balanced binary search tree from (a) to also maintain at each node, the number of recipes in the subtree rooted at the node that contain saffron. These subtree sums can be maintained during insertion in $O(\log n)$ time by increasing the count of each ancestor of an inserted node by one if the node's recipe contains saffron. Additionally, maintain an ingredient hash table keyed on ingredient names, with each ingredient pointing to a dynamic array of recipes containing that ingredient.

- `Add_Recipe` is modified to update saffron subtree sums as described above, and add the provided recipe to each of its ingredient's recipe array linked from the ingredients hash table, which can be done in amortized and expected $O(1)$ because 20 is a constant.
- `Find_Recipe` and `Similar_Price` do not need to be changed.
- `Ingredient_Recipe` should return the recipe array associated with the ingredient from the ingredient hash table. This array can be found in expected $O(1)$ and has size $O(R')$, so can be returned in expected $O(R')$ time.
- `Best_Recipe` will first build a hash table on the N ingredients. Then for each recipe from an in-order traversal of the recipe tree, look up the recipe's ingredients in the queried ingredients hash table, marking it as valid if all of the recipe's ingredients are contained in the queried set. Return the first and last valid recipe, which will have the lowest and highest price respectively. Building the queried ingredient hash table takes expected $O(N)$, the in-order traversal takes worst case $O(R)$, while there are $O(R)$ look ups into the queried ingredient hash table taking expected $O(R)$ time total, yielding expected $O(R + N)$ time.
- `Saffron_Count` searches for the queried price in the recipe tree, and during the search, calculates the number of recipes containing saffron below the queried price by adding or subtracting a constant number of subtree sums local to each node

during the search. The search touches at most $O(\log R)$ nodes, doing a constant amount of work at each node, so this operation runs in worst-case $O(\log R)$ time.

- 3 points for augmenting tree with subtree sums (or other valid structure)
- 3 points for hash table of dynamic arrays (or other valid structure)
- 2 points for modification of `Add_Recipe`
- 1 point for mentioning `Find_Recipe` and `Similar_Price`
- 2 points for description of `Ingredient_Recipe`
- 3 points for description of `Best_Recipe`
- 3 points for description of `Saffron_Count`
- 1 point for stating `Ingredient_Recipe` is expected
- 1 point for stating `Find_Recipe` is expected
- 1 point for stating `Saffron_Count` is worst-case
- Partial credit may be awarded for rubric items worth more than 1 point

Problem 6. [0 points] **EXTRA CREDIT: Suspicious Traveler** (3 parts)
Srya Atark lives in Easteros, a nation of N towns connected by a network of T trails, where each trail connects two neighboring towns. Each town has a **unique numeric rank** representing its **loyalty** to the Queen, with the capital city the most loyal town of all. Srya has a map of Easteros, labeled with the length of each trail and the loyalty of each town. Townspeople are very suspicious. When Srya travels along a single trail from town A to town B, she will **attract suspicion** if town A is strictly less loyal to the queen than town B.

For each of the following three parts, describe a graph, and a graph algorithm applied to the graph, that will help Srya plan her route. Justify your choice of algorithm, and state your algorithm's running time. Each part is worth 10 extra credit points!

(a) [0 points] Srya Atark needs to travel from the capitol to her home in Summerfell to deliver a secret message to her sister. Describe an efficient algorithm to find the shortest length route from the capitol to Summerfell that **never** traverses a trail that would attract suspicion.

Solution: Construct a weighted directed graph on the N towns, with a directed edge connecting town a to town b if the loyalty of town a is higher than the loyalty of town b, and there is a trail connecting a and b, weighting the edge by the trail's length. This graph is acyclic because the total order specified by loyalty represents a topologically sorted order. This graph takes $O(N + T)$ time to construct. Relax edges in this order to find weighted shortest paths from the capitol in $O(N+T)$ time, and return the shortest path to Summerfell. Such a route will never traverse a trail that would attract suspicion because such trails were not included in the graph. Note that such a path might not exist, even if the capitol and Summerfell are connected via trails.

- 4 points for description of graph
- 3 points for choice of a correct algorithm
- 3 points for justification of algorithm (remove here if correct, not optimal)
- 2 points for running time based on algorithm described
- Partial credit may be awarded

(b) [0 points] Having delivered her message to Summerfell, Srya must return to the capitol, which might be impossible to do without attracting suspicion. Thus she will take her time, spending one entire day sneaking along each trail between two cities. However, when she traverses a trail that attracts suspicion, she will spend three more days hiding in the city before moving on. Describe an efficient algorithm to find the minimum number of days that it will take Srya to return to the capitol.

Solution: Construct an unweighted directed graph connecting the N towns, with one directed edge from a to b if the loyalty of town a is higher than the loyalty of town b, and there is a trail connecting a and b; as well as four consecutive directed edges from b to a, representing the one day of travel and three days for waiting after attracting suspicion.

This graph has at most $5T$ edges and $N+3T$ vertices, so takes $O(N+T)$ time to construct. Run breadth-first-search to find shortest paths from Summerfell in $O(N+T)$ time. The number of edges in the shortest path to the capitol is three more than the minimum number of days that it will take for Srya to return, as we do not count days hiding after entering the capitol.

- 4 points for description of graph
- 3 points for choice of a correct algorithm
- 3 points for justification of algorithm (remove here if correct, not optimal)
- 2 points for running time based on algorithm described
- Partial credit may be awarded

(c) [0 points] Upon returning to the capitol, Srya steals a dragon egg from the Queen's dungeon, causing a panic. She must flee back to Summerfell as fast as she can, running along trails all the way back home at a fixed maximum speed S without rest. However, when entering a town from a trail that **attracts suspicion**, a person loyal to the Queen will fight her, delaying her travel by a fixed amount of time D. Describe an efficient algorithm to return Srya home as quickly as possible.

Solution: Construct a weighted directed graph on the N towns, with directed edge from a to b when there is a trail connecting a and b; if the loyalty of town a is higher than town b, weight the edge by its length divided by S, while if the loyalty of town b is higher (and b is not Summerfell), weight the edge by its length divided by S plus D. This graph has N vertices and $2T$ edges, so can be constructed in $O(N+T)$ time. Because all distances are positive, so are all edge weights, so run Dijkstra to compute shortest paths from the capitol in $O(N \log N + T)$ time. The shortest path from the capitol to Summerfell will have total weight equal to the shortest time for Srya to return home. We cannot use linear time approaches as the graph has cycles and non-integral weights, so because all weights are positive, we can use Dijkstra, will be faster than Bellman-Ford.

- 4 points for description of graph
- 3 points for choice of a correct algorithm
- 3 points for justification of algorithm (remove here if correct, not optimal)
- 2 points for running time based on algorithm described
- Partial credit may be awarded