COSC 302, Practice Exam #2
April 3, 2018

Honor Code

I agree to comply with the spirit and the rule of the Colgate University Academic Honor Code during this exam. I will not discuss the contents of this exam with other students until all students have finished the exam. I further affirm that I have neither given nor received inappropriate aid on this exam.

Signature: ______________________________

Printed Name: ______________________________

Write and sign your name to accept the honor pledge. Do not open the exam until instructed to do so.

You have 50 minutes to complete this exam.

There are 5 questions and a total of 68 points available for this exam. Don't spend too much time on any one question.

If you want partial credit, show as much of your work and thought process as possible.

Question	Points	Score
1	6	
2	12	
3	15	
4	15	
5	20	
Total:	68	

1. (6 points) Answer the following True/False questions by clearly circling your answer. No justification is required.

   (a) **True or False**: In a red-black tree storing n elements, every simple path from the root to a descendant leaf has $\Omega(\lg n)$ nodes.

   Solution: True. Since each such path has the same number of black nodes, every path to a descendent leaf has precisely $\Theta(\lg n)$ nodes.

   (b) **True or False**: The worst-case time to search for an element in a hash table with open addressing is $\Theta(1)$.

   Solution: False, the worst case is $\Theta(n)$.

   (c) **True or False**: In a graph $G = (V, E)$, a simple path cannot have more than $|V| - 1$ edges.

   Solution: True. The path can have no more than $|V|$ vertices on it, since there cannot be cycles. Such a path would have $|V|-1$ edges between the $|V|$ vertices.

   (d) **True or False**: The Bellman-Ford algorithm always runs in $\Theta(VE)$ time.

   Solution: True. There are two nested for loops, one over the vertices, and one over the edges.

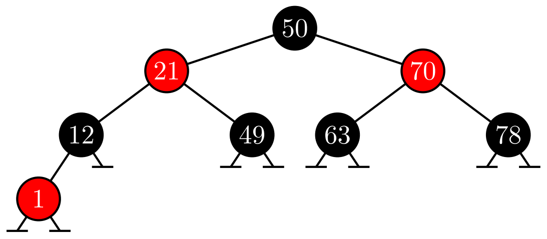
   (e) **True or False**: Every graph has a topological ordering.

   Solution: False. Only dags have a topological ordering.

   (f) **True or False**: Breadth-first search on an undirected graph yields only tree edges and cross edges.

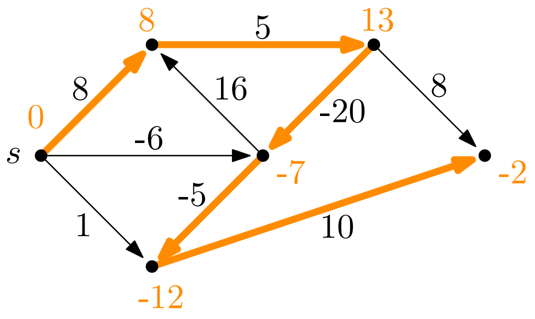
   Solution: True. However, note that BFS can yield back edges, but only if the graph is directed.

2. (a) (6 points) Consider the following binary search tree. Can its nodes be colored so that it is a valid red-black tree? If so, color the red nodes. If not, explain why not.



Solution: Yes. See the coloring above.

(b) (6 points) Highlight a shortest-paths tree with source vertex s on the following graph.



3. (15 points) Mr. Skeptical questions the value of implementing a priority queue with a max-heap data structure. In particular, he believes a balanced binary search tree (BBST), such as a red-black tree, can do everything a max-heap can do with the same running time. You decide to implement several operations to see if he is right.

   (a) Briefly describe how to implement EXTRACT-MAX with a BBST. Give as tight as possible best- and worst-case bounds on the running time of your algorithm.

   Solution: With a BBST, we simply find the max element in the tree by repeatedly following the right child beginning at the root, and when the right child of the current node is NIL, we have found the maximum element in the BBST. We can then remove this value from the tree and return it. In a balanced binary search tree such as a Red-Black tree, this has takes time $\Theta(\lg n)$ in both the best and worst case, since every path from the root to leaf has height $\Theta(\lg n)$ and removal requires us to traverse this height to find a predecessor or successor node to remove.

   (b) Briefly describe how to efficiently build the proposed priority queue data structure given an input array of elements with their priorities. Give as tight as possible the best- and worst-case bounds on the running time of your algorithm.

   Solution: Simply insert every element from the array into the BBST. This takes time $\Theta(n \lg n)$ (both best- and worst-case) because of the time to insert all n elements into a BBST is $\sum_{i=1} n\Theta(\lg i) = \Theta(\lg(n!)) = \Theta(n \lg n)$.

   (c) Is Mr. Skeptical right? Circle one: **Yes / No** Explain your answer.

   Solution: No! If we use a heap, the time to build the priority queue would be $\Theta(n)$, which is strictly faster than the $\Theta(n \lg n)$ time to build a BBST.

4. (15 points) In this problem we will solve the minimum spanning tree problem with edge weight constraints. Suppose you are given a connected, undirected, weighted graph $G = (V, E, w)$.

   (a) Suppose that all edge weights are 3. Describe an algorithm to compute a minimum spanning tree in G in $O(V + E)$ time. Argue for, but do not formally prove, correctness of your algorithm.

   Solution: Since all edge-weights are equal, any spanning tree has the same weight of $3|V|$. Thus, we can run BFS in time $O(V + E)$ to compute a breadth-first search tree, which is a spanning tree, and hence is a minimum spanning tree.

   (b) Suppose that all edge weights are now 1 or 0. Describe an algorithm to compute a minimum spanning tree in G in $O(V+E)$ time. Formally prove that your algorithm is correct.

   Solution: Note that since all edges are one of two weights, we can implement a special priority queue, using an array $Q[0..1]$ of size two, where each cell contains a linked list of vertices, indexed by weight 0 or 1. Such a priority queue allows us to implement Prim-Jarník in time $O(V + E)$ as follows. Assuming each vertex is a value from 0 to $|V| - 1$, we keep a Boolean array inTree$[1..n]$, indexed by vertex, which has inTree$[v]$ = true if and only if v is in our growing MST T. When we add a vertex v to our growing MST T, we add all edges incident to a vertex not in the tree to the priority queue, with the weight of their edge (0 or 1).

   We then call ExtractMin on the priority queue, giving us the vertex w of lowest weight from Q in time $\Theta(1)$: we first try to pull from $Q[0]$, and if it is empty, from $Q[1]$. If w is in the tree, we again call ExtractMin(), and continue calling ExtractMin() until we get a vertex w with inTree$[w]$ = false. This vertex is the next vertex to add to our MST, as its edge has minimum weight leaving the tree. Correctness directly follows from the correctness of Prim-Jarník.

5. (20 points) It's snowing outside and you and a friend are bored. You decide to make up a game with building blocks. Your friend gives you a box of k blocks of various positive integer heights, and challenges you to build a tower of height n (which is also a positive integer) with some subset of your blocks. Can you build it? You decide to solve this problem with dynamic programming.

   (a) Suppose it *is* possible to build a tower of exactly height n using a subset of your k blocks. Define the subproblems that you would use to solve this problem efficiently with dynamic programming. Give a recurrence that can be used to solve this problem.

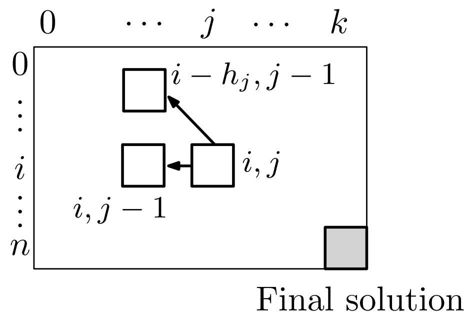
Solution:
We number each of the blocks 1 to k, and let block i have height h_i. This is very similar to the 0/1-Knapsack Problem, and can be viewed as k items, with values h_i, and weights h_i, with a knapsack weight of n. Let $T_{i,j}$ be the largest height of a tower that can be built, with height at most i, using blocks 1 to j. Then,

$$T_{i,j} = \begin{cases} -\infty & j < 0, \\ -\infty & i < 0, \\ 0 & \text{if } j = 0 \text{ or } i = 0, \\ \max\{T_{i,j-1}, T_{i-h_j,j-1} + h_j\} & \text{if } i > 0 \text{ and } j > 0. \end{cases}$$

If $T_{n,k} = n$, then it is possible to build a tower of height n with all k blocks.

   (b) Sketch the array that you would use to solve this problem with dynamic programming and draw arrows to indicate an order in which the array should be filled. Note the array's dimensions, and mark where the final solution is stored.

Solution:



The array can be filled in from top to bottom, left to right, or left to right, top to bottom.

Problem continued on next page →

(c) Now suppose that it might not be possible to build a tower of height n. How can you efficiently determine whether or not it is possible?

> **Solution:** If $T_{n,k} = n$, then it is possible to build a tower of height n with all k blocks, otherwise it is not.

(d) Describe how to change your algorithm to efficiently determine the largest height $h \leq n$ of a tower that can be built with the k blocks.

> **Solution:** Return $T_{n,k}$, this will be the highest tower height possible with the k blocks, with height at most n.

(e) Now describe how to change your algorithm to efficiently determine the smallest height $h \geq n$ of a tower that can be built with the k blocks.

> **Solution:** Let H be the sum total of all the block heights, that is
>
> $$H = \sum_{1 \leq i \leq k} h_i \, .$$
>
> We fill in a table, with dimensions $H \times k$ (instead of $n \times k$). We then iteratively check values $T_{n,k}$, $T_{n+1,k}$, until we reach $T_{H,k}$. We stop at the first value $T_{h,k}$ with height at least h, and return h.

(This page is intentionally blank. Label any work with the corresponding problem number.)

(This page is intentionally blank. Label any work with the corresponding problem number.)