Midterm #2

CIS 121—Fall 2016

In-class exam: Thursday, November 3rd, 2016.
Exam Starts: 10:30AM
Exam Ends: 11:50AM

This is a closed book exam. No computers or internet-connected devices are allowed during the exam. You are permitted to use one 8.5”x11” page with handwritten notes (these can be on both sides of the paper). If you need scratch paper, please get some from the front of the classroom. There are 8 questions on this exam, plus 2 extra credit questions, for a total of 10 questions. May the Force be with you.

Your name:

Your PennKey (this should be your username like ccb, not a number):

(circle one) Mon 12-1 Mon 2-3 Mon 4-5 Mon 5-6 Tues 12-1 Tues 2-3 Tues 3-4

Tues 4-5 Tues 5-6 Mon 1-2 Mon 3-4 Mon 6-7 Tues 6-7 Mon 11-12 Tues 1-2
1 Graphs
10 points

For each of the statements below, please say whether it is true or false, and give a 1 sentence explanation of your answer.

1. For a connected undirected graph with $N$ vertices, the minimum number of edges that it can have is $N - 1$.
   
   Solution True. In order for the graph to be connected, every vertex must be connected to at least one other vertex.

2. For a connected undirected graph with $N$ vertices, the maximum number of edges that it can have is $N^2$. Assume that we allow self-loops but not parallel edges.
   
   Solution False. The maximum number of edges is the number of pairs of vertices plus the number of self-loops, which is \( \binom{N}{2} + N = \frac{N(N-1)}{2} + N = \frac{N(N+1)}{2} \). $N^2$ would imply that the edges are directed.

3. In a typical adjacency matrix representation of a graph with $N$ vertices and $M$ edges, the amount of space required is $N^2$.
   
   Solution True. An adjacency matrix has $N$ rows and $N$ columns (one per vertex) to represent the edges.

4. If all edges in a graph have distinct weights, then the shortest path between two vertices is unique.
   
   Solution False. Even if no two edges have the same weight, there could be two paths with the same weight. For example, there could be two paths from $s$ to $t$ with lengths $3 + 5 = 8$ and $2 + 6 = 8$. These paths have the same length (8) even though the edge weights (2, 3, 5, 6) are all distinct.

5. A boolean adjacency matrix representation of a graph supports parallel edges.
   
   Solution False. In the boolean adjacency matrix, FALSE would indicate the lack of an edge, and TRUE would indicate the presence of an edge, so there would be no be a way to indicate multiple parallel edges.
2  Dijkstra’s running time
10 points

• Give a short English description of Dijkstra’s shortest path’s algorithm.

• What is the worst case running time of Dijkstra’s shortest path algorithm for a connected graph with $E$ edges and $V$ vertices when using a binary heap for the priority queue? What are the operations? How many times is each operation called? What is the running time for each operation?

Solution

• Dijkstra’s algorithm finds the shortest path from one vertex to all vertices in a graph. It stores the distances for the shortest paths found so far, and then uses a min heap to greedily look at the vertices in non-decreasing order of this distance in order to try to find a shorter path to all out-neighbors by going through the current vertex.

• At every iteration, Dijkstra’s pops a vertex off of the min heap that has at most $V$ vertices. Thus, we have a $O(V \log V)$ running time to pop off all the vertices from the heap. Throughout these iterations, it iterates through every single edge in the graph to potentially relax each of them at most once. Relaxing can take $O(\log V)$ time as it may have to decrease the key of a neighbor in the heap. Thus, we end with a $\sum_{v \in V} \deg(v) \log V = O(E \log V)$ running time to relax all edges. Our overall running time is $O((E + V) \log V)$. In the case of a connected graph, this is simply $O(E \log V)$. 
3 TSTs
10 points

1. What are the results of searching for the following keys?
   
   (a) CAT Solution 8
   (b) ACT Solution 9
   (c) CAG Solution null

2. Insert the key CCB with value 11 into the TST above. (You can draw on the figure itself. Be sure to draw all children, including null links).
   Solution

3. What is the shortest number of steps needed for a search hit for a key of length \( L \) in a TST? When is it guaranteed to occur?
   Solution \( L - 1 \). This will occur when all the connections between letters are downlinks. Downlinks are guaranteed to occur for the first key added to the tree and for any keys that are substrings of it.
4 R-way tries
10 points

1. Please define $R$ for an R-way trie.
   
   **Solution** $R$ is the maximum number of children per node or the number of characters in the alphabet.

2. What is the worst case running time for inserting a value for a key of length $L$? Please give your answer in big O notation, explaining any variables that you use, and saying when the worse case happens for insertion.

   **Solution** The worst case running time occurs when the key is not present in our trie. This means that we must create a node for each character in our key. Creating a node takes $O(R)$ time, since we must generate an array of size $R$, and since our key is of length $L$, the overall running time of this procedure is $O(RL)$.

3. What is the best case running time for inserting a value for a key of length $L$?

   **Solution** The best case running time occurs when the key we want to insert is already present in the trie. Since a search hit takes $O(L)$ lookup time, once we find this key we are able to simply set the value at the appropriate end node in our trie.

4. What operations can take less than $L$ time, and when do these occur?

   **Solution** Search misses can take less than $L$ time. They occur when there is no value for the search key in the tree and when the key is not a substring of some other string in the tree.
5 Balanced trees
15 points

1. Which of the following trees are balanced? Write “balanced” or “not balanced” next to each one. Give a short explanation about how it is balanced, or give a valid counterexample if it is not balanced.

(a) Binary search trees
   **Solution** Not balanced. Counterexample: inserting 1, 2, 3, . . . , n (in that order) would create a list-like structure.

(b) Heaps
   **Solution** Balanced. Heaps maintain the property of being almost-complete binary trees.

(c) Red-black trees
   **Solution** Balanced. Red-black trees maintain the property that the number of black nodes on any part from a particular node to the leaves of the tree is the same.

(d) 2-3 trees
   **Solution** Balanced. 2-3 trees maintain perfect balance by only splitting when a node becomes a 4-node.

(e) R-way tries
   **Solution** Not balanced. Counterexample: inserting a single string creates a linked list.

(f) Ternary search tries
   **Solution** Not balanced. Counterexample: inserting 'a', 'b', 'c', . . . , 'z' (in that order) creates a linked list.

(g) Subtrees corresponding to the connected components in the weighted Quick-Union algorithm
   **Solution** Balanced. The weighted quick-union optimization guarantees that the height of any subtree will not exceed \( O(\log_2 N) \).

2. What is the correspondence between the height of a balanced tree with \( C \) children, the number of elements stored in it, and the log function? As a reminder, \( \log_b(x) \) is the unique real number \( y \) such that \( b^y = x \). For example, as \( 64 = 2^6 \), then: \( \log_2(64) = 6 \).

   **Solution** The height of a balanced binary tree with \( N \) elements in it is \( O(\log_2 N) \). If each node in a balanced tree has \( C \) children, the tree’s height is \( O(\log_C N) \).

3. What impact does the height of a tree have on the worst case running time for standard search tree operations?

   **Solution** Operations on a search tree take time proportional to the height of the tree. In other words, operations are \( O(h) \) for height \( h \).

4. When does the height of a search tree change? Give your answer for a 2-3 tree and for a red-black tree.

   **Solution** For a 2-3 tree, the height changes when the root becomes a 4-node. There was a typo in the red-black tree part of this question, as it meant to ask: “When does the **black height** of a red-black tree change?” Then the answer would have been when a color flip occurs at the root, as that is analogous to the 2-3 case of the root splitting once it becomes a 4-node. As the question was worded, however, there are multiple cases in which the actual height of a red-black tree might change, and a rule regarding when the height changes is not clearly defined. Thus, we gave everyone the point for this part of the question.
6 Linear Probing Hash Table

15 points

1. Given the linear probing hash table above, record where the following keys would be added. Write the key into the appropriate slot in the table, assume that they are added in the order that we give them in.

(a) Insert Z. Hash(Z) = 9.
(b) Insert Y. Hash(Y) = 6.
(c) Insert C. Hash(C) = 4.
(d) Insert K. Hash(K) = 14.

Solution The resulting hash table after the given insertions is shown below:

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keys</td>
<td>P</td>
<td>M</td>
<td>K</td>
<td>A</td>
<td>C</td>
<td>S</td>
<td>H</td>
<td>Y</td>
<td>Z</td>
<td>E</td>
<td>R</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Hash tables store values associated with keys. How are values stored in a linear probing hash table?

Solution The values are stored in a parallel array, where each key and its associated value have the same index in their corresponding arrays.

3. What is the maximum number of keys that the linear probing hash table above could store? Why would it be undesirable to store that many keys in the hash table?

Solution This hash table could store at most 16 keys. It would be undesirable to fill the entire array because as the array fills up most keys tend to be displaced farther and farther from their actual hash index. This means that when we want to search for a key, the probe will tend to take longer.

4. Name another method that can be used to resolve collisions that result from the hash function, and give a pro and a con for using it over linear probing.

Solution Separate chaining, where collisions are added to a linked list. Pro: Its performance degrades more gracefully than that of linear probing. Con: Separate chaining requires extra memory to represent the linked lists.

Alternate answer #1: Double hashing (with associated pro/con).
Alternate answer #2: Quadratic probing (with associated pro/con).
7 Simple Data Structures Support Complex Algorithms
15 points

Kruskal’s algorithm for computing a minimum spanning tree can be described as follows:

- Consider edges in ascending order of weight
- Add the next edge to the tree unless doing so would create a cycle.
- Stop when we have added enough edges to form a spanning tree.

Say what data structures an implementation could use to support the below operations, and what their running time would be for those operations. If multiple data structures could be used, then pick the one with the smallest running time.

1. Ordering all of the edges in an undirected graph, starting with the edge with the lowest weight and going in order to finally return the edge with the highest weight.

   **Solution** Min priority queue (or min heap). Add all edges to the min priority queue (buildHeap), and call extractMin until the MST is complete. Building the heap takes $O(E)$ time. We call extractMin at most $O(E)$ times, and each extractMin call takes $O(lgE)$ time. Thus, the total running time is $O(E lg E)$.

   **Alternate solution #1:** Array. Put all $E$ edges into the array, and sort them using an $O(n lg n)$ sorting algorithm (e.g., mergesort, quicksort) in non-decreasing order according to their weights. The total running time is $O(E lg E)$.

   **Alternate solution #2:** Balanced binary search tree. Put all $E$ edges into a balanced BST, and return an in-order traversal of the BST. The total running time is $O(E lg E)$.

2. Finding whether there is a cycle formed when adding an edge to the partial MST that we have constructed so far, and adding that edge to the data structure if it doesn’t create a cycle.

   **Solution** Union-find. Add all edges that are currently in the MST to the union-find. Then, for the new edge $e = (u, v)$, if $\text{find}(u) \neq \text{find}(v)$ (i.e., if $u$ and $v$ are in different connected components in the partial MST), add $e$ to the MST with union($u, v$). The running time of the final find/union combination is $O(lg^* V)$ amortized.

   **Note:** A set (or list) of already-seen vertices cannot be used for cycle detection, since a partially completely MST generated by Kruskal’s is a forest (not a single tree).

3. Returning the set of edges that form the MST in the order that Kruskal’s algorithm added them to the MST.

   **Solution** Queue (FIFO data structure). When a new edge in the MST is discovered in Kruskal’s, enqueue that edge. When Kruskal’s is finished, dequeue all edges to get the order in which edges were added to the MST. enqueue and dequeue both take amortized $O(1)$ time, and both operations are called $V - 1$ times. Thus, the total running time is $O(V)$. 
8 Make money with an app  
15 points

An entrepreneurial team of Penn undergrads builds an app using Dijkstra’s shortest paths algorithm (not the buggy version in the slides). Their app lets drivers find the shortest distance to their destination.

1. How is the system of roads represented in the graph? What are the vertices and edges? What type of graph is it?
   
   Solution We will construct a weighted directed graph. For each intersection $I$ in the system of roads, add a vertex $v_I$ to the graph. For each span of road of length $\ell$ between two intersections $I_i$ and $I_j$, add a directed edge $(v_{I_i}, v_{I_j})$ with weight $\ell$ to the graph. (It is also fine to say that destinations, instead of intersections, are vertices.)

The US congress has decided that in order to reduce the national deficit, all roads are now toll roads. Users of the app demand an update that will let them find the lowest cost route to their destination in terms of dollars instead of miles.

2. What do the Penn students need to do to update their graph?
   
   Solution The weights on edges would need to be updated to be the cost of driving on that span of road, instead of its distance.

The newly elected president decides to name the best, classiest road in each American city “Presidential Avenue”. The president decrees that instead of charging drivers a toll, the government will pay drivers whenever they drive on any street named Presidential Avenue.

3. Assuming that the graph automatically updates each time the tolls change, what happens to the edge weights in the graph? Can the app still give drivers the lowest cost ($) route to their destination?
   
   Solution By paying drivers to go on certain roads rather than charging them, the new president introduces edges with negative weights. Dijkstra’s algorithm does not work with negative weights, so the app fails to give valid directions.

4. How could the Penn students determine if the app could be fixed, or if it is impossible to correctly calculate shortest paths? If it is fixable, what is the solution?
   
   Solution The students need to determine if the graph contains negative cycles or not. If there are no negative cycles, then they can replace Dijkstra’s algorithm with the Bellman-Ford algorithm to find the shortest paths for drivers. They can also use the Bellman-Ford algorithm to detect whether there are negative cycles in the graph.

5. If the app can no longer provide driving directions, how could the Penn students still use it to make money for themselves?
   
   Solution If there are negative cycles, then they cannot use the Bellman-Ford algorithm to give drivers directions, but they can use it to find a negative cycle. By driving on that cycle, the students would make money. The payment they get from driving down Presidential Avenue would be greater than the cost of the toll roads that complete the cycle. Assuming that they make a profit when the cost of fuel is factored in, then they have found a way of making money through arbitrage. This could be exploited to take as much money from the US government as they wanted, which would drive up the deficit.
9 Extra Credit: Heaps with twice as many elements
5 points

Fifi wants to build a heap from two unsorted arrays, each of length $N$. She wants you to describe an algorithm that will combine the two arrays and put them into heap order. Describe the algorithm that you should use and its running time. Your answer should include brief descriptions of what heap order is, what sink and swim operators are, and what the running time is of applying sink or swim once.

Solution First, we combine the two arrays into one larger array of size $2N$. Then, we call `buildMaxHeap` on the new array. This will take $O(2N) = O(N)$ time. Heap order requires that the values for all children of any node must be less than the value of the node (or greater than, depending on what type of heap you're building). Assuming we are talking about a max-heap, we use a sink operation to help a node find its correct place in the heap when it may be smaller than some of its descendants. We use a swim operation to help a node find its correct place in the heap when it may be larger than some of its ancestors (definitions for a min-heap are the same, except substitute larger/smaller). The worst case running time for both of these operations is $O(\lg(N))$. In `buildMaxHeap`, we use the sink operation on each element in the front half of the array until we have a heap.
10 Extra Credit: Picking the right tool for the job (2 points each)  
12 points

Below there are a short description of programs that we would like to write. Please say which of the data 
structures or algorithms we learned so far in class would be most practical to use in each program. Give a 
1 sentence description why you picked that one.

1. Address book autocomplete: based on typing the first few letters of a name, list all contacts stored in 
an address book that match those first few letters.

   Solution: Trie, since it makes it easy to find words with common prefixes.

2. k-best list: Given a stream of a million numbers, keep track of the k highest numbers in a space efficient 
way.

   Solution: Min heap of size k, since it would allow \(O(\lg k)\) insertion and removal of elements. Whenever 
the heap stores more than \(k\) elements, delete the minimum element.

3. Given Facebook’s graph (where friendship is an edge and each node is a person), find all the connected 
components. Pick an algorithm that allows new friendships to be formed over time. You don’t have 
to support ending friendships.

   Solution: Use an undirected graph, and find its connected components using a union-find. UF allows 
dynamic connectivity (so it supports adding new friends, but not unfriending them).

   Note: DFS or BFS could be used to correctly find connected components on a snapshot of the Facebook 
graph; however, they would not support maintaining connected components over time. Using a union-
find reflects the online nature of the problem.

4. Spell check: given a dictionary containing correctly spelled words, check all words in a document, 
highlighting ones that do not match the dictionary.

   Solution: Use a hash table to store the dictionary. For each word in the document, if the hash table 
does not contain the word, highlight it. We also accepted using a trie, as the usage is analogous.

5. Given a map of cities connected by one-way roads, we want to identify groups of cities such that for 
any pair of cities in the group, it is possible to drive from the first city to the second and vice versa.

   Solution: Construct a directed graph. For each city \(A\) on the map, add a vertex \(v_A\) to the graph. For 
each one-way road from location \(A\) to location \(B\) on the map, add an edge \((v_A, v_B)\) to the graph. Run 
Kosaraju’s SCC algorithm on the graph, and return the graph’s strongly connected components.

6. We want to build a program by compiling individual files in some order. It may be the case that a 
file can only be compiled after certain other files have been compiled. How do we write a script that 
determines the correct order to compile the files? You can assume that there exists no cycles (e.g. it 
will not be the case that file A requires file B, but file B also requires file A).

   Solution: Construct a directed acyclic graph (DAG) from the file dependencies. Run topological sort 
using DFS on the graph, and return a topological ordering of the files.