Quiz 2

Instructions:
• Do not open this quiz booklet until directed to do so. Read all the instructions on this page.
• Write your name below and circle your recitation at the bottom of this page.
• Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem. Pages will be separated for grading.
• You are allowed two one-sided, letter-sized sheets with your own notes. No calculators or programmable devices are permitted. No cell phones or other communication devices are permitted.

Advice:
• You have 120 minutes to earn a maximum of 120 points. Do not spend too much time on any single problem. Read them all first, and attack them in the order that allows you to make the most progress.
• When writing an algorithm, a clear description in English will suffice. Using pseudo-code is not required.
• Do not waste time rederiving facts that we have studied. Simply state and cite them.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Parts</th>
<th>Points</th>
<th>Grade</th>
<th>Grader</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>120</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Name: ____________________________

Circle your recitation:

<table>
<thead>
<tr>
<th>R01</th>
<th>R02</th>
<th>R03</th>
<th>R04</th>
<th>R05/R07</th>
<th>R06</th>
<th>R08</th>
<th>R09</th>
<th>R10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brando</td>
<td>Brando</td>
<td>Parker</td>
<td>Alex</td>
<td>Danil</td>
<td>Peinan</td>
<td>Kevin</td>
<td>Allen</td>
<td>Daniel</td>
</tr>
<tr>
<td>Miranda</td>
<td>Miranda</td>
<td>Zhao</td>
<td>Jaffe</td>
<td>Tylumankov</td>
<td>Chen</td>
<td>Tian</td>
<td>Park</td>
<td>Manesh</td>
</tr>
<tr>
<td>10AM</td>
<td>11AM</td>
<td>12PM</td>
<td>12PM</td>
<td>1PM 2PM</td>
<td>1PM</td>
<td>2PM</td>
<td>3PM</td>
<td>4PM</td>
</tr>
</tbody>
</table>
Problem 0. What is Your Name? [2 points] (2 parts)

(a) [1 point] Flip back to the cover page. Write your name and circle your recitation section.

(b) [1 point] Write your name on top of each page.
Problem 1. True or False [14 points] (7 parts)

For each of the following questions, circle either T (True) or F (False). There is no need to justify the answers. Each problem is worth 2 points.

(a) T F The rolling hash technique enables us to compute and compare hashes of two arbitrary strings, each of length $l$, in $O(1)$ time.

(b) T F When using an adjacency matrix graph representation, relaxing all the edges in a graph $G(V, E)$ can be done in $O(E)$ time.

(c) T F Consider a modification to the table doubling procedure in which whenever the number of keys $n$ is at least $\frac{m}{2}$, where $m$ is the current size of the table, we set its new size $m'$ to be $m' = 2m + \lfloor \sqrt{m} \rfloor$. If we never delete any elements from our table, then the resulting amortized overhead of this modified table doubling is still only $O(1)$ per each hash table operation.

(d) T F If $G(V, E)$ is a directed graph in which all the edge weights are positive integers bounded by a fixed constant, then all shortest paths from a single source can be computed in $O(V + E)$ time.

(e) T F If we run Dijkstra’s algorithm on a graph in which there is a single negative-weight edge outgoing from the source $s$, then all the distances computed by this algorithm will be correct.

(f) T F The union of a shortest path from $s$ to $u$ and a shortest path from $u$ to $t$ must also be a shortest path from $s$ to $t$.

(g) T F Assume that a data structure supports some operation in $O(T)$ amortized time. Consider now a situation in which we initialize the data structure and then make $k$ calls to that operation. The worst-case time needed to serve the first of these calls is $O(T)$. 
Problem 2. Ben Bitdiddle is an Ace Trader [19 points] (3 parts)

Ben Bitdiddle has landed a summer internship at a high-frequency trading firm Algorithmic Power (AP). Given his extensive 6.006 expertise, he has been put in charge of analyzing various communication networks the firm uses to send and receive trading information.

(a) [5 points] Ben’s first project requires him to implement the Bellman–Ford algorithm to find single-source shortest paths on graphs that may contain negative-weight cycles. Working too quickly without checking his work, Ben accidentally reversed the order of the inner and outer loops controlling the edge relaxation in the algorithm, so he relaxed each edge $V-1$ times before moving on to the next edge.

Draw a graph for which Ben’s algorithm would provide an incorrect set of shortest path lengths. Make sure that you specify the graph, the directions and weights of all the edges, as well as the order of edge relaxations. No further justification is necessary.
(b) [6 points] Ben can’t find the error and correct the code he wrote in part (a), but he somehow discovers that by running this code multiple times without re-initialization, his results improve.

(i) What is the minimum number of times that Ben must run his defective code on a given graph $G = (V, E, w)$ to ensure that all shortest paths are correctly computed?

(ii) What is the worst-case asymptotic running time of Ben’s hacked version of his defective code to produce the correct result on a graph $G$? 

Don’t forget to provide a concise justification.
(c) [8 points] Ben’s next assignment is to compute the best route to support the traffic between the firm and the NASDAQ stock exchange. The underlying communication network is modeled as a directed graph $G = (V, E, w)$ with non-negative edge weights representing the average transit times along different network links. The AP headquarters corresponds to a vertex $s$ in that graph, and the NASDAQ stock exchange is a vertex $t$.

Ben needs now to compute the shortest $s \rightarrow t$ path in that communication network. Unfortunately, before he entered all the data from his paper diagram of that network, he spilled coffee on it and can no longer read the average transit time $w(e)$ of one of the network links $e$. This is especially regrettable because Ben remembered his supervisor telling him that the shortest $s \rightarrow t$ path is very likely to contain $e$. Ben would now like to reconstruct what the weight of $e$ would need to be in order to lie on such a shortest $s \rightarrow t$ path.

Help Ben by doing the following:

- Provide an efficient algorithm that will compute the largest edge weight $w^*$ that the edge $e$ can have while still being on a shortest $s \rightarrow t$ path in graph $G$.
- State the running time of your algorithm (no justification necessary).
- Argue, briefly, the correctness of your algorithm.
Problem 3. Computing the “Longest” Shortest Path [26 points] (5 parts)

Prof. Madry enjoys sightseeing, so whenever he travels he always tries to choose a route that is as “round-about” as possible, provided it does not increase the total time needed to reach the destination.

To help Prof. Madry plan his trips, you are asked to design a fast algorithm for computing the “longest” shortest path. Specifically, consider a weighted directed graph \( G = (V, E, w) \), with all weights being positive, and an origin \( s \). The task is to compute, for each possible destination \( t \), the number of edges on the “longest” shortest \( s \rightarrow t \) path, i.e., the shortest \( s \rightarrow t \) path in \( G \) that maximizes the number of visited vertices among all the shortest \( s \rightarrow t \) paths in \( G \).

(a) [5 points] Let us call an edge \( e = (u, v) \) tight if

\[
\delta(s, u) + w(u, v) = \delta(s, v),
\]

where \( \delta(s, v') \) is the shortest path distance from \( s \) to the vertex \( v' \).

Note that an edge is tight if and only if it is a part of some shortest path from \( s \) in \( G \).

Give an algorithm that computes the subgraph \( \hat{G} \) of \( G \) containing all the tight edges of \( G \) in \( O(E \log V + V \log V) \) time.
(b) [6 points] Argue that, for any destination \( t \),

(i) every \( s \leadsto t \) path in the subgraph \( \tilde{G} \) is a shortest \( s \leadsto t \) path in the graph \( G \);

(ii) every shortest \( s \leadsto t \) path in \( G \) is an \( s \leadsto t \) path in \( \tilde{G} \).

*Hint:* Note that if an edge \( e = (u, v) \) is tight then \( w(u, v) = \delta(s, v) - \delta(s, u) \).
(c) [4 points] Prove that the subgraph $\widehat{G}$ is acyclic.
(d) [5 points] Give an algorithm that finds the length $l(t)$ of the longest $s \leadsto t$ path in $\hat{G}$ for all destinations $t$ in $O(V + E)$ time. (Note that the graph $\hat{G}$ is unweighted.)

*Hint:* Part (c) might be useful here.
(e) [6 points] Consider now an algorithm that, first, uses part (a) to compute the graph $\hat{G}$ of all the tight edges in $G$ and, then, uses part (d) to find the length $l(t)$ of the longest $s \leadsto t$ path in that graph $\hat{G}$, for each destination $t$.

Prove that, for each destination $t$, the length $l(t)$ computed by the above algorithm is exactly the number of edges on the “longest” shortest $s \leadsto t$ path in $G$. 
Problem 4. Variants of Dijkstra’s Algorithm [18 points] (3 parts)

In addition to the standard Dijkstra’s algorithm, we discussed two other variants of that algorithm that sometimes might speed it up: bidirectional search and $A^*$ search. Still, these variants might perform differently on different graphs, and sometime work even worse than the standard Dijkstra’s algorithm. Your task will be to rank the effectiveness of each one of these three variants on the graphs provided below.

Specifically, for each graph below, give a ranking (by assigning ranks 1, 2, and 3) of these three variants of Dijkstra’s algorithm according to the number of attempted edge relaxations made when computing a single source, single target shortest path from $s$ to $t$ in that graph. (Rank 1 means the smallest number of such relaxations, and rank 3 means the largest number of them.) No justification is necessary.

Whenever an algorithm needs to break ties to decide which edge to relax next, consider the worst case, i.e., tie breaking that will result in a maximum number of edge relaxations for that algorithm.

Each of these graphs is undirected and each edge has unit weight.

Also, the potentials we use for $A^*$ search are given by the distance from $t$ in the horizontal direction only. That is, for each vertex $v$, its potential $\lambda_t(v)$ is equal to $|x(v) - x(t)|$, where $x(v)$ is the horizontal coordinate of vertex $v$. (This is almost the same potential used in the lecture and recitation notes, except that the vertical coordinate has been ignored.) To fix the scale, we let the distance between $s$ and $t$ along the horizontal axis be exactly 1. So, $\lambda_t(s) = 1$ and $\lambda_t(t) = 0$.

Note: Partial credit will be given based on the correctness of the relative rankings.

(a) [6 points] Dijkstra’s Alg. Bidirectional Search $A^*$ Search
(b) [6 points] Dijkstra’s Alg.  Bidirectional Search  $A^*$ Search

(c) [6 points] Dijkstra’s Alg.  Bidirectional Search  $A^*$ Search
Problem 5. Ben Bitdiddle’s Adventures with Hashing [24 points] (4 parts)

Ben Bitdiddle learned about open addressing and is eager to give this new paradigm a try.

At first, he wanted to use the linear probing technique, that is, to use a hash function \( h_L \) in which the \( i \)-th probe is given by

\[
h_L(k,i) = h(k) + i \pmod{m},
\]

where \( h(k) \) is a “classic” hash function that satisfies the simple uniform hashing assumption (SUHA).

However, he heard rumors that linear probing might perform poorly, so he decided to also consider applying the double hashing technique. Unfortunately, he did not realize that double hashing requires combining two different hash functions. He implemented instead a hash function \( h_D \) in which the \( i \)-th probe is given by

\[
h_D(k,i) = (h(k) + i \cdot (h(k) + 1)) \pmod{m},
\]

where the same “classic” hash function \( h \) is used twice and the “+1” was introduced to deal with the case of \( h(k) = 0 \).

(a) [5 points] Recall that the function \( h \) satisfies the simple uniform hashing assumption (SUHA). Does this imply that the hash function \( h_D \) that Ben designed satisfies the uniform hashing assumption (UHA)? Justify your answer.
(b) [5 points] Ben wants now to compare the performance of the hash functions $h_L$ and $h_D$. To this end, he constructs a hash table of size $m$, with $m$ even, that has the first half of its slots occupied. He inserts a new key $k$ using the hash function $h_L$ first.

Provide an asymptotic estimate (i.e., a $\Theta$-estimate) of the expected number of probes $P_L$ needed to insert that new key $k$ into that hash table if we use the hash function $h_L$. Document your work.
(c) [6 points] Next, Ben wants to test the hash function $h_D$ in the same scenario as $h_L$ was tested in part (b). That is, he again considers a hash table of size $m$, with $m$ even, that has the first half of its slots occupied, and this time wants to insert a new key $k$ into it using the hash function $h_D$.

Provide an asymptotic estimate (i.e., a $\Theta$-estimate) of the expected number of probes $P_D$ needed to insert that new key $k$ into that hash table if we use the hash function $h_D$.

Document your work.

*Hint:* Recall that $\sum_{i=1}^{n} \frac{1}{i} = \Theta(\log n)$. 
(d) [8 points] Ben starts to suspect that the hash function $h_D$ is better than the hash function $h_L$. However, he is not sure yet. To really confuse him, devise a scenario in which the hash function $h_L$ performs much better than the hash function $h_D$.

Specifically, devise an initial configuration of the occupied slots of the hash table such that the expected number $P'_L$ of probes needed to insert a new key $k$ into that hash table when using the hash function $h_L$ is asymptotically smaller than the expected number $P'_D$ of probes needed to insert $k$ into the hash table using the hash function $h_D$.

Remember to provide both a precise description of the initial configuration and asymptotic estimates of $P'_L$ and $P'_D$. Document your work.
Problem 6. Escaping the Dungeon of an Evil Overlord [17 points] (3 parts)

You have been trapped by an Evil Overlord in his dungeon! You need to find an exit as soon as possible, before the overlord realizes that he left your cell unlocked.

The Evil Overlord’s dungeon can be modeled as an $n$-vertex tree, i.e., an undirected graph with no cycles, in which vertices correspond to rooms and edges to connections between these rooms. Your starting room is $s$ and the exit room is $t$.

You don’t know anything more about that dungeon graph except that, due to architectural feasibility constraints, there is at most $B$ different vertices that are at a distance at most $D$ from $s$. We will refer to that property as being $B$-bounded. (Note that, a priori, the degree of each vertex can be arbitrary, as long as it does not make the graph be not $B$-bounded.)

Now, your job is to construct a strategy that, starting from the vertex $s$, explores the dungeon graph and finds the room $t$ containing the exit while trying to minimize the number of edges you traverse.

Note that whenever one travels from a room $v$ to room $v'$, one cannot simply “jump” there – one needs to follow the (unique) $v \sim v'$ path in the dungeon and traverse all the edges of this path, possibly revisiting many of the already visited rooms.

(a) [5 points] Consider a strategy that corresponds to visiting the rooms in a DFS order. Show that such strategy might perform pretty poorly. That is, that it might require us to traverse $\Omega(n)$ edges before finding the exit $t$, even if the exit $t$ is only at a distance $D = 1$ away from the initial room $s$.

More precisely, specify first an $n$-vertex dungeon graph that is $B$-bounded, for any $B \geq 3$, with $s$ and $t$ marked and being within a $D = 1$ distance of each other. Then, present a DFS ordering of that graph’s vertices in which $s$ is the first vertex visited but reaching $t$ by following that DFS ordering requires $\Omega(n)$ edge traversals. (Your example can be quite simple. Number the vertices by the order they are first visited in your DFS search.)
(b) [6 points] Consider now a strategy that corresponds to visiting the rooms in a BFS order. Show that such a strategy might require us to traverse $\Omega(B \cdot D)$ edges before finding the exit $t$, for any setting of $B \geq 2 \cdot D$.

To this end, for any given $D$ and $B \geq 2 \cdot D$, specify first an $n$-vertex $B$-bounded dungeon graph with $s$ and $t$ marked and being within a distance $D$ of each other. (Note that your construction needs to be parametrized by $D$ and $B$.) Then, present a BFS ordering of that graph’s vertices in which $s$ is the first vertex visited but reaching $t$ by following that BFS ordering requires traversing $\Omega(B \cdot D)$ edges.

*Hint:* Recall that you cannot “jump” between non-adjacent vertices during your exploration.
(c) [6 points] Propose a strategy that, for any $D$ and $B \geq 2 \cdot D$, enables us to always find an exit in a $B$-bounded dungeon graph after at most $O(B)$ edge traversals, provided we know the distance $D$ between $s$ and $t$ in advance. Don’t forget to provide an analysis that establishes the worst-case $O(B)$ edge traversals bound.