Quiz 2 Solutions

Problem 1. [1 points] Write your name on top of each page.

Problem 2. True/False [28 points] (14 parts)

Circle (T)rue or (F)alse. You don't need to justify your choice.

(a) T F [2 points] Computing $\lfloor \sqrt{a} \rfloor$ for an n-bit positive integer a can be done in $O(\lg n)$ iterations of Newton's method.

Solution: True. This is the bound obtained by Newton's Method's quadratic convergence.

(b) T F [2 points] Suppose we want to solve a polynomial equation $f(x) = 0$. While our choice of initial approximation x_0 will affect how quickly Newton's method converges, it will always converge eventually.

Solution: False. Take e.g. $f(x) = x^3 - 2x + 2$ and $x_0 = 0$. Then $x_{2i+1} = 1$ and $x_{2i} = 0$ for all i (that is, the approximations alternate between 0 and 1 without ever converging).

(c) T F [2 points] Karatsuba's integer multiplication algorithm always runs faster than the grade-school integer multiplication algorithm.

Solution: False. Problem Set 5 has shown that the $O(N^2)$ algorithm runs faster for small numbers.

(d) T F [2 points] If we convert an n-digit base-256 number into base 2, the resulting number of digits is $\Theta(n^2)$.

Solution: False. $\log_{256} n = \frac{\log_2 n}{\log_2 256} = \frac{\log_2 n}{8}$. By converting a base-256 number to base 2, the number of digits is multiplied by 8. For all $b_1, b_2 \neq 1$, converting a base-b_1 number to base-b_2 results in a linear increase or decrease in the number of digits.

(e) T F [2 points] In a weighted undirected graph $G = (V, E, w)$, breadth-first search from a vertex s finds single-source shortest paths from s (via parent pointers) in $O(V + E)$ time.

Solution: False. Only in unweighted graphs.

(f) T F [2 points] In a weighted undirected **tree** $G = (V, E, w)$, breadth-first search from a vertex s finds single-source shortest paths from s (via parent pointers) in $O(V + E)$ time.

Solution: True. In a tree, there is only one path between two vertices, and breadth-first search finds it.

(g) T F [2 points] In a weighted undirected **tree** $G = (V, E, w)$, **depth**-first search from a vertex s finds single-source shortest paths from s (via parent pointers) in $O(V + E)$ time.

Solution: True. In a tree, there is only one path between two vertices, and depth-first search finds it.

(h) T F [2 points] If a graph represents tasks and their interdependencies (i.e., an edge (u, v) indicates that u must happen before v happens), then the breadth-first search order of vertices is a valid order in which to tackle the tasks.

Solution: No, you'd prefer depth-first search, which can easily be used to produce a topological sort of the graph, which would correspond to a valid task order. BFS can produce incorrect results.

(i) T F [2 points] Dijkstra's shortest-path algorithm may relax an edge more than once in a graph with a cycle.

Solution: False. Dijkstra's algorithm *always* visits each node at most once; this is why it produces an incorrect result in the presence of negative-weight edges.

(j) T F [2 points] Given a weighted directed graph $G = (V, E, w)$ and a source $s \in V$, if G has a negative-weight cycle somewhere, then the Bellman-Ford algorithm will necessarily compute an incorrect result for some $\delta(s, v)$.

Solution: False. The negative-weight cycle has to be reachable from s.

(k) T F [2 points] In a weighted directed graph $G = (V, E, w)$ containing no zero- or positive-weight cycles, Bellman-Ford can find a *longest* (maximum-weight) path from vertex s to vertex t.

Solution: True. Negate the weights.

(l) T F [2 points] In a weighted directed graph $G = (V, E, w)$ containing a negative-weight cycle, running the Bellman-Ford algorithm from s will find a shortest acyclic path from s to a given destination vertex t.

Solution: False. Bellman-Ford will terminate, and can detect the presence of that negative-weight cycle, but it can't "route around it." (You could always remove an edge to break the cycle and try again, though.)

(m) T F [2 points] The bidirectional Dijkstra algorithm runs asymptotically faster than the Dijkstra algorithm.

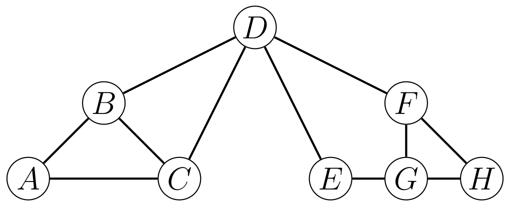
Solution: False. The constant factor behind bidirectional Dijkstra is better, but the worst-case running time is the same.

(n) T F [2 points] Given a weighted directed graph $G = (V, E, w)$ and a shortest path p from s to t, if we doubled the weight of every edge to produce $G' = (V, E, w')$, then p is also a shortest path in G'.

Solution: True. Multiplying edge weights by any positive constant factor preserves their relative order, as well as the relative order of any linear combination of the weights. All path weights are linear combinations of edge weights, so the relative order of path weights is preserved. This means that a shortest path in G will still be a shortest path in G'.

Problem 3. MazeCraft [26 points] (5 parts)

You are playing SnowStorm's new video game, *MazeCraft*. Realizing that you can convert a maze into a graph with vertices representing cells and edges representing passages, you want to use your newly learned graph-search algorithms to navigate the maze. Consider the following converted graph.



For the following questions, assume that the graph is represented using adjacency lists, and that **all adjacency lists are sorted**, i.e., the vertices in an adjacency list are always sorted alphabetically.

(a) [4 points] Suppose that you want to find a path from A to H. If you use breadth-first search, write down the resulting path as a sequence of vertices.

Solution: A, B, D, F, H

Scoring: For Problem 3(a), 3(b), and 3(e), we computed the score as $\lfloor \frac{P \cdot L}{\textsc{Max}(N_a, N_s)} \rfloor$, where P is the total points of the problem, L is the length of the longest common subsequence between our solution and the student's answer, N_a is the length of the student's answer, and N_s is the length of our solution.

(b) [4 points] If you use depth-first search to find a path from A to H, write down the resulting path as a sequence of vertices.

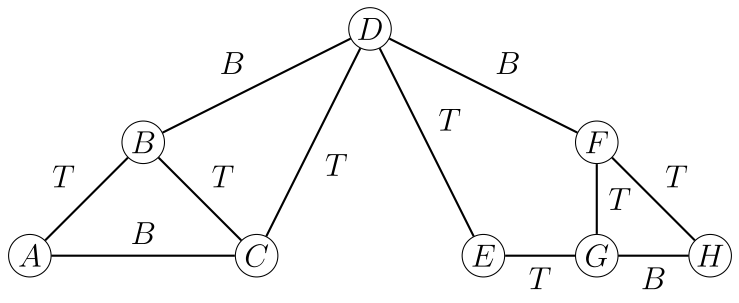
Solution: A, B, C, D, E, G, F, H if you use recursion, or A, C, D, F, H if you use stack-based implemention. Essentially, the recursion-based implementation will visit the neighbors in alphabetical order, while the stack-based implementation will visit the neighbors in reverse alphabetical order.

(c) [6 points] To determine whether the maze has cycles or multiple paths to the same destination, you decide to use the edge classification of depth-first search. Run depth-first search on the graph reproduced below, starting from vertex A, and label every edge with T if it's a tree edge, B if it's a back edge, F if it's a forward edge, and C if it's a cross edge.

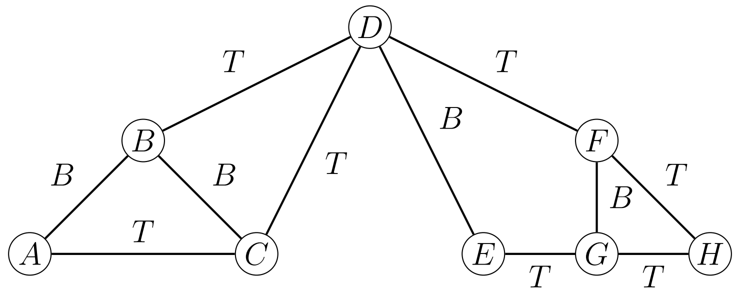
As a reminder, recall that an edge (u, v) is

- a ***tree edge*** (T) if v was first discovered by exploring edge (u, v) (tree edges form the depth-first forest);
- a ***back edge*** (B) if v is u's ancestor in a depth-first tree;
- a ***forward edge*** (F) if v is u's descendant in a depth-first tree; and
- a ***cross edge*** (C) if none of the above apply.

Solution: Recursion-based implementation:



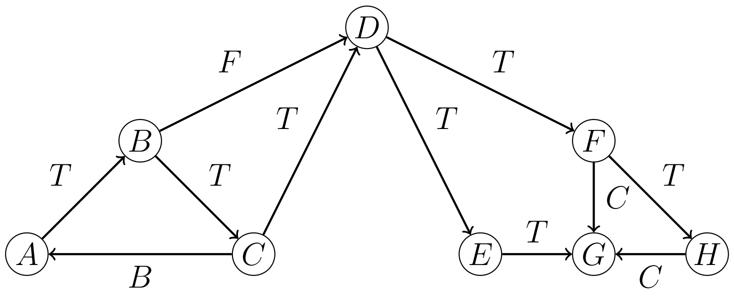
Stack-based implementation:



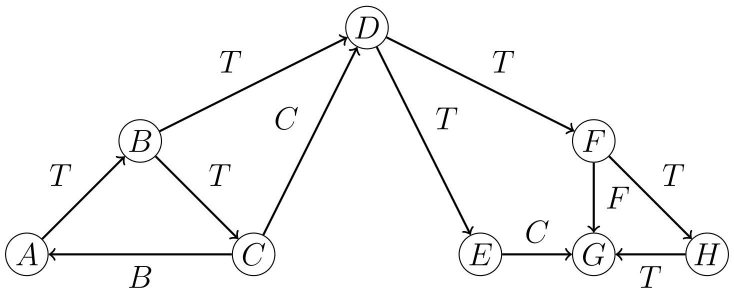
Scoring: For Problem 3(c) and 3(d), we computed the score as $\lfloor P - \frac{1}{2}N_w \rfloor$, where P is the total points of the problem and N_w is the number of wrong labels.

(d) [6 points] Now suppose that the passages in the maze are directional. Rerun depth-first search in the directed graph below and label the edges accordingly.

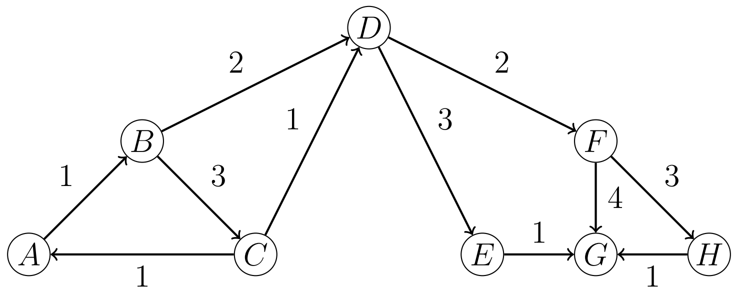
Solution: Recursion-based implementation:



Stack-based implementation:



(e) [6 points] Suppose each passage in the maze causes a different amount of ***damage*** to you in game. You change the graph to use weights to represent the damage caused by each edge. You then use Dijkstra's algorithm to find the path from A to H with the lowest possible damage. Write down the order in which vertices get removed from the priority queue when running Dijkstra's algorithm.



Solution: A, B, D, C, F, E, G, H

Problem 4. Malfunctioning Calculator [25 points] (5 parts)

Former 6.006 student Mallory Funke is at a math competition, but her calculator isn't working. It seems to work fine for whole numbers, but the numbers after the decimal point always seem to be the sequence 07734 repeated over and over again, making those digits useless. For one of the problems, she has been asked to compute $\lfloor A^{3/4} \rfloor$ for a few different integer values of A. Mal knows that Newton's Method can be used to compute the square root or the cube root of an integer A. So as a first step in computing $\lfloor A^{3/4} \rfloor$, Mal wants to use Newton's Method to compute $\lfloor A^{1/4} \rfloor$. She then plans to use that information to compute $\lfloor A^{3/4} \rfloor$.

(a) [5 points] Mal decides to use the function $f(x) = x^4 - A$, because it has a root at $x = A^{1/4}$. Use Newton's Method on $f(x)$ to generate a formula for computing increasingly accurate estimates of $\lfloor A^{1/4} \rfloor$. In other words, give a formula for the more accurate estimate x_{i+1} in terms of a less accurate estimate x_i. The formula you construct must use **only** addition, subtraction, multiplication, and division. (You do not need to simplify the formula.)

Solution: If we take the derivative of $f(x)$, we get $f'(x) = 4x^3$. Hence, Newton's Method says that we have

$$x_{i+1} = x_i - \frac{x_i^4 - A}{4x_i^3} = x_i - \frac{x_i \cdot x_i \cdot x_i \cdot x_i - A}{4 \cdot x_i \cdot x_i \cdot x_i}.$$

(b) [5 points] Mal decides to use the technique from part **(a)** to compute the value $B = \lfloor A^{1/4} \rfloor$. She then plans to compute $\lfloor A^{3/4} \rfloor$ by calculating the value $C = B^3 = B{\cdot}B{\cdot}B$. Provide an explanation of why this technique does not work.

Hint: Define α to be the fractional part of $A^{1/4}$, so that $B = A^{1/4} - \alpha$. What happens when you compute $C = B^3$?

Solution: When you expand out the formula for C, you get

$$C = B^3 = (A^{1/4} - \alpha)^3 = A^{3/4} - 3A^{2/4}\alpha + 3A^{1/4}\alpha^2 - \alpha^3.$$

If A is large, then $\gamma = 3A^{2/4}\alpha - 3A^{1/4}\alpha^2 + \alpha^3$ will be significantly greater than 1, and so we'll have $C = A^{3/4} - \gamma$ with $\gamma > 1$. Hence, C will not be $\lfloor A^{3/4} \rfloor$.

(c) [5 points] Mal clearly needs a way to check her answer for $\lfloor A^{3/4} \rfloor$, using only integers. Given a pair of positive integers A and C, explain how to check whether $C \le A^{3/4}$ using $O(1)$ additions, subtractions, multiplications, and comparisons.

Solution: The equation $C \le A^{3/4}$ is equivalent to the equation $C^4 \le A^3$, which is equivalent to $C \cdot C \cdot C \cdot C \le A \cdot A \cdot A$.

(d) [5 points] Explain how to check whether $C = \lfloor A^{3/4} \rfloor$, again using only $O(1)$ additions, subtractions, multiplications, and comparisons.

Hint: Recall how the floor function is defined.

Solution: If $C = \lfloor A^{3/4} \rfloor$, then by the definition of the floor function, we have $C \le A^{3/4}$ and $C + 1 > A^{3/4}$. We can check both of these using the technique from part **(c)**.

(e) [5 points] Give a brief description of an algorithm that takes as input a d-digit positive integer A and computes $\lfloor A^{3/4} \rfloor$. The only arithmetic operations you can use are integer addition, integer subtraction, integer multiplication, integer division, and integer comparison. Your algorithm should use $\Theta(\lg d)$ arithmetic operations in total, but partial credit will be awarded for using $\Theta(d)$ arithmetic operations. For this question, you may assume that Newton's Method has a quadratic rate of convergence for whatever function you devise.

Solution:

$\Theta(\lg d)$ Solution: The technique from part **(b)** didn't work because we threw away information (taking the floor) before performing another operation. Instead, we first calculate $B = A^3$, then compute $C = \lfloor B^{1/4} \rfloor$ using the formula from part **(a)** to continually improve our estimate. Because we have a quadratic rate of convergence, we will improve our estimate $\Theta(\log d)$ times, and each time we will perform a constant number of arithmetic operations.
Note: This is equivalent to applying Newton's Method to the function $f(x) = x^4 - A^3$, but does not require you to rederive the equation for x_{i+1} in terms of x_i.

$\Theta(d)$ Solution: Use the comparison method from part **(c)** to do binary search. This requires us to do $\Theta(1)$ operations for each of the bits in the result, for a total of $\Theta(d)$ operations.

Problem 5. The Tourist [15 points]

Your new startup, *Bird Tours*, brings people around Boston in a new aviation car that can both drive and fly. You've constructed a weighted directed graph $G = (V, E, w)$ representing the best time to drive or fly between various city sites (represented by vertices in V). You've also written a history of Boston, which would be best described by visiting sites $v_0, v_1, \ldots, v_k$ in that order.

Your goal is to find the shortest path in G that visits $v_0, v_1, \ldots, v_k$ in order, possibly visiting other vertices in between. (The path must have $v_0, v_1, \ldots, v_k$ as a *subsequence*; the path is allowed to visit a vertex more than once. For example, $v_0, v_2, v_1, v_2, \ldots, v_k$ is legal.) To do the computation, you've found an online service, Paths 'Я Us, that will compute the shortest path from a given source s to a given target t in a given weighted graph, for the bargain price of \$1. You see how to solve the problem by paying \$$k$, calling Paths 'Я Us with $(v_0, v_1, G), (v_1, v_2, G), \ldots, (v_{k-1}, v_k, G)$ and piecing together the paths. Describe how to solve the problem with only \$1 by calling Paths 'Я Us with (s, t, G') for a newly constructed graph $G' = (V', E', w')$, and converting the resulting path into a path in G.

Solution: To form the graph G', we start from the disjoint union of k copies of the given graph G, using superscripts to denote the different copies $G^1, G^2, \ldots, G^k$. That is, for each vertex $v \in V$, we make k copies $v^1, v^2, \ldots, v^k$ in V'; and for each edge $(u, v) \in E$, we form the edges $(u^1, v^1), (u^2, v^2), \ldots, (u^k, v^k)$ in E', each of weight $w(u, v)$. Then we add the edges (v_1^1, v_1^2), (v_2^2, v_2^3), $\ldots$, $(v_{k-1}^{k-1}, v_{k-1}^k)$, each of weight 0, to complete the graph G'. Finally we set the start vertex $s = v_0^1$ and the target vertex $t = v_k^k$, and call Paths 'Я Us with (s, t, G').

Intuitively, the subgraph G^i represents a situation in which the path has visited the Boston sites $v_0, v_1, \ldots, v_{i-1}$, and is currently aiming to visit v_i. Once the path visits v_i, it can take the zero-weight edge (v_i^i, v_i^{i+1}), which represents the fact that v_i has been visited. (A path may choose not to follow this edge, but it must do so eventually in order to reach the next copy G^{i+1} and eventually the target vertex t in G^k.)

Once we have a shortest path in G', we can convert it into the corresponding path in G by simply replacing every vertex $u^i \in V'$ with $u \in V$, and then removing any loop edges (v_i, v_i) from the path (resulting from the zero-weight edge (v_i^i, v_i^{i+1})). Intuitively, we just need to flatten the k copies of G down to the single copy.

Another solution to this problem, found by many students, is to replace the endpoints of the edges instead of using zero-weight edges. For example, after making k copies of the graph G, we can replace each edge (u^i, v_i^i) with (u^i, v_i^{i+1}). This has the same effect as the zero-weight edges, and removes the need to remove loops at the end.

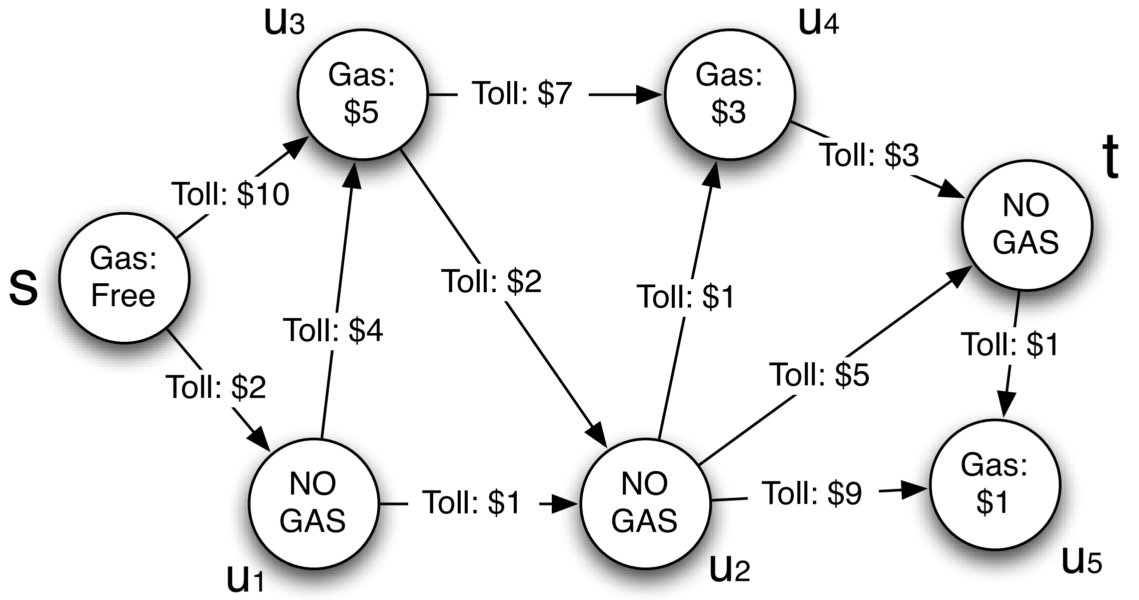
Problem 6. Fill 'Er Up! [25 points]

You are traveling by car from one city to another city. Unfortunately, you have a hole in your gas tank, and you have to refill your gas tank to travel across more than two roads. In addition, there is a toll booth on every road that charges you for using that road. Your goal is to find the least-expensive path from your start to your destination.

You represent the city network using a directed graph $G = (V, E, w)$ with weights w defined on both edges and vertices. The vertices V represent the cities and the edges E represent the roads. The weight $w(e)$ of an edge e represents the toll amount on that road. The weight $w(v)$ of a vertex v is the price of filling your gas tank in that city (which is a fixed price independent of how much gas you have left, or ∞ if there is no gas available to purchase). You are allowed (but not obligated) to end your journey with an empty tank, and you may assume that you always start your journey with a full tank.

Below is an example graph that we will use to answer part (a). One seemingly cheap path from s to t is (s, u_1, u_2, t) at a cost of \$8. Unfortunately, this path is not valid, because our leaky gas tank won't permit moving across three edges without refilling our gas tank.

One valid path is (s, u_3, u_2, t) at a cost of \$22. (This is a valid path: we begin with a full tank, travel across one edge to a gas station where we refill our tank, and then travel two edges to the destination, arriving with an empty gas tank. Notice that we are unable to continue the journey to u_5 if we wanted to, because even though there is a gas station there, we have to traverse a third edge to get to it.)



There are some extra copies of this graph at the end of the exam (for your convenience).

(a) [5 points] The valid path given in the description above is not the *best* path. Find a least-expensive path from s to t in the graph above.

Solution: (s, u_1, u_3, u_2, t)

(b) [5 points] Find the least-expensive path from s to u_5 in the graph above.

Solution: $(s, u_1, u_3, u_2, u_4, t, u_5)$

(c) [15 points] Give an $O(V \log V + E)$ algorithm to find, in a given graph $G = (V, E, w)$, a least-expensive *valid* path from a city s to a city t, or report that no such path exists.

Solution: We will solve this problem by transforming $G = (V, E, w)$, the graph of cities, roads, and toll/gas costs, into a new graph $G' = (V', E', w')$. We will then use Dijkstra's algorithm to find the desired path.

First construct the vertex set V'. For each vertex $v \in V$, create three vertices in V': v_f, v_h, and v_e. These stand for a **f**ull tank, a **h**alf-full tank, and an **e**mpty tank, respectively. This operation takes $O(V)$ time, since we iterate $|V|$ times and create three new vertices each time.

Next we will discuss the construction of E' and w'. For each edge $(u, v) = e \in E$, create edges in E' as follows. Create edges (u_f, v_h) and (u_h, v_e) to represent the emptying of half the tank during the move from u to v. The weights on both of these edges will be the original weight $w(u, v)$. This operation takes $O(E)$ time, since we iterate $|E|$ times and insert two new edges into E' each time.

Finally, we will represent the cost of fueling up by creating additional edges as follows. For every vertex $v \in V$, we check whether there is a gas station there. If there is a gas station at this vertex, then we add the edges (v_e, v_f) and (v_h, v_f) to E'. Since the cost of fueling up is not dependent on the amount of gas that is already in the tank, we assign both of these edges the cost of fueling up at that city. This procedure takes $O(V)$ time, since we iterate over all of the vertices and create a constant number of new edges.

We can now run Dijkstra's algorithm on this graph to find paths from s_f to t_f, t_h, and t_e, and we return the shortest of the three. The graph transformation took $O(V + E)$ time and, as the number of vertices and the number of edges increased by a constant factor, running Dijkstra on this graph takes $O(V \log V + E)$ time. Therefore, the total running time is $O(V \log V + E)$.