Midterm #2 Solutions

- Answer all questions.
- Read them carefully first.
- Be precise and concise.
- Justify correctness of your answers (except in the T/F questions).
- Write in the space provided.
- Good luck!

1 Lost in Translation (20 pts)

Consider the following encoding of the letters $a, \ldots, z$ and $A, \ldots, Z$ by strings of decimal digits.

- a is encoded by 1, b is encoded by $2, \ldots, z$ is encoded by 26
- A is encoded by 101, B is encoded by $102, \ldots, Z$ is encoded by 126

Given a number string, such as 1872311, you want to return the number of ways that you can represent this number string as a string composed of letters. For example, an input of the number string 121 should return 4 since there are four ways to write this as a string namely, $'aba'(1,2,1)$, $'la'(12,1)$, $'au'(1,21)$, and $'U'(121)$.

- (15 pts) Give an $O(n)$ time algorithm to find the number of representations of a number string where n is the length of the input number string.

  Solution: We use dynamic programing. Let the number string be $s[1], \ldots, s[n]$. To count the number of representations of length i, we need to consider 3 scenarios depending if the last letter is of length 1, length 2 or length 3. We use the following dynamic programing where the $i'th$ subproblem $N[i]$ is the number of representations of the string $s[1] \ldots s[i]$. Initially, we set $N[i] = 0$. First, we check if $s[i]$ is between one and nine. If so, we let $N[i] += N[i-1]$. If $s[i-1]s[i]$ consists of a legal encoding of a single letter (lower, k and higher) we let $N[i] += N[i-2]$ and if $s[i-2]s[i-1]s[i]$ consist of a legal encoding of a single capital letter, we let $N[i] += N[i-3]$. We see that we have considered all the possibilities (depending on the encoding of the last letter).

  Assuming $n \geq 3$ we obtain the following pseudo-code:

  - $N[0] = 1, N[1] = 1$.
  - If $s[1]s[2]$ is a letter, let $N[2] = 2$. Otherwise, let $N[2] = 1$.
  - For $i = 3$ to $i = n$ do: First, let $N[i] = 0$
    - If $s[i]$ is between 1 and 9, let $N[i] += N[i-1]$
    - If $s[i-1]s[i]$ represents one of the letters 'k' to 'z', let $N[i] += N[i-2]$.
    - If $s[i-2]s[i-1]s[i]$ represents one of the letters 'A' to 'Z', let $N[i] += N[i-3]$

  The main loop consist of $O(1)$ operations and is run for $O(n)$ steps. Therefore the running time is $O(n)$.

  Remarks: One common mistake was to initialize $N[i] = 1$ in the for loop. Some students took the max or product instead of sum.

- (5 pts) For every $n \geq 1$, find a number string of length n that has a unique representation as a string of composed letters.

  Solution: Look at the sequence $3 \dots 3$ of length n. The only letter starting by 3 is c. Therefore the unique decoding of this sequence is $c \dots c$ of length n.

2 I need some space! (25 pts)

It's midterm time again, and there are K students that are going to take the test. The room that we've reserved is a bit strange. There is a single row of N desk spaced irregularly in a straight line (with $N \geq K$). Desk i has integer coordinate x_i with $0 = x_1 < x_2 \ldots < x_N = L$, where L is the length of the room (which is also integer). We want to maximize the personal space of students. We wish to seat the students at the desks such that the minimum distance between any two students is maximized. For example, if there are $K = 3$ students and $N = 5$ tables located at $x_1 = 0, x_2 = 6, x_3 = 7, x_4 = 9, x_5 = 10$, then the students will be seated at tables $x_1 = 0, x_2 = 6$ and $x_5 = 10$, so that the distance between any two students is at least 4. In this example there is no other way of sitting the 3 students at the desks with minimum distance less than 4.

- (5 pts) Show that for any input there exists an optimal solution where the left most desk $x_1 = 0$ is occupied.

  Solution: Consider an optimal solution. If $x_1 = 0$ is occupied, there is nothing to prove. Otherwise, let x_i be table with the smallest coordinate that is occupied. Consider a solution where we move the student seated at x_i to the table $x_1 = 0$. This increases the distance between every pair of students and therefore it is also an optimal solution.

  Remarks: To do this by contradiction, we first assume that all optimal solutions have x_1 unoccupied. However, some students said to first assume one optimal solution has x_1 unoccupied. Additionally, some students were using "average" distance between desks to argue the case. Remember, even though this statement may be intuitively true, we are still expecting you to formally show this.

- (10 pts) For a given integer distance D, give an $O(N)$ algorithm that returns a seating assignment of the students where the minimum distance between two students is at least D if such a seating assignment exists, and indicates if it is impossible otherwise (you may use the previous item even if you didn't solved it).

  Solution We use the following Gready Algorithm:

  - Set $t = 1$.
  - For $s = 1$ to K
    - If $t > N$ return FAIL
    - $Seat[s] = x[t]$.
    - While $(x[t] - Seat[s] < D$ and $t \leq N)$, $t := t + 1$.

  If the algorithm doesn't FAIL it returns K seats $Seat[1], \ldots, Seat[K]$ each pair differ by at least D. By the same argument in the first part of the question, if there exists a solution, then there must exist a solution with $Seat[1] = x[t_1] = 0$. By induction, there is a solution where a student is seated at the first table t_2 $x[t_2] > D$, in the first table t_3 where $x[t_3] - x[t_2] > D$ etc. This is exactly what the algorithm is doing.

  To analyze the running time, we note that it is dominated by the number of times t is increased which is $O(N)$.

  Remarks: Most students got this problem right. Just remember to prove correctness, even if it's one line.

- (10 pts) Devise an algorithm that will run in $O(N \log L)$ time that finds a seating assignment that maximizes the minimum distance between any two students (you may used the previous two items even if you didn't solved them).

  Solution: Perform binary search over the value of the distance D. Start by checking if there is a seating with distance $\lceil L/2 \rceil$. If the answer is no, try $\lceil L/4 \rceil$. If the answer is yes, try $\lceil 3L/4 \rceil$. Each check is done by the procedure in the previous item. The correctness of the algorithm follow from the fact that if there is a seating with distance $d_1 > d_2$ then it is also a seating for distance d_2. The number of iterations is bounded by $O(\log L)$ so the total complexity is $O(N \log L)$ as needed.

  Remarks: If your algorithm doesn't match the runtime stated, don't say it's $O(N \log L)$. Writing more can sometimes make you lose points. Also, similarly to the previous part, be sure to briefly mention correctness, even if it's a short sentence. A common mistake for correctness was proving that binary search was correct, rather than proving that the overall algorithm is correct.

3 The Short Way Out (35 pts)

Consider an undirected road graph representing the n cities of the Netherlands and the bike paths between them. The graph is **undirected** with **positive edge weights**. A young dutch boy is asked by his PE teacher to practice biking on this road graph.

- (15 pts) On the first day, the boy is asked by the PE instructor to bike between a pair of cities of his choice, one on the east side of the Netherlands and one on the west side of the Netherlands. Suppose that the cities on the east side are $u_1, \ldots, u_k$ and the cities on the west side are $v_1, \ldots, v_\ell$ (with $k + \ell \leq n$). Design an algorithm to help the young boy to find the shortest bike path among all the possible paths starting at one of the eastern cities u_i and ending at one of the western cities v_j. You may additionally assume that $u_i \neq v_j$ for all $1 \leq i \leq k, 1 \leq j \leq \ell$. The running time of the algorithm should be of the same order as the running time of Dijkstra's algorithm (even for large values of k and ℓ such as $k = \ell = n^{1/2}$).

  Solution:

  Let $U = \{u_1, \ldots, u_k\}$, and $V = \{v_1, \ldots, v_\ell\}$

  Add node u with zero weight edges to every node in U and a node v with zero weight arcs from every node in V and use Dijkstra's to compute the u-v distance. Running time is the same as that of Dijkstra's.

  Some other methods of doing the same thing:

  1. Collapse the vertices in U to a single metanode and do the same for V, and run Dijkstra's.
  2. Collapse the vertices in U to a single metanode, run Dijkstra's, and stop as soon as you call `delete-min` on a vertex in V.
  3. Run Dijkstra's, but initialize the distances of all nodes in U to 0.

  We can see all these methods will run Dijkstra on a graph with the same order of vertices and edges, thus will be on the same order as running Dijkstra as G.

- (15 pts) On the second day, the young dutch boy is asked to bike between two specific cities u and v chosen by the teacher and then back from v to u. He may choose to traverse any edge any number of times. However, the PE teacher asks that the student chooses one edge (w, w') along the path from u to v to be traversed 3 times: upon arriving to w the student is to bike to w', then from w' to w and finally from w to w' before continuing on to v. Design an algorithm to help the young boy find paths from u to v and from v to u as well as the edge (w, w') that will minimize his total bike trip. The running time of the algorithm should be of the same order as the running time of Dijkstra's algorithm.

  Solution: To find the path from v to u we may just use Dijkstra. Therefore the goal is to find the path from u to v and the edge (w, w') repeated 3 times. To do so we define a new directed graph $G' = (V', E')$. $V' = V \times \{0, 1\}$. To each edge $(u', v') \in E$ of length x we generate 6 edges in G':

  - $(u', 0) \to (v', 0)$ and $(v', 0) \to (u', 0)$ of length x.
  - $(u', 1) \to (v', 1)$ and $(v', 1) \to (u', 1)$ of length x.
  - $(u', 0) \to (v', 1)$ and $(v', 0) \to (u', 1)$ of length $3x$

  Now a path from $(u, 0)$ to $(v, 1)$ in G' has exactly one edge that goes from the bottom layer of the graph to the top layer and that edge has length that is triple the length in the original graph. Thus the shortest path from $(u, 0)$ to $(v, 1)$ in G' corresponds to exactly the required path from u to v (where we ignore the second coordinate of each vertex). The required edge (w, w') is the unique edge of the form $(w, 0) \to (w', 1)$.

  To analyze the running time of the algorithm, we note that G' has at most 2 times the vertices and at most 6 times the number of edges that G has. Thus running Dijkstra on G' is of the same complexity of running Dijkstra as G. Moreover generating G' from G takes linear time in the size of G.

  A different solution (sketch): Run Dijkstra from u and run Dijsktra from v. For each edge (w, w') let

  $$d(w, w') = 3\ell(w, w') + \min(d(u, w) + d(v, w'), d(u, w') + d(v, w)),$$

  where $\ell(w, w')$ is the length of the edge from w to w'. Note that $d(w, w')$ is the length of the shortest past from v to u that uses the edge (w, w') three time. Therefore taking the minimum over (w, w') will result in the shortest desired path. The complexity is running Dijkstra twice and then taking the minimum over $|E|$ edges which is of smaller order.

  A wrong solution: You can try to modify Dijkstra to include the extra information. However, you have to verify that when a node is popped it will never be updated again. For most modifications we saw this wasn't the case.

- (5pt) Give an example of a graph where on the second day the shortest trip must contain an edge that is traversed on the path from u to v that is not part of any shortest path from v to u.

  Solution: Consider the graph with $V = \{u, v, w\}$ and the edge (u, v) of length 5 and the edges (u, w) of length 3 and (w, v) of length 4. The unique shortest path between u and v is (u, v) of length 5. However if we need to repeat an edge 3 times, then the shortest path is to repeat (u, w) three times followed by (w, v) (of length 13 compared to 15 by using (u, v) three times).

4 Binary Answers (T/F) (20 pts)

No need to justify.

1. (2pts) The path compression optimization reduces the big-O time complexity of Kruskal's algorithm.
   Sol: True. We've seen that it reduces it from $O(m \log n)$ to $O(m \log^* n)$ where m is the number of edges and n is the number of vertices.

2. (2pts) In any connected undirected graph, the uniquely heaviest edge if it exists (edge with weight strictly greater than any other edge weight in graph) cannot be part of any minimum spanning tree of the graph.
   Sol: False. Consider a graph consisting of a single edge - then this edge is uniquely heaviest and in any MST.

3. (2pts) Any directed graph $G = (V, E)$, can be made strongly connected by adding at most $|V|$ edges.
   Sol: True Write the vertices in some order $v_1, \ldots, v_n$ and add a directed edge between v_i and v_{i+1} as well as between v_n and v_1. If some of these edges exist, there is no need to add them.

4. (2pts) Suppose we knew a graph only had exactly k negative edges and no negative cycles. Then, we can repeat Bellman-Ford's inner loop only k times and get the correct shortest path distances between all vertices.
   Sol: False. A counter example is a path of length $n > k$ with all edge weights 1.

5. (2pts) Any full binary tree represents the Huffman code corresponding to some set of frequencies.
   Sol: True. It was discussed in class.

6. (2pts) The edit distance between the string 123456789 and the string 987654321 is 9.
Sol: False. The distance is 8. It is easy to see that it is at most 8 by matching the 5 and making 8 substitutions.

7. (2pts) The horn formula
$$x \implies y,\ y \implies z,\ z \implies x,\ \bar{y}$$
is satisfiable.
Sol: True. Set all variables to FALSE.

8. (2pts) For an undirected graph with edge weights that are all either 1 or 2, let $b(n)$ be the running time of computing all distances from s using BFS and $d(n)$ be the running time of computing all distances from s using Dijkstra with binary heaps. Then $b(n) = \Theta(d(n))$.
Sol: FALSE. BFS takes linear time in this case. Dikjstra's algorithm takes more than linear time.

9. (2pts) The node that receives the lowest pre number in DFS must lie in a source strongly connected component.
Sol: False. Consider the graph $a \to b$ and start the DFS at b.

10. (2pts) Any directed graph on $n \geq 3$ vertices in which every node has at least one outgoing edge has at least one directed cycle.
Sol: True. Follow the outgoing edges until you hit the same node twice.