COMP 285 (NC A&T, Fall '22) Homework 8 (100 pt)

Fun with Dynamic Programming, Exhaustive Search + Backtracking and More

Due 12/01/22 @ 11:59PM ET

Submitting

In order to submit this assignment, you must download your assignment as a ZIP file and upload it to Gradescope. There will be a written component and a coding component.

For the written component, you will write your answers in the corresponding `.txt` files.

For the coding component, you will write your solution in `answers.cpp`. We will use an autograder to test your solutions - the results of which can be seen after submitting to Gradescope. Note that you have unlimited tries to submit. In order to receive full credit, you must also provide documentation on your approach, see the commented sections in the `answers.cpp` file.

Question 0: Feedback is a Gift (pt. 2)! (Optional/Extra Credit)

We are nearing the end of the semester! We'd love to collect feedback on COMP 285 so far so that we can adapt, grow, and better the course.

Your inputs to this survey will not influence your grades in the class in any way. The survey data will only be used in aggregated form without identification of individuals.

You can fill out the survey here.

The survey closes on Wednesday, November 30th. If >=80% of the class fills out the survey, we will add +1% to everyone's final grade.

Question 1: Gift Card Balance (20 pts)

You received a gift card to your favorite store for your birthday - score! You want to spend down the entire gift card on your next trip to the store. Based on the items available in the store, let's write a program that tells us whether we can use up our gift card in full. Assume there is only one copy of each item.

Part a. (15 pt) Write your function that takes in a `vector<int>` of item prices and an integer called `giftCardBalance` and returns a `bool` representing whether or not we can spend down the entire gift card.

```
Input: items={2, 3, 3, 7}, giftCardBalance = 6
Output: true // We can buy items with prices 3+3 = 6

Input: items={4, 6}, giftCardBalance = 7
Output: false // We cannot buy items with prices that sum to 7.
```

Write your solution in `canSpendGiftCardHelper` in `answers.cpp`

Part b. (5 pt) What is the optimal, tightest Big-Oh runtime of this algorithm?

Write your answer in `q1.txt`

Question 2: Bellman-Ford Analysis (20 pts)

Unlike Dijkstra's, Bellman-Ford uses Dynamic Programming to find the Single-Source Shortest Path (SSSP) even if there are negative edge weights. Given the following pseudocode for Bellman-Ford, answer the questions below.

```
algorithm Bellman-Ford(G, s)
 Input:
   Graph G = (V, E) with weights w(u, v) for each (u, v) in E
   Starting node s
 Output:
   array dist, where dist[u] is the shortest paths from s to u
   array prev, where prev[u] is the node before u along the shortest path

0   for each vertex v in G
1     dist[v] = inf
2     prev[v] = null
3   dist[s] = 0
4
5   Repeat |V| - 1 times
6     for each edge (u, v) in E
7       temp = dist[u] + w(u, v)
8       if temp < dist[v]
9         dist[v] = temp
10        prev[v] = u
11
12  for each edge (u, v) in G
13   if dist[u] + w(u, v) < dist[v]
14     throw error("Negative Cycle Exists")
15
16  return dist, prev
```

Part a. (5 pt) Explain in a few sentences why we have to relax all edges |V| - 1 times (i.e. where does |V| - 1 come from in line 5?). Then, explain how we could make this algorithm more efficient if we were only interested in paths from s to u that were at most T edges long.

Part b. (5 pt) Explain why the last for loop in the pseudocode can detect a negative cycle in the graph. Recall that a negative cycle is a cycle within the graph whose net total weight is negative.

Part c. (5 pt) Based on the pseudocode, what is the time complexity in terms of `V` and/or `E`?

Part d. (5 pt) Based on the pseudocode, what is the space complexity in terms of `V` and/or `E`?

Question 3: Floyd-Warshall Analysis (20 pts)

Floyd-Warshall is an algorithm to efficiently find the shortest paths between every single pair of vertices in a graph. Instead of solving the Single-Source Shortest Path (SSSP) problem like Dijkstra and Bellman-Ford, this approach solves the All-Pairs Shortest Path (APSP) problem.

Use the dynamic programming outline for Floyd-Warshall to answer the questions below.

Floyd-Warshall Dynamic Programming Outline: 1. **Define Recursive Subproblem** - `F[k][u][v]` = the weight of the best path from vertex u to vertex v with access of any of u, v, and the vertices labeled 1...k along the way - Note: we label each vertex with 1, 2, 3 ... n before we start working with the graph.

2. **Relating Subproblems**

- Base cases: `F[0][u][v]` = edge weight from u to v (if it exists, otherwise `inf`) - Otherwise `F[k][u][v] = min(F[k-1][u][v], F[k-1][u][k] + F[k-1][k][v])`

3. **Order of Solving Bottom-Up**

- For each `k = 1...n`, create a new adjacency matrix and solve for every `u-v` pair

4. **Solve Original Problem**

- Return 2-d F matrix at `k = n` (i.e. `F[n]`) as final answer

Part a. (5 pt) Describe in a few sentences why the 2D matrix at `k = n` (i.e. `F[n]`) returned in step 4 is the solution to APSP by referencing step 1.

Write your answer in `q3.txt`

Part b. (5 pt) Describe in a few sentences what the two choices we are taking the minimum over in step 2 are. Conceptually, what does each term represent?

Write your answer in `q3.txt`

Part c. (10 pt) How many subproblems does Floyd-Warshall have? What is the time per subproblem? Use this to calculate the time complexity.

Write your answer in `q3.txt`

Question 4: Call Me Maybe! (20 pts)

For a given string that contains only digits from `2-9` (inclusive), write a function that returns all possible letter combinations that the number could represent and return them in alphabetical order.

A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

```
 Input: digits = "23"
Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]

Input: digits = ""
Output: []

Input: digits = "8"
Output: ["t","u","v"]

Input: digits = "456"
Output: ["gjm","gjn","gjo","gkm","gkn","gko","glm","gln","glo","hjm","hjn","hjo","hkm","hkn","hko","hl
```

Write your solution in `answers.cpp`

Question 5: Find All Permutations (20 pts)

Mathematically, a permutation of a vector is an arrangement or rearrangement of its elements into a sequence. For example, for the vector `{1, 2, 3}`, valid permutations include `{2, 3, 1}` and `{1, 3, 2}`; of course, `{1, 2, 3}` itself is also a valid permutation.

You may know from Discrete Math that there are `3! = 6` permutations for the vector {1, 2, 3}, which are, in lexicographic order,

```
 {1, 2, 3}
{1, 3, 2}
{2, 1, 3}
{2, 3, 1}
{3, 1, 2}
{3, 2, 1}
```

Implement the function `findAllPermutations` which takes a parameter N and returns all permutations of the vector {1, 2, ..., N} in lexicographic order.

For the purpose of this exercise, you **may not** use `std::next_permutation` or `std::prev_permutation` in `<algorithm>`. Instead, you must use the exhaustive search strategies we covered in lecture (with a helper which chooses, explores, then unchooses).

Write your solution in `answers.cpp`