Instructor Name: Siddhartha Nath
Email: s.nath@northeastern.edu

Northeastern University

Program Structure and Algorithms (INFO 6205)
Homework #4 – 100 points

Student NAME: Xijing Zhang

Student ID: 002205911

Question 1 (15 points). *Please provide an example of a weighted, directed graph $G = (V; E)$ which has some edge weights negative, and Dijkstra's algorithm correctly finds the shortest paths from a source s in the graph. You can assume the graph has no negative cycles. Please clearly indicate the source s in your graph and which edge weights are negative.*

Consider the following directed graph $G = (V, E)$:
Vertices: $V = \{A, B, C, D\}$
Edges and weights:

- $A \to B = 1$
- $A \to C = 4$
- $B \to C = -2$ (Negative edge)
- $C \to D = 2$

Let the source node be $s = A$. This graph contains one negative edge $B \to C = -2$, but no negative cycles.
We apply Dijkstra's algorithm starting from node A:

1. Initialization:
$$\text{dist}(A) = 0, \quad \text{dist}(B) = \infty, \quad \text{dist}(C) = \infty, \quad \text{dist}(D) = \infty$$
2. From A:
$$\text{dist}(B) = 0 + 1 = 1, \quad \text{dist}(C) = 0 + 4 = 4$$
3. Visit B (shortest known distance = 1):
$$\text{dist}(C) = \min(4, 1 + (-2)) = -1$$
4. Visit C (shortest known distance = -1):
$$\text{dist}(D) = -1 + 2 = 1$$

5. All nodes visited.

The final shortest path distances from source A are:

$$\text{dist}(A) = 0, \quad \text{dist}(B) = 1, \quad \text{dist}(C) = -1, \quad \text{dist}(D) = 1$$

Question 2 (20 points)**.** *There is a network of roads $G = (V; E)$ connecting a set of cities V. Each road in E has an associated length l_e. There is a proposal to add* **one** *new road to this network, and there is a list $E^{'}$ of pairs of cities between which the new road can be built. Each such potential road $e^{'} \in E^{'}$ has an associated length. As a designer for the public works department you are asked to determine the road $e^{'} \in E^{'}$ whose addition to the existing network G would result in the maximum decrease in the driving distance between two fixed cities s and t in the network.*

(i) (15 points) Describe an efficient algorithm by using Dijkstra's algorithm in English for solving this problem.

(ii) (5 points) Please explain the running time of your algorithm.

(i) To determine which proposed road $e' \in E'$ should be added to maximize the decrease in driving distance from city s to city t, we can use the following algorithm:

1. Run Dijkstra's algorithm from the source city s to compute the shortest distances from s to all other cities. Let this be $\text{dist}_s[v]$ for each $v \in V$.

2. Run Dijkstra's algorithm from the destination city t to compute the shortest distances to all other cities (i.e., distances from every city to t). Let this be $\text{dist}_t[v]$ for each $v \in V$.

3. Let $D_{\text{original}} = \text{dist}_s[t]$, the original shortest distance from s to t.

4. For each candidate road $e' = (u, v) \in E'$ with length $l_{e'}$, calculate the new potential shortest path from s to t if e' is added. There are two cases to consider:

$$D_{\text{new}} = \min(\text{dist}_s[u] + l_{e'} + \text{dist}_t[v],\ \text{dist}_s[v] + l_{e'} + \text{dist}_t[u])$$

5. Compute the improvement in distance: $\Delta = D_{\text{original}} - D_{\text{new}}$

6. Select the edge $e' \in E'$ that gives the maximum Δ

This approach ensures we use Dijkstra's algorithm only twice (once from s and once from t), and then efficiently evaluate each candidate road.

(ii) Let $|V| = n$, $|E| = m$, and $|E'| = k$.

- Running Dijkstra's algorithm from s: $O(m + n \log n)$
- Running Dijkstra's algorithm from t: $O(m + n \log n)$
- For each candidate edge in E', computing the potential new path takes $O(1)$, and there are k such edges: $O(k)$

Therefore, the total running time is:

$$O(m + n \log n + k)$$

Question 3 (25 points). *Suppose you are given an infinite supply of coins whose values are one of* $1¢, 5¢, 10¢$ *and a dollar value* N*, which is a positive integer.*

(i) (10 points) Please describe an efficient greedy algorithm in English to make change for $N¢$ *using the three denominations of coins.*

(ii) (2 points) What is the running time of your algorithm?

(iii) (5 points) Please describe the order of coins and their denominations your algorithm will use when $N = 13¢$*?*

(iv) (8 points) Suppose you are not given $5¢$ *but instead given* $6¢$ *denomination. Please explain if your algorithm will still work correctly. Why?*

(i) The greedy algorithm works by always choosing the largest possible coin that does not exceed the remaining amount. The steps are:

1. Start with the largest coin (10¢), then 5¢, then 1¢.
2. For each coin, use as many of that coin as possible without exceeding the remaining amount.
3. Subtract the total value of those coins from the remaining amount.
4. Repeat the process with the next smaller denomination.
5. Continue until the remaining amount is 0.

This algorithm works correctly for the 1¢, 5¢, and 10¢ denominations, since they are canonical and greedy always produces an optimal solution.

(ii) The running time is constant, $O(1)$, because the algorithm performs a fixed number of operations (division and subtraction) for a fixed number of denominations (3 coins), regardless of the value of N.

(iii) Applying the greedy algorithm:

- Use one 10¢ coin: $13 - 10 = 3$¢ remaining.
- Use zero 5¢ coins: $3 < 5$.
- Use three 1¢ coins: $3 - 3 = 0$¢ remaining.

Result: 1 coin of 10¢, 0 coins of 5¢, and 3 coins of 1¢.

(iv) The greedy algorithm does not always work correctly when using the 1¢, 6¢, and 10¢ denominations. Here is a counterexample:

Suppose $N = 12$¢.

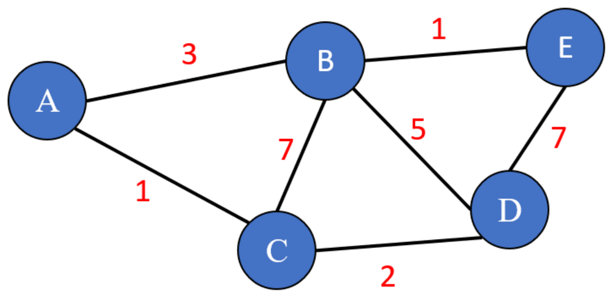
- The greedy algorithm first takes one 10¢ coin: 2¢ remaining.
- Cannot use 6¢ (too large), so use two 1¢ coins.
- Total coins used: 1 (10¢) + 2 (1¢) = 3 coins.

However, the optimal solution is:

- Two 6¢ coins = 12¢ using only 2 coins.

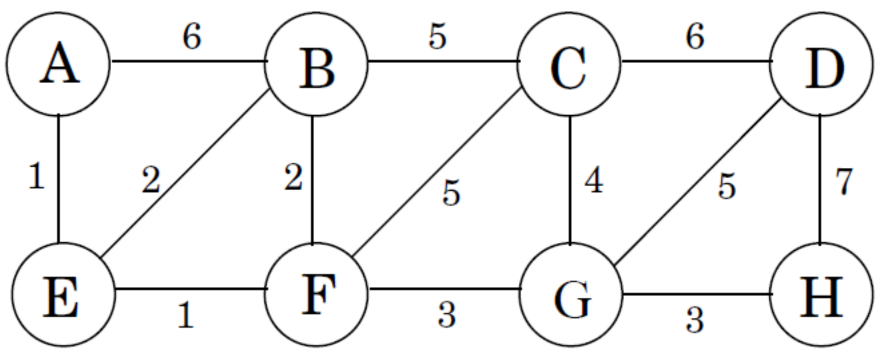
Therefore, the greedy algorithm does not always give the minimum number of coins when using the 6¢ denomination.

Question 4 (20 points). *Please execute Dijkstra's algorithm from vertex A in the following graph and fill the table below. d(·) denotes the shortest distance from A to the vertex. The column "PQ" should only list the vertices (in order) in the Priority Queue whose distances ≠ ∞. Break all ties lexicographically (i.e, according to alphabetical order).*



Iter	**PQ**	$d(A)$	$d(B)$	$d(C)$	$d(D)$	$d(E)$
1	*[A]*	*0*	∞	∞	∞	∞
2	*[C, B]*	*0*	*3*	*1*	∞	∞
3	*[B, D]*	*0*	*3*	*1*	*3*	∞
4	*[D, E]*	*0*	*3*	*1*	*3*	*4*
5	*[E]*	*0*	*3*	*1*	*3*	*4*
6	*[ ]*	*0*	*3*	*1*	*3*	*4*

Question 5 (20 points). *Consider the following graph.*



(a) (10 points) What is the cost of its minimum spanning tree (MST)?

(b) (4 points) How many minimum spanning trees does it have?

(c) (6 points) Suppose Kruskal's algorithm is run on this graph, in what order are the edges added to the MST?

(a)We apply Kruskal's algorithm by sorting all edges by weight and adding the smallest edge that does not form a cycle.
Edges added to the MST:

$$E–A\ (1),\ E–F\ (1),\ E–B\ (2),\ F–G\ (3),\ G–H\ (3),\ C–G\ (4),\ D–G\ (5)$$

Total cost of MST:

$$1 + 1 + 2 + 3 + 3 + 4 + 5 = 19$$

(b) There are exactly 2 different MSTs with cost 19. The only choice occurs at weight 2, where one can either add BE or BF to bring B into the spanning tree (but not both). All other edges of weight $1, 1, 3, 3, 4, 5$ are forced in order to achieve the minimum total of 19.

(c)Edges are added in the following order:

1. $AE = 1$ (added)
2. $EF = 1$ (added)
3. $BE = 2$ (added)
4. $BF = 2$ (skipped, cycle)
5. $FG = 3$ (added)
6. $GH = 3$ (added)
7. $CG = 4$ (added)
8. $BC = 5$ (skipped)
9. $CF = 5$ (skipped)
10. $DG = 5$ (added)
11. $AB = 6$ (skipped)
12. $CD = 6$ (skipped)
13. $DH = 7$ (skipped)