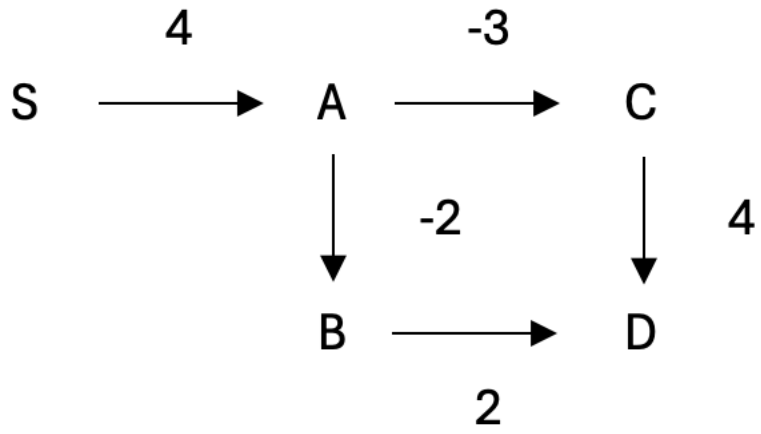
Question 1 (15 points). *Please provide an example of a weighted, directed graph $G = (V; E)$ which has some edge weights negative, and Dijkstra's algorithm correctly finds the shortest paths from a source s in the graph. You can assume the graph has no negative cycles. Please clearly indicate the source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

Dijkstra's algorithm is not designed to work correctly with graphs that contain negative edge weights. However, there is an edge case where Dijkstra's algorithm can still function correctly:

- If there are negative edge weights, but **no negative cycles**, Dijkstra's algorithm can work correctly if the negative edges only appear in situations where they don't affect the optimal path. I give an example about no negative cycle as follows:



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

Dijkstra's algorithm can still find the shortest path in this graph because the negative edges are not part of the **shortest path** from the source to the other vertices.

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.

1.

i. **Initially, calculate G = (V ; E) to use Dijkstra's algorithm twice:**
First Run: Find the shortest path from city s to all other cities; Second Run: Find the shortest path from city ttt to all other cities.

ii. **Evaluate Each Potential Road:**

```
for u, v, l in potential_cities:
    max_dist = min(dist_s[u] + l + dist_t[v], dist_t[v] + dist_s[u] + l)
```

iii. **Determine the Maximum Decrease in Distance:**

```
decrease = dist_s[t] - max_dist
if decrease > max_deceased:
    max_deceased = decrease
```

iv. **Output the Best Road:**

```
def find_max_decrease(self, s, t, potential_cities):
    dist_s = self.dijkstra(s)
    dist_t = self.dijkstra(t)
    max_deceased= 0
    best_road = None
    for u, v, l in potential_cities:
        max_dist = min(dist_s[u] + l + dist_t[v], dist_t[v] + dist_s[u] + l)
        decrease = dist_s[t] - max_dist
        if decrease > max_deceased:
            max_deceased = decrease
            best_road = (u, v, l)
    return best_road, max_deceased
```

2.

i. **Dijkstra's Algorithm**: The time complexity of Dijkstra's algorithm is O((V+E)logV). Running algorithm twice will multiply 2.

ii. Comparing the new distances for each road takes O(|E'|).

iii. Total time complexity is O((V+E)logV) + O(|E'|).

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. Firstly, Start with the largest denomination (10¢). Next, use the 5¢ denomination. Finally, use the 1¢ denomination.

ii. The running time is **O(1).**

iii.

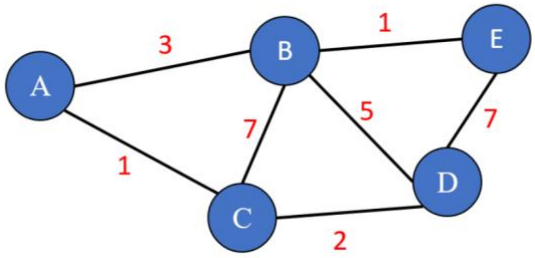
Step	N	Order of Coins	Number of Coins
1	13	10	1
2	3	1	1
3	2	1	1
4	1	1	1

iv.

If the 5¢ coin is replaced with a 6¢ coin, **the greedy algorithm will still work correctly** because:

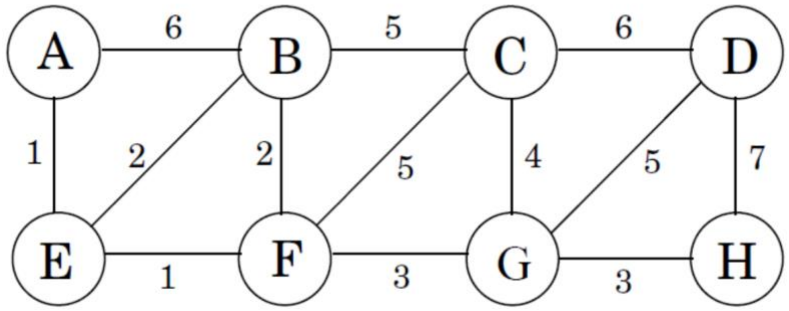
- The greedy approach always tries to use the largest denomination first.
- The problem with a 5¢ coin is that sometimes it would lead to suboptimal results.
- With a 6¢ coin, the algorithm will still select the largest possible denomination, and since 6 is a divisor of 12, it will correctly make change for numbers like 12 or 13 without suboptimal behavior.

Question 4 (20 points). *Please execute Dijkstra's algorithm from vertex A in the following graph and fill the table below. $d(\cdot)$ 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 $\neq \infty$. 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

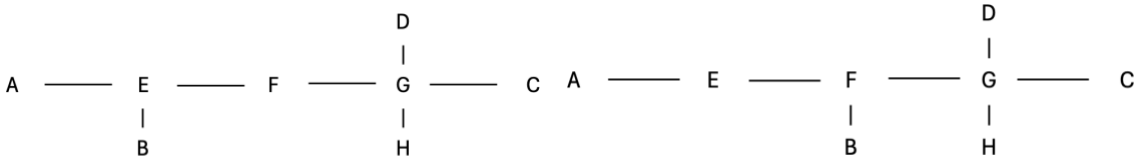
Question 5 (20 points). *Consider the following graph.*



1.

Edge	**Weight**	T/F	Weight
AE	1	T	1
EF	1	T	1
EB	2	T	2
BF	2	F	
GH	3	T	3
FG	3	T	3
CG	4	T	4
BC	5	F	
CF	5	F	
DG	5	T	5
AB	6	F	
DH	7	F	
the cost of MST			19

2.



There are 2 MST in this graph.

3.

Step1: Sort the list of edges

Weight	Edge
1	AE,EF
2	EB,BF
3	GH,FG
4	CG
5	BC,CF,DG
6	AB
7	DH

Step2: Calculate the MST

Weight	Edge	Find_Vertex	Total_Dist
1	**AE,EF**	A,E,F	2
2	**EB**,BF	A,E,F,B	4
3	**GH,FG**	A,E,F,B,G,H	10
4	**CG**	A,E,F,B,G,H,C	14
5	BC,CF,**DG**	A,E,F,B,G,H,C,D	19
6	AB	-	19
7	DH	-	19