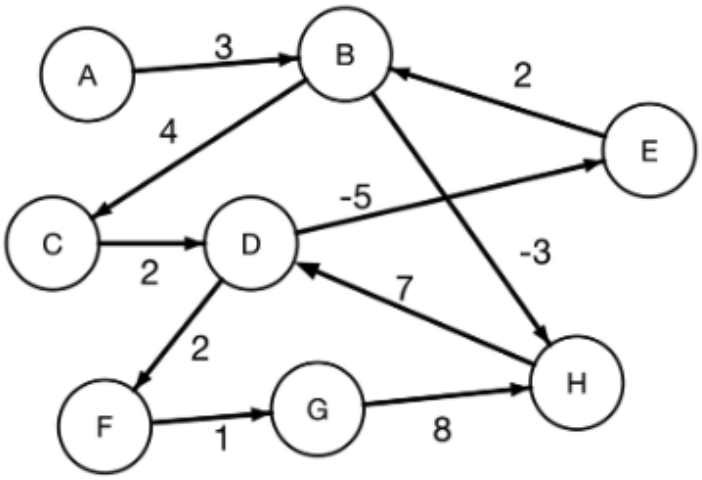
Problem Set 4
Erdun E
October 07, 2024

CS 5800: Algorithm
Problem Set 4

Exercises

1). Apply Dijkstra's algorithm on the following graph, with source vertex A to all vertices in our graph. Show all steps. Dijkstra's is not supposed to work if there are negative cost edges, but does it work anyways for this specific example? Why, or why not?



Solution:

Step 1: Initial the table of distance to the source vertex A by Dijkstra's algorithm. Table 1

Vertex	Distance from A	Visited	Previous Vertex
A	0	N	-
B	∞	N	-
C	∞	N	-
D	∞	N	-
E	∞	N	-
F	∞	N	-
G	∞	N	-
H	∞	N	-

Table 1:

Step 2: Start at vertex A, and calculate next vertex B. Table 2

Vertex	Distance from A	Visited	Previous Vertex
A	0	Y	None
B	3	N	A
C	∞	N	-
D	∞	N	-
E	∞	N	-
F	∞	N	-
G	∞	N	-
H	∞	N	-

Table 2:

Step 3: Start at vertex B, and calculate next vertex C and H. Table 3

Vertex	Distance from A	Visited	Previous Vertex
A	0	Y	None
B	3	Y	A
C	7	N	B
D	∞	N	-
E	∞	N	-
F	∞	N	-
G	∞	N	-
H	0	N	B

Table 3:

Step 4: Because H sum is smaller, so start at vertex H, and calculate next vertex D. Table 4

Vertex	Distance from A	Visited	Previous Vertex
A	0	Y	None
B	3	Y	A
C	7	N	B
D	7	N	H
E	∞	N	-
F	∞	N	-
G	∞	N	-
H	0	Y	B

Table 4:

Step 5: Because Vertex D has another way to reach that from the Vertex C, calculate C -> D = 7 + 2 = 9, greater than H -> D = 7, so keep tracking D distance from A is 7. Table 5

Vertex	Distance from A	Visited	Previous Vertex
A	0	Y	None
B	3	Y	A
C	7	Y	B
D	7	N	H
E	∞	N	-
F	∞	N	-
G	∞	N	-
H	0	Y	B

Table 5:

Step 6: Start at vertex D, and calculate next vertex E and F. Table 6

Vertex	Distance from A	Visited	Previous Vertex
A	0	Y	None
B	3	Y	A
C	7	Y	B
D	7	Y	H
E	2	N	D
F	9	N	D
G	∞	N	-
H	0	Y	B

Table 6:

Step 7: Because E sum is smaller, so start at vertex E, and calculate next vertex B. Table 7

Vertex	Distance from A	Visited	Previous Vertex
A	0	Y	None
B	3	Y	A
C	7	Y	B
D	7	Y	H
E	2	Y	D
F	9	N	D
G	∞	N	-
H	0	Y	B

Table 7:

Step 8: Because Vertex B has another way to reach that from the Vertex A, due to A -> B = 3, and E -> B = 2 + 2 = 4, greater than A -> B = 7, so keep tracking B distance from A is 3. Table 8

Vertex	Distance from A	Visited	Previous Vertex
A	0	Y	None
B	3	Y	A
C	7	Y	B
D	7	Y	H
E	2	Y	D
F	9	N	D
G	∞	N	-
H	0	Y	B

Table 8:

Step 9: Start at vertex F, and calculate next vertex G. Table 9

Vertex	Distance from A	Visited	Previous Vertex
A	0	Y	None
B	3	Y	A
C	7	Y	B
D	7	Y	H
E	2	Y	D
F	9	Y	D
G	10	N	F
H	0	Y	B

Table 9:

Step 10: Start at vertex G, and calculate next vertex H, due to G -> H = 10 + 8 = 18 and B -> H = 0, greater than B -> H, so keep tracking H distance from A is 0. Table 10

Vertex	Distance from A	Visited	Previous Vertex
A	0	Y	None
B	3	Y	A
C	7	Y	B
D	7	Y	H
E	2	Y	D
F	9	Y	D
G	10	Y	F
H	0	Y	B

Table 10:

For question: Dijkstra's is not supposed to work if there are negative cost edges, but does it work anyways for this specific example? Why, or why not?

Dijkstra's algorithm doesn't work well with negative edges because it assumes once you find the shortest path, it won't change.

For this specific graph, Dijkstra's won't work because of the negative edges. It could give wrong answers. there are negative edges like B→H and D→E. These can mess up the algorithm by making it miss shorter paths later on.

2). Apply the Bellman-Ford algorithm on graph in #1, with source vertex A to all vertices in our graph. Show all steps.

Again, initial the table of distance to the source vertex A, Table 11

Vertex	Distance from A	Visited	Previous Vertex
A	0	N	-
B	∞	N	-
C	∞	N	-
D	∞	N	-
E	∞	N	-
F	∞	N	-
G	∞	N	-
H	∞	N	-

Table 11:

Bellman Ford requires this process to be repeated $|V|$ -1 times, where $|V|$ is the number of vertices (8 in this case).

Iteration 1:

1. **A -> B(3)**

- Update B's distance:
- Distance(B) = 0 + 3 = 3

2. **B -> C(4)**

- Update C's distance:
- Distance(C) = 3 + 4 = 7

3. **B -> H(-3)**

- Update H's distance:
- Distance(H) = 3 - 3 = 0

4. **C -> D(2)**

- Update D's distance:
- Distance(D) = 7 + 2 = 9

5. **D -> E(-5)**

- Update E's distance:
- Distance(E) = 9 - 5 = 4

6. **D -> F(2)**

- Update F's distance:
- Distance(F) = 9 + 2 = 11

7. F -> G(1)

- Update G's distance:
- Distance(G) = 11 + 1 = 12

8. G -> H(8)

- No update is needed because for H, distance(H) = 0 is smaller

9. H -> D(7)

- Update D's distance:
- Distance(D) = 0 + 7 = 7

Summary in table 12

Vertex	Distance from A	Visited	Previous Vertex
A	0	Y	None
B	3	Y	A
C	7	Y	B
D	7	Y	H
E	4	Y	D
F	11	Y	D
G	12	Y	F
H	0	Y	B

Table 12:

Iteration 2:

1. A -> B(3)

- No change, due to distance(B) = 3

2. B -> C(4)

- No change, due to distance(C) = 7

3. B -> H(-3)

- No change, due to distance(H) = 3

4. C -> D(2)

- No change, due to distance(D) = 7

5. D -> E(-5)

- Update E's distance:
- Due to the distance to D is 7, relaxing the edge D -> E gives a new distance(E) = 7 - 5 = 2

6. D -> F(2)

- Update F's distance:
- Due to the distance to D is 7, relaxing the edge D -> F gives a new distance(F) = 7 + 2 = 9

7. F -> G(1)

- Update G's distance:
- Due to the distance to F is 9, relaxing the edge F -> G gives a new distance(G) = 9 + 1 = 10

8. G -> H(8)

- No change, due to distance(H) = 0

9. H -> D(7)

- No change, due to distance(D) = 7

Thus, the final shortest distances from A to all vertices are in table 13

Vertex	Distance from A	Visited	Previous Vertex
A	0	Y	None
B	3	Y	A
C	7	Y	B
D	7	Y	H
E	2	Y	D
F	9	Y	D
G	10	Y	F
H	0	Y	B

Table 13:

Conclusion:

In Iteration 1, E, F, and G were not updated to their optimal values because D was updated late. In Iteration 2, the correct shorter paths were found using the updated distance of D.