Analysis of Algorithms

Homework 4 – Shortest Path Algorithms

Abraham Murciano

December 28, 2020

1 Dijkstra's Algorithm with negative weights

Part A

Figure 1 shows a graph with negative weights such that if we apply Dijkstra's algorithm to find the shortest path between vertices S and D, it will return the wrong path.

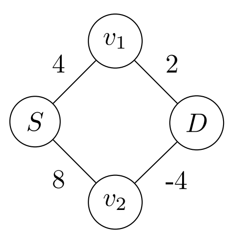

Figure 1: Graph for which Dijkstra doesn't work

Starting off, we assign the unvisited vertices v_1 and v_2 with the distances 4 and 8 respectively, marking S as visited. Then we take the unvisited vertex with the smallest distance, v_1, and check its neighbours, namely D. We assign it the distance 6 and mark v_1 as visited. Now that our destination vertex is the unvisited vertex with the shortest distance, the algorithm would claim that it has finished, with the shortest path going through v_1 with a distance of 6.

However, in reality the shortest path goes through v_2 and has a total distance of $8 - 4 = 4$. This path was not considered by the algorithm because the path to the intermediate vertex v_2 has a larger distance than the path it found first.

Part B

If we take the example graph in figure 1 and modify it so that the edges are directed (away from S or towards D), then that would form a directed acyclic graph for which Dijkstra's algorithm would not work for a similar reason to that of part A.

2 Floyd-Warshall with Negative Cycles

We are to add pseudocode to the Floyd-Warshall algorithm which checks for negative cycles. First, let us take a look at the algorithm.

```
function FloydWarshall(V, E)
    for (u, v) ∈ V × V do                         ▷ Initialise all distances to infinity
        D_{u,v} := ∞
    for (u, v) ∈ E do                             ▷ Apply distances of each edge
        D_{u,v} := Weight(u, v)
    for v ∈ V do                                  ▷ Set distance to itself to zero
        D_{v,v} := 0
    for k ∈ V do            ▷ k is a possible intermediate vertex between all (u, v)
        for u ∈ V do
            for v ∈ V do
                D_{u,v} := Min(D_{u,k} + D_{k,v}, D_{u,v})        ▷ Seek shorter path via k
    return D
```

Suppose there exists at least one negative cycle (one such that the weights of its edges sum up to a negative number) in a graph $G = (V, E)$. Now suppose $a, b \in V$ are two distinct vertices within one such cycle. At the start of the algorithm, $D_{a,a} = 0$. At some later point in the algorithm, the variables u and v will be referring to a and the variable k will be referring to b. When this occurs, $D_{a,a} = 0$ will be compared to $D_{a,b} + D_{b,a}$. However, we can be certain that $D_{a,b} + D_{b,a} < 0$, because a path $(a, \ldots, b, \ldots, a)$ forms a negative cycle. And thus $D_{a,a}$ will be assigned a negative value.

Therefore, if G contains negative cycles, when the algorithm concludes, it will tell us that $\exists v \in V$ such that $D_{v,v} < 0$. So in order to check if there are negative cycles, we can extend the algorithm as follows.

```
function FloydWarshallNegativeCycles(V, E)
    D := FloydWarshall(V, E)
    for v ∈ V do
        if D_{v,v} < 0 then
            Error: Input contains a negative cycle
    return D
```

3 Shortest Path of Alternating Colour

We are given a directed, positively weighted graph whose vertices are each either red or blue. We are to write an algorithm which seeks the shortest path from two given vertices, v_1 and v_2, which alternates colours. Meaning if a vertex in the returned path is of one colour, the vertices immediately before and after it must be of the other colour.

This can easily be achieved by removing all edges between two vertices of the same colour, then finding the shortest path with any other algorithm suitable for that purpose.

```
function ShortestAlternatingPath(V, E)
    E' := φ
    for (u, v) ∈ E do
        if Colour(u) ≠ Colour(v) then
            insert (u, v) into E'
    return ShortestPath(V, E')
```

Suppose the complexity of ShortestPath is $O(f(V, E))$, therefore the complexity of ShortestAlternatingPath would be $O(E + f(V, E))$, which would change depending on which algorithm was used by ShortestPath.

4 Running Dijkstra

We are given the graph in figure 2, and told to run Dijkstra's algorithm on it, starting from vertex 2. Table 1 shows the intermediate values of d and π throughout the running of the algorithm. Figure 3 shows the tree of shortest paths returned by the algorithm.

	d	π	d	π	d	π	d	π	d	π
1	∞	−−	∞	−−	∞	−−	∞	−−	∞	−−
2	0	−−	0	−−	0	−−	0	−−	0	−−
3	∞	−−	∞	−−	∞	−−	9	6	9	6
4	∞	−−	∞	−−	∞	−−	∞	−−	20	7
5	∞	−−	7	2	7	2	7	2	7	2
6	∞	−−	∞	−−	8	5	8	5	8	5
7	∞	−−	∞	−−	∞	−−	12	6	12	6
8	∞	−−	∞	−−	9	5	9	5	9	5

Table 1: Intermediate values of d and π for running Dijkstra on the graph in figure 2

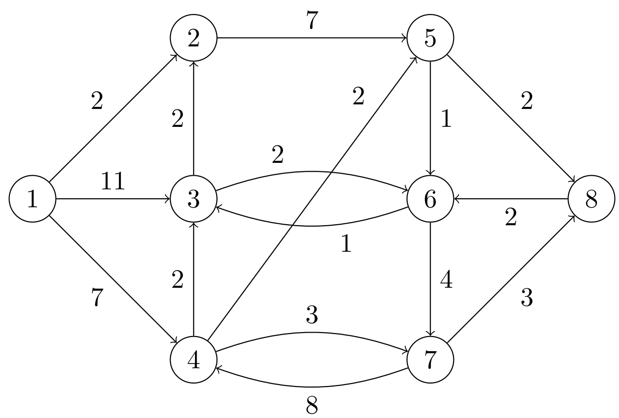

Figure 2: A directed positively weighted graph to run Dijkstra on

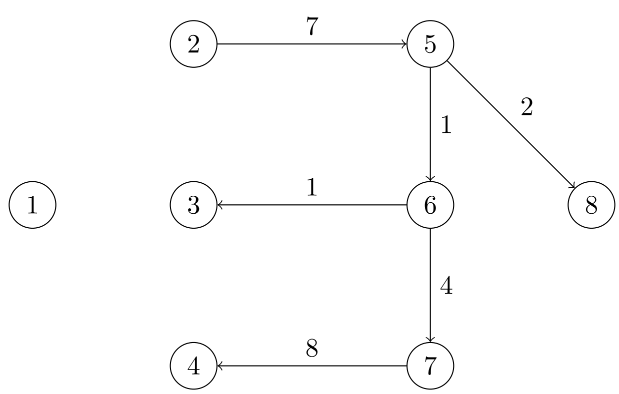

Figure 3: Tree of shortest paths of the graph in figure 2 as returned by Dijkstra

5 Does a Shortest Path Exist?

We are given a weighted directed graph G. It is known that there are negative cycles in G. Below is an algorithm that receives source and target vertices, $s, t \in G$, and checks if a shortest path from s to t exists (i.e. there is no negative cycle that can be included in this path).

```
function ExistsShortestPath(V, E, s, t)
    D := FloydWarshall(V, E)
    for v ∈ V do
        if D_{s,v} + D_{v,t} < ∞ then              ▷ There is a path from s to t via v
            if D_{v,v} < 0 then    ▷ v is part of a negative cycle (see question 2)
                return false
    return D_{s,t} < ∞              ▷ If no path from s to t then no shortest path
```

6 True or False

For each of these propositions we are to determine whether or not they are true.

Proposition A. If during the run of Bellman-Ford algorithm for $i < |V| - 1$ there are no changes applying relaxation, then shortest paths have been found and the run can be halted.

True. If there are no relaxations in one of the iterations for $i < |V| - 1$, then nothing will change between iteration i and iteration $i + 1$. Therefore there will not be any relaxations in iteration $i + 1$. For the same reason there will be no more relaxations for all iterations after the i^{th}.

Proposition B. If there is a negative cycle in a graph, then no shortest path between any two vertices can be defined.

False. There may still be a shortest path between two nodes if there does not exist any path between said nodes which contains any node in the negative cycle.

7 Bellman-Ford Example

Figure 4 shows a graph to run Bellman ford on, starting from vertex A. Table 2 shows the intermediate values of d and π throughout the running of the algorithm. Figure 5 shows the tree of shortest paths returned by the algorithm.

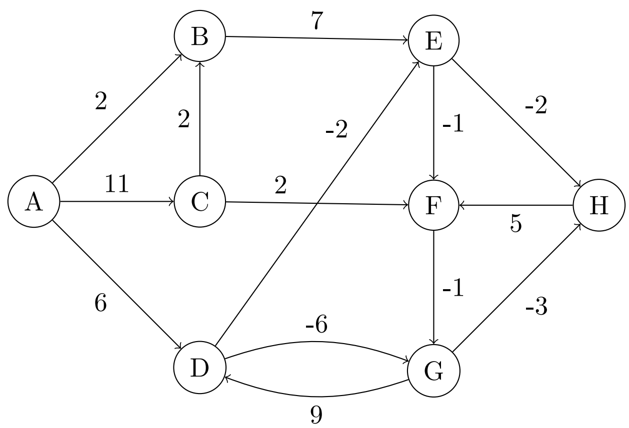

Figure 4: A directed weighted graph to run Bellman-Ford on

	d	π	d	π
A	0	$--$	0	$--$
B	∞	$--$	2	A
C	∞	$--$	11	A
D	∞	$--$	6	A
E	∞	$--$	4	D
F	∞	$--$	2	H
G	∞	$--$	0	D
H	∞	$--$	-3	G

Table 2: Intermediate values of d and π for running Bellman-Ford on the graph in figure 4

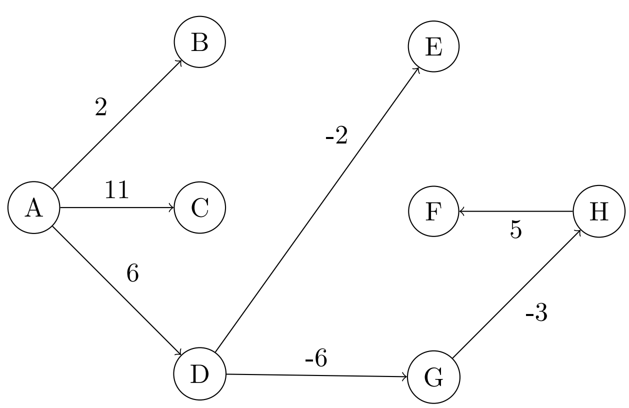

Figure 5: A tree of shortest paths returned by Bellman-Ford from the graph in figure 4