[CS252]-Algorithms II Spring 2021-22

Homework: 02

Name: Sudhir Sharma RollNo: 12041500 email: sudhirsharma@iitbhlai.ac.in
Collaborators Names:

Solution of problem1.

Forward:

We know that if a Graph G has no negative cycles, then there is a shortest path from s to t that is simple (no cycles) and hence have at most $n-1$ edges. This means that if there are no negative cycles in G, then $opt(i,v) = opt(n-1,v)$ for all vertices v and for all $i \geq n$.

Backward:

Suppose $opt(n,v) = opt(n-1,v)$ for all $v \in V$

$$\begin{aligned} opt(n,v) &= \min\left(opt(n-1,v), \min_{u \in E}(opt(n-1,u) + w_{u,v})\right) \\ \implies opt(n,v) &\leq opt(n-1,u) + w_{u,v} \ \ \forall \ u \in E \\ \implies w_{u,v} &\geq opt(n,v) - opt(n-1,u) \end{aligned}$$

Let C be any arbitrary cycle.

$$\sum_{(u,v) \in C} w_{u,v} \geq \sum_{(u,v) \in C} (opt(n,v) - opt(n-1,u)) = 0$$

Therefore, there does not exist any negative weight cycle.

The below figure is the Bellman-Ford algorithm for no negative edge weight cycle. We can create the matrix M of

```
Shortest-Path(G, s, t)
  n = number of nodes in G
  Array M[0 ... n − 1, V]
  Define M[0, t] = 0 and M[0, v] = ∞ for all other v ∈ V
  For i = 1, ..., n − 1
    For v ∈ V in any order
      Compute M[i, v] using the recurrence (6.23)
    Endfor
  Endfor
  Return M[n − 1, s]
```

size $(0...n,V)$ instead of $(0...n-1,V)$. We can run the outermost for loop for n times instead of $n-1$ times. After the end of algorithm and before the return statement, we can include a for loop to check whether $M[n][v] = M[n-1][v]\ \forall\ v \in V$. If not, then there exists a negative weight cycle.

Below is the modified Bellman-Ford Algorithm.

Algorithm 1 Bellman-Ford Modified Algorithm

```
1   procedure BELLMAN-FORD-MODIFIED(G,s,t)
2   n = number of nodes in G
3   Array M[0...n,V]                                          ▷ n + 1 rows
4   Define M[0,t] = 0 and M[0,v] = ∞ for all other v ∈ V
5   for i ← 1,n do                                            ▷ Run till n
6       for v ∈ V do
7       M(i,v) = min(M(i − 1,v),min_{u∈A}(M(i − 1,u) + w_{u,v}))
                                                              ▷ Check for negati
8       for v ∈ V do                                          cycle
9       if M[n][v] != M[n − 1][v] then
10      return True,M[n − 1,s]  ▷ Return true i.e. ne   gative cycle exists and M[n − 1,s]
11  return False,M[n − 1,s]   ▷ Return false i.e. negative cycle does not exists and  M[n − 1,s]
```

Proof of detection of negative-weight cycles.

Suppose $v_0 \to v_1 \to \cdots \to v_k$ is a negative cycle reachable from s, where $v_0 = v_k$. Formally, this means we have

$$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$$

We proceed by contradiction. Suppose that we have $d_{n-1}(v_i) \leq d_{n-1}(v_{i-1}) + w(v_{i-1}, v_i)$ for all $i = 1, \ldots, k$. Then summing up the inequality for each vertex on the cycle, we get

$$\sum_{i=1}^{k} d_{n-1}(v_i) \leq \sum_{i=1}^{k} d_{n-1}(v_{i-1}) + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Observe that the first two terms are the same. That's because $v_0 = v_k$, so

$$\sum_{i=1}^{k} d_{n-1}(v_{i-1}) = \sum_{i=0}^{k-1} d_{n-1}(v_i) = d_{n-1}(v_0) + \sum_{i=1}^{k-1} d_{n-1}(v_i) = \sum_{i=1}^{k} d_{n-1}(v_i)$$

Because those two terms are the same, we can cancel them out to get

$$\sum_{i=1}^{k} w(v_{i-1}, v_i) \geq 0$$

contradicting the supposition that $v_0 \to v_1 \to \cdots \to v_k$ is a negative cycle.

Solution of problem2.

Backward:

Let$d_{ii}^{(n)}$ is the weight of the path from vertex$i$ to i. If $d_{ii}^{(n)} < 0$, it means that there is a path from i to i i.e a cycle of negative weight.

Forward:

Let G has a negative weight cycle. Let C be the mininum negative weight cycle out of all cycles. If C has only one vertex (i.e. self loop), then$w_{ii} < 0$ so d_{ii}^{k} for some k starts negative and remains negative becaused_{ii}^{k} value is never increased in the algorithm.

If the cycle consists at least two vertices. Let k be the highest numbered vertex in the cycle and i be any other vertex in the cycle. and have the shortest paths weights $d_{ik}^{(k-1)}$ $d_{ki}^{(k-1)}$ because neither of them include the vertex k as intermediate vertex and i and k are on C which has the minimum negative weight. Since $i \rightarrow k \rightarrow i$ is a

negative weight cycle, the sum of there weights becomes negative so $d_{ii}^{(k)}$ becomes neagtive and its value is never increased in the algorithm.

The below figure is the Floyd-Warshall algorithm for no negative edge weight cycle.

```
FLOYD-WARSHALL(W)
1  n = W.rows
2  D^(0) = W
3  for k = 1 to n
4      let D^(k) = (d_ij^(k)) be a new n × n matrix
5      for i = 1 to n
6          for j = 1 to n
7              d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
8  return D^(n)
```

Figure 1.1: Flyod Warshall Algorithm

After line 7, in the above figure we can check weather the diagonal entries of matrix $D^{(n)}$. If any of the entry is negative, it means that the graph contains a negative weight cycle. If not, then the graph has no negative weight cycle.

Add the following loop After line 7 and before the return statement in Figure1.1.

Algorithm 2 Floyd-Warshall updated

```
1  procedure MODIFICATION                ▷ Check for diagonal entries of D^(n)
2  for i ← 1,n do
3  if d_ii^(n) < 0 then
4  return False
```

Final Algorithm

Floyd-Warshall-Algorithm

Input: A digraph G with $V(G) = \{1, \ldots, n\}$ and weights $c : E(G) \to \mathbb{R}$

Output: An $n \times n$ matrix M such that $M[i, j]$ contains the length of a shortest path from vertex i to vertex j.

```
1  M[i,j] := ∞ ∀i ≠ j
2  M[i,i] := 0 ∀i
3  M[i,j] := c((i,j)) ∀(i,j) ∈ E(G)
4  for i := 1 to n do
5      for j := 1 to n do
6          for k := 1 to n do
7              if M[j,k] > M[j,i] + M[i,k] then M[j,k] := M[j,i] + M[i,k]
8  for i := 1 to n do
9      if M[i,i] < 0 then return('graph contains a negative cycle')
```

Proof of detection of negative-weight cycles.

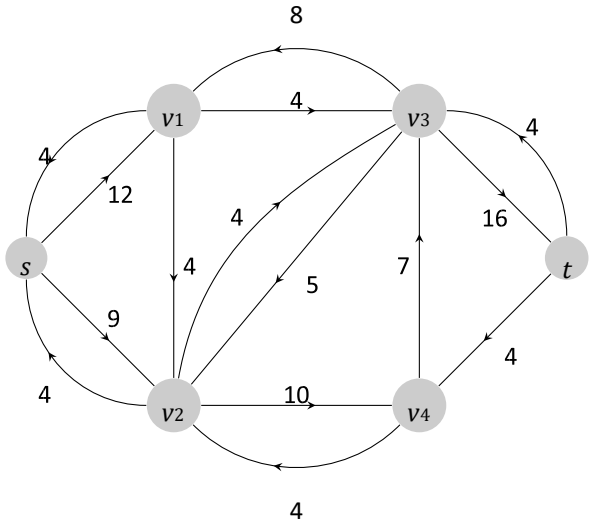
If no negative-weight edges are present, which is often the case, the final loop may be omitted altogether from the algorithm, since it will never be useful. If negative-weight edges are present, then the final loop will *always* detect the presence of a negative-weight cycle. Suppose a negative-weight cycle exists. Then, choose any vertex k on this cycle. At the termination of the main loop, *dist[k][k]* will be negative, since the algorithm will inevitably have found the shortest path from k back to itself using each vertex in the graph at most once (the reason for this is explained in the previous section), which is, of course, of negative weight. (The presence of negative-weight cycles implies the presence of negative-weight simple cycles, as all non-simple cycles can be decomposed into simple cycles.) The algorithm then prints out an appropriate error message.

Solution of Problem 3

(a) Value of the flow is sum of flow generated at source i.e. 4+4 = 8. This is not the

maximum $s-t$ flow in this graph.

(b) Residual graph



(c) Value of the minimum $s-t$ cut is same as the max flow from $s-t$. $v(f) = 23$. Send 12 along edge $s-v_1$ and send 11 along edge $s-v_2$. The minimum cut can be $A = \{s\}$ and $B = V-\{s\}$. The capacity of this cut is $16+13 = 29$.