COMP 6651: Solutions to Assignment 7

Fall 2020

Submission through Moodle is due by November 8th at 23:55

1. (a) $D[v] =$ the weight of the longest weighted path from s to v.

   (b) $D[v] = \max_{u:(u,v)\in E}\{w(u,v) + D[u]\}$, where max over an empty set is defined as $-\infty$ indicating that a node is not reachable from s. Base case is $D[s] = 0$. The array entries should be computed in a topological order. This ensures that in the max computation for $D[v]$ entries $D[u]$ have been previously computed.

   (c) Longest weighted path in a DAG has an optimal substructure property. If $p = \langle v_0, v_1, \ldots, v_k\rangle$ is a longest weighted path from $v_0 = s$ to $v_k = v$ then $p' = \langle v_0, v_1, \ldots, v_{k-1}\rangle$ must be a longest weighted path from s to $v_{k-1} = u$. This is by the cut-and-paste property. If p' was not longest weighted path from s to u and some other path p'' from s to u satisfied $w(p'') > w(p')$ then we could append edge (u, v) to p'' and obtain p''' – path from s to v with total weight

   $$w(p''') = w(p'') + w(u,v) > w(p') + w(u,v) = w(p),$$

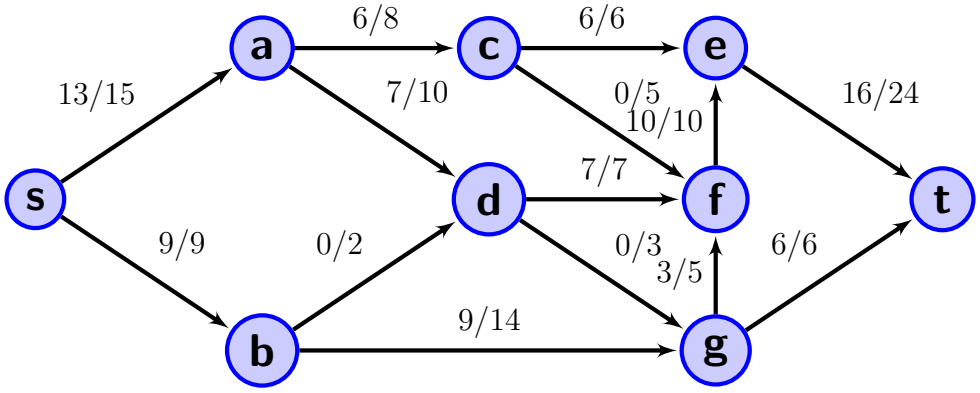
   contradicting the fact that p is a longest weighted path from s to v.

   (d) Topological sort can be done in $O(V + E)$. After that the algorithm examines each edge (u, v) at most once (when $D[v]$ is computed). Examination of each edge requires constant time. Each vertex is also examined once. Thus, the overall time is $O(V + E)$.

   (e) It works with negative weight edges because our correctness argument does not require edges to be positive.

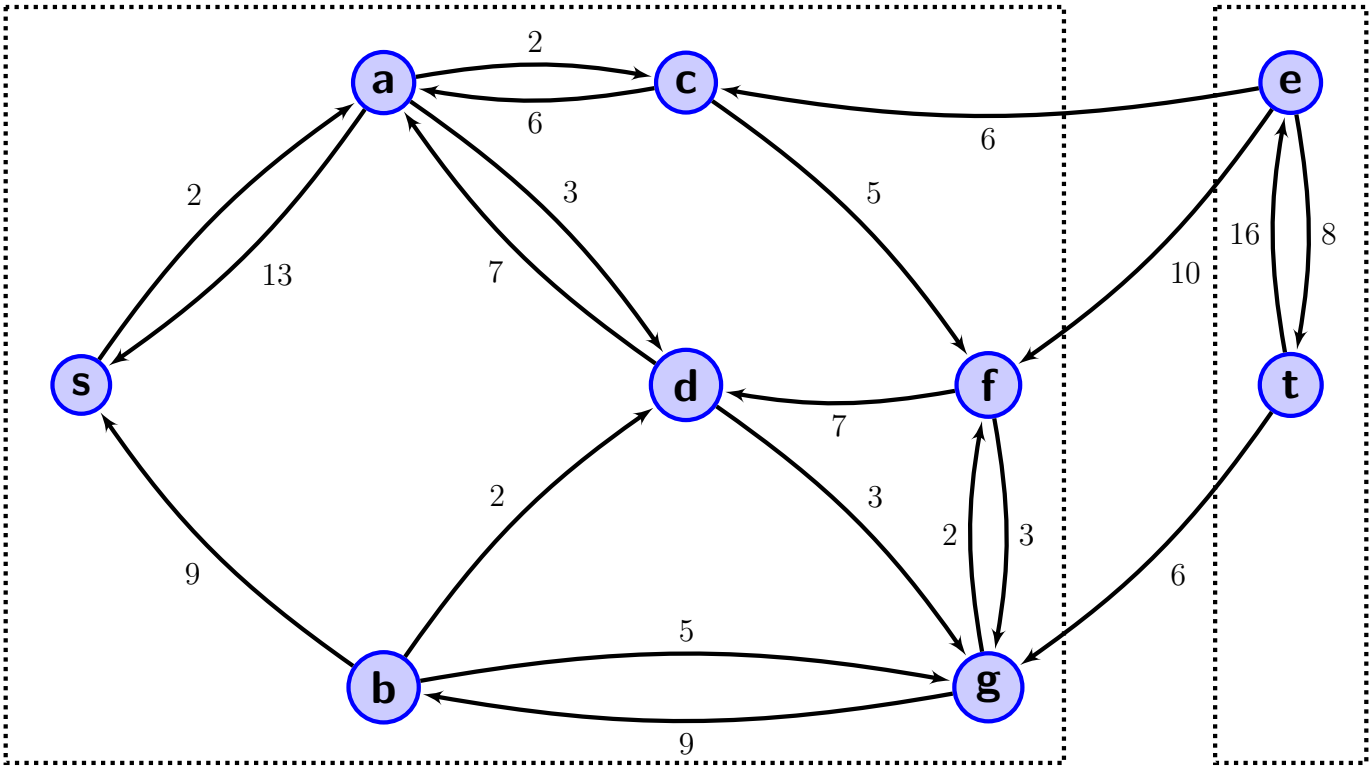
   (f) The algorithm does not work if G is not a DAG, since topological sort fails. If you try running the algorithm anyway, it would fill in the table $D[\cdot]$ with some numbers, but they cannot be correct in general, since longest weighted paths are undefined for nodes in a positive-weight cycle.

2. (a) The figure is below



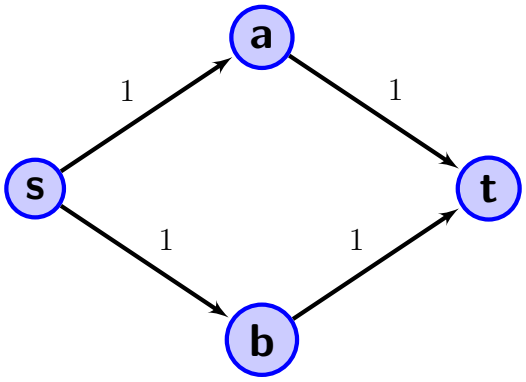
Max flow f with value $|f| = 22$. Min cut (S, T) where $S = \{s, a, b, c, d, f, g\}$ and $T = \{e, t\}$ with capacity $c(S, T) = 6 + 10 + 6 = 22$.

(b) The figure is below



(c) $(c, e), (f, e), (g, t)$.

(d) The figure is below



(e) Algorithm: compute max flow f. Run BFS from s in the residual graph G_f to find the set of nodes U_s that are reachable from s. Compute the transpose of G_f that is G_f^T by reversing all edges. Run BFS from t in the transpose of the residual graph G_f^T to compute the set U_t such that t is reachable from each node in U_t in G_f. Declare all edges $(u, v) \in E$ such that $u \in U_s$ and $v \in U_t$ to be constricting.

Argument of correctness: first note that since f is maximum $U_s \cap U_t = \emptyset$. By Max-Flow Min-Cut theorem an edge (u, v) is constricting if and only if it has residual capacity 0 and increasing its capacity by any nonnegative amount creates a new augmenting path (independent of which maximum flow f we use to define residual capacity). This happens if and only if prior to the increase of the capacity u was reachable from s and t was reachable from v in the residual graph. The above algorithm computes exactly the set of such edges.

Running time: first step is to run a max-flow algorithm. This could be, for instance, Edmonds-Karp, which takes $O(|V||E|^2)$ time. After that, constructing the residual graph, its transpose, running BFS - all take linear time, i.e., $O(|V| + |E|)$, assuming adjacency lists representation.

3. (a) Construct a weighted graph $G' = (V, E')$ with weights $w : E' \to \mathbb{R}$, such that

   - if (u, v) has residual capacity $c_f(u, v) > 0$ then $(u, v) \in E'$ and $w(u, v) = 0$;
   - if (u, v) has residual capacity $c_f(u, v) = 0$ and $(u, v) \in E$ then $(u, v) \in E'$ and $w(u, v) = p(u, v)$.

   Observe that G' has positive edge-weights. Run Dijkstra's algorithm from s on G'. Return the edges (u, v) on the shortest path computed by Dijkstra from s to t such that $w(u, v) > 0$.

   (b) Pseudocode is given below:

```
ComputeEdgesToIncrease(G=(V,E),s,t,c,f,p)
  initialize graph G' on vertices V and weight function w
  for each edge (u,v) in E
    if c_f(u,v) > 0
      add (u,v) to E'
      w(u,v) = 0
    if c_f(v,u) > 0
      add (v,u) to E'
      w(v,u) = 0
    if c_f(u,v) == 0
      add(u,v) to E'
      w(u,v) = p(u,v)
  Run Dijkstra on G',w from vertex s
  Dijkstra computes arrays dist[] and prev[]

  Result = empty set
  end = t
  while end != s
    if w(prev[end],end) > 0
      Result.add((prev[end],end))
    end = prev[end]

  return Result
```

   (c) By Max-Flow Min-Cut theorem we can increase flow f by one unit if and only if there is an augmenting path in G_f (since flow and capacities are integral, the residual capacity of such a path would be at least 1). Since f is maximum there are no more augmenting paths. In particular, this happened because certain edges that were present in G disappeared from G_f as they got saturated (their residual capacities became 0). We can reintroduce edge e by increasing its capacity by 1 and paying the price $p(e)$. Thus, the original question of this problem can be rephrased as finding the cheapest augmenting path in the graph G' that has all edges of G_f with weight 0 (those are readily available to be used in augmenting path without any extra price) and the edges of G that got saturated by f with the weight given by p. This is precisely what the algorithm does.

   (d) Constructing the graph G' and the weights w takes $O(|V| + |E|)$ time assuming the input graph is given in the adjacency lists representation. Running Dijkstra on this graph and using the binary heap implementation of the priority queue results in the dominating part of the overall

runtime $O((|V| + |E|) \log |V|)$. This is because the last part – reconstructing the solution – takes $O(|V|)$ time.