CS/ECE 374 ✧ Spring 2017
Homework 8 Question 2

Due Wednesday, April 5, 2017 at 10am

Ho Yin Au (hoyinau2), Lanxiao Bai(lbai5), Renheng Ruan(rruan2)

- Question 1

  To find the best edge not in $E(H)$ to add, we need to know the distance of each node of H to start vertex s (denoted as $dist(s,v)$ for each node v). Then we can check each edge not in $E(H)$ but in $E(G)$ that can further minimize distance from s (denoted as $newdist(s,v)$ for each node v).

  The first part of the algorithm, which is know the distance of each node of H to start vertex s, can be done with Dijkstra's algorithm on graph H. For the second part, we need to use the Dijkstra's algorithm to gather another information for the case of adding edge. The information we needed are the shortest path from s to t that allows adding edge, and $newdist(s,t)$ that allows adding edge to know if that is shorter than the one without adding edge. After we get the information, we can just search on the path for the edge not in $E(H)$ and output that edge(guarantee at most 1 edge, if no edge not in $E(H)$, return $null$).

  To find the shortest path from s to t that allows adding edge, we need to record $dist(s,v)$ for each vertex v that allows adding edge. $newdist(s,v)$ will be updated according to the incoming edge of v. If the incoming edge uv in $E(H)$, $newdist(s,v) = min(newdist(s,u)+w(uv), newdist(s,v))$, where $w(uv)$ is weight of edge uv. If the incoming edge uv not in $E(H)$, $newdist(s,v) = min(dist(s,u)+w(uv), newdist(s,v))$, where $w(uv)$ is weight of edge uv, $dist(s,u)$ is the distance of s to u according to Dijkstra's algorithm on graph H. To record the shortest path, we also need to record the parent of shortest path for each node v (denoted as $parent(v)$) in the both parts, which will be updated if a shorter path is found from s to node v.

  For the running time, we run one Dijkstra's algorithm for the first part, another one Dijkstra's algorithm for the second part, and use $O(|V(G)|)$ to get the shortest path from s to t according to stored parent information and identify the last edge in the shortest path not in $E(H)$ (we know there is at most 1 edge). As the running time of Dijkstra's algorithm is $O(|E(G)|+|V(G)|\ log(|V(G)|))$, $O(|E(G)|+|V(G)|\ log(|V(G)|))+O(|E(G)|+|V(G)|\ log(|V(G)|))+O(|V(G)|) = O(|E(G)|+|V(G)|\ log(|V(G)|))$, which is similar to running time of Dijkstra's algorithm.

  The algorithm is named FindBestReinsertEdge. All subroutine is listed below this algorithm. Assume all the subroutine can read and write on dist, newdist, and parent. $w(uv)$ return edge weight of $uv \in E(G)$. Assume all operation of priority queue (Q.$popMinBySmallestWeight()$ for example) costs $O(log(|V(G)|))$ time. Q.$popMinBySmallestWeight(excludeVertices =$ searchedVertexSet$)$ means that the pop will return the vertex with smallest weight according to the number that grouped with vertex by a tuple when insert to priority queue and will not pop vertices already in searchedVertexSet. Note that in case of vertex t in the problem is not reachable from s, FindBestReinsertEdge and its subroutine FindShortestPathEdgeNotInGraph will return $null$.

```
FindBestReinsertEdge(Graph $G$,Graph $H$, Vertex $s$, Vertex $t$) :

    Integer dist$[1..|V(G)|]$
    Integer newdist$[1..|V(G)|]$
    Vertex parent$[1..|V(G)|]$

    for each $v$ in $V(G)$
        dist$[v] = \infty$
        parent$[v] = null$

    dist$[s] = 0$

    Dijkstra 1$(H, s)$

    newdist $\leftarrow$ dist$.copy()$

    Dijkstra 2$(G, H, s)$

    return FindShortestPathEdgeNotInGraph$(H, s, t)$
```

```
Dijkstra 1(Graph $H$, Vertex $s$) :

    # comment: modified from cs374 lecture slides

    Q $= createPriorityQueue()$
    Q$.insert((s, 0))$

    for each $v$ in $V(H)$
        Q$.insert((v, \infty))$

    searchedVertexSet $= \emptyset$

    for $i = 1$ to $|V(H)|$
        if Q$.empty$
            break out the loop

        $(v,$ dist$[v]) =$ Q$.popMinBySmallestWeight(excludeVertices =$ searchedVertexSet$)$
        searchedVertexSet$.addToSet(v)$
        for each $vw$ in $Out(v)$
            if dist$[v] + w(vw) <$ dist$[w]$
                dist$[w] \leftarrow$ dist$[v] + w(vw)$
                parent$[w] \leftarrow v$
                Q$.removeByVertex(w)$
                Q$.insert((w,$ dist$[w]))$
```

```
DIJKSTRA 2(Graph $G$, Graph $H$, Vertex $s$) :

  # comment: modified from Dijkstra 1

  Q = $createPriorityQueue()$
  Q.$insert((s, 0))$

  for each $v$ in $V(G)$
      Q.$insert((v, \infty))$

  SEARCHEDVERTEXSET = $\emptyset$

  for $i = 1$ to $|V(G)|$
      if Q.$empty$
          break out the loop
      $(v, \textsc{newdist}[v])$ = Q.$popMinBySmallestWeight(excludeVertices = \textsc{searchedVertexSet})$
      SEARCHEDVERTEXSET.$addToSet(v)$
      for each $vw$ in $Out(v)$
          if $vw \notin E(H)$
              DIST $\leftarrow \textsc{dist}[v] + w(vw)$
          else
              DIST $\leftarrow \textsc{newdist}[v] + w(vw)$

          if DIST < $\textsc{newdist}[w]$
              $\textsc{newdist}[w] \leftarrow$ DIST
              $\textsc{parent}[w] \leftarrow v$
              Q.$removeByVertex(w)$
              Q.$insert((w, \textsc{newdist}[w]))$
```

```
FINDSHORTESTPATHEDGENOTINGRAPH(Graph $H$, Vertex $s$, Vertex $t$) :

  $v \leftarrow t$
  while $v \neq s$
      $u \leftarrow \textsc{parent}[v]$
      if $u = null$
          return $null$

      if $uv \notin E(H)$
          return $uv$
      $v \leftarrow u$

  return $null$
```