CS 4349 — Advanced Algorithm Design and Analysis, Homework 5

https://iniyanjoseph.com | iniyanijoseph@gmail.com

Question 1

a

15.1-3

Not just any greedy approach to the activity-selection problem produces a maximum-size set of mutually compatible activities. Give an example to show that the approach of selecting the activity of least duration from among those that are compatible with previously selected activities does not work. Do the same for the approaches of always selecting the compatible activity that overlaps the fewest other remaining activities and always selecting the compatible remaining activity with the earliest start time.

b

We must show that the approach of selecting the activity of least duration greedily does not provide an optimal solution.
We take an example as follows

Activity	s_i	f_i
1	5	7
2	0	6
3	6	10

Here, selecting this algorithm will choose activity 1, which will be the only available activity. In contrast, both longer tasks can be selected if the short activity is not selected.

c

We must show that the approach of selecting the activity of fewest overlaps greedily does not provide an optimal solution.
We take an example as follows

Activity	s_i	f_i
1	0	2
2	0	1
3	0	1
4	0	1
5	2	4
6	3	6
7	5	7
8	2	3
9	6	7

Greedy: 3-6, 6-7,0-2. Optimal: 0-1, 2-3,3-6,6-7

d

We must show that the approach of selecting the activity of earliest start time does not provide an optimal solution.
We take an example as follows

Activity	s_i	f_i
1	0	3
2	0	1
3	2	3

Question 2

a

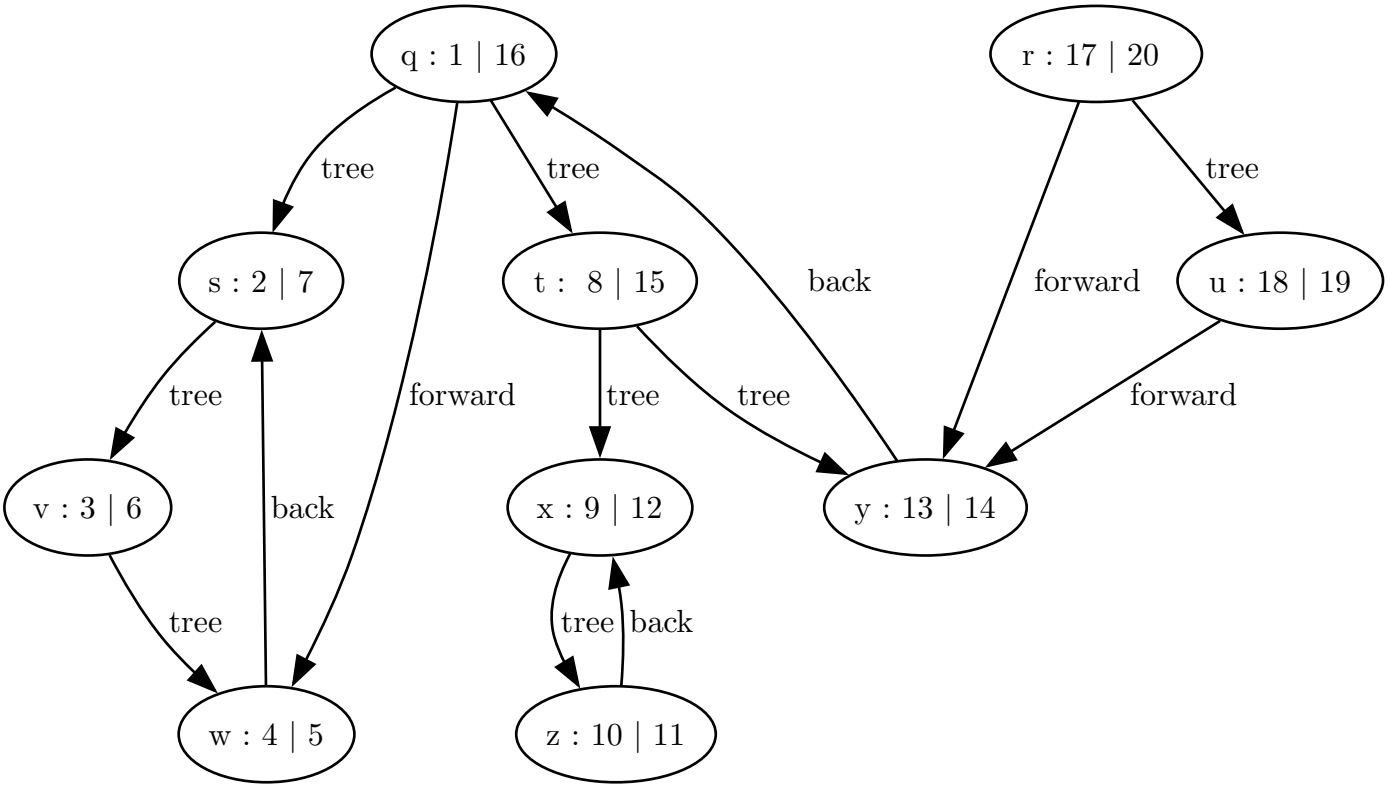
20.3-2

Show how depth-first search works on the graph of Figure 20.6. Assume that the **for** loop of lines 5–7 of the DFS procedure considers the vertices in alphabetical order, and assume that each adjacency list is ordered alphabetically. Show the discovery and finish times for each vertex, and show the classification of each edge.

```
digraph {
  q -> s
  s -> v
  v -> w
  w -> s
  q -> w
  q -> t
  t-> x
  x-> z
  z-> x
  t -> y
  y-> q
```

```
  r -> u
  u-> y
  r -> y
}
```

b


q : 1 | 16
r : 17 | 20
s : 2 | 7
t : 8 | 15
u : 18 | 19
v : 3 | 6
x : 9 | 12
y : 13 | 14
w : 4 | 5
z : 10 | 11
tree
tree
tree
back
forward
tree
tree
forward
tree
back
tree
tree
back

Question 3

21.1-4

Give a simple example of a connected graph such that the set of edges $\{(u, v) :$ there exists a cut $(S, V - S)$ such that (u, v) is a light edge crossing $(S, V - S)\}$ does not form a minimum spanning tree.

$$V = \{1, 2, 3, 4\}$$

$$E = \{(1, 2), (1, 3), (2, 3), (2, 4), (3, 4)\}$$

With weights $1, 2, 1, 7, 2$ respectively. Take for example $S = \{1\}$. If vertices 2, 3, and 4 are already connected, it may be by the edges $\{(2, 4), (3, 4)\}$. The unique light crossing edge adds $(1, 2)$, however although all vertices are now connected, the graph is not a minimum spanning tree. Instead of the edge $(2, 4)$, the edge $(2, 3)$ will form the minimum spanning tree.

In other words, choosing an arbitrary light edge does not create a minimum spanning tree unless the connection between every disjoint pair of subgraphs in a connected graph is by the light edge.

Question 4

22.3-11

Suppose that you are given a weighted, directed graph $G = (V, E)$ in which edges that leave the source vertex s may have negative weights, all other edge weights are nonnegative, and there are no negative-weight cycles. Argue that Dijkstra's algorithm correctly finds shortest paths from s in this graph.

Here, we can exploit the property of the SSSP having optimal substructure. First, observe that Dijkstra's algorithm can be run from all vertices x such that (s, x), since there are no negative weights in this graph. Let $S = \{(s, x) : w((s, x)) < 0 \wedge (s, x) \in E\}$, the set of all negative weight edges. Let D_i be the distance from x to $i \in V \setminus \{s\}$

The shortest path from s to x is clearly the weight of the edge (s, x), since there are no negative weight cycles which could lead to a shortest path with less cost. Thus by the definition of optimal substructure, the distance from s to all other vertex $i \in V \setminus \{s\}$ in the graph are $D'_i = \min_{y \in S}(y + D_i)$

Question 5

23.2-6

Show how to use the output of the Floyd-Warshall algorithm to detect the presence of a negative-weight cycle.

The Floyd-Warshall algorithm works by computing the lengths of paths going through a give intermediary vertex.

In a positive cycle, the cost of traversing the cycle is always greater than 0. However, in a negative cycle, the cost to traverse from a node to itself can be minimized by going through the cycle, and thus the diagonal of the output matrix will be negative for all nodes in a negative cycle.