Note: Your TA may not get to all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. The discussion worksheet is also a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 Short Answer

For each of the following, either prove the statement is true or give a counterexample to show it is false.

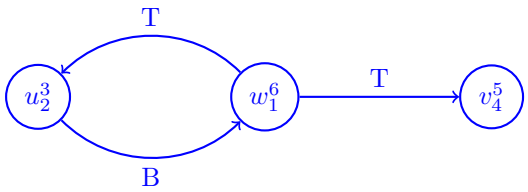
(a) If (u, v) is an edge in an undirected graph and during DFS, $\text{post}(v) < \text{post}(u)$, then u is an ancestor of v in the DFS tree.

(b) In a directed graph, if there is a path from u to v and $\text{pre}(u) < \text{pre}(v)$ then u is an ancestor of v in the DFS tree.

(c) In any connected undirected graph G there is a vertex whose removal leaves G connected.

Solution:

(a) True. There are two possible cases: $\text{pre}(u) < \text{pre}(v) < \text{post}(v) < \text{post}(u)$ or $\text{pre}(v) < \text{post}(v) < \text{pre}(u) < \text{post}(u)$. In the first case, u is an ancestor of v. In the second case, v was popped off the stack without looking at u. However, since there is an edge between them and we look at all neighbors of v, this cannot happen.



(b) False. Consider the following case:

(c) True. Remove a leaf of a DFS tree of the graph.

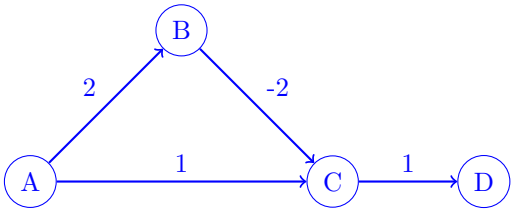
2 Dijkstra's Algorithm Fails on Negative Edges

Draw a graph with five vertices or fewer, and indicate the source where Dijkstra's algorithm will be started from.

1. Draw a graph with no negative cycles for which Dijkstra's algorithm produces the wrong answer.
2. Draw a graph with at least two negative weight edge for which Dijkstra's algorithm produces the correct answer.

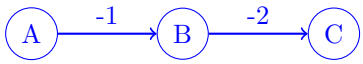
Solution:

1. Here's one example:



Dijkstra's algorithm from source A will give the distance to D as 2 rather than 1, because it visits C before B.

2. Dijkstra's algorithm always works on directed paths. For example:



3 More Short Answer Questions

(a) Let G be a dag and suppose the vertices $v_1, \ldots, v_n$ are in topologically sorted order. If there is a path from v_i to v_j in G, are we guaranteed that $i < j$?

(b) Let $G = (V, E)$ be any strongly connected directed graph. Is it always possible to remove one vertex (as well as edges to and from it) from the graph so that the remaining directed graph is strongly connected?

(c) Give an example where the greedy set cover algorithm does not produce an optimal solution.

(d) Let G be a connected undirected graph with positive lengths on all the edges. Let s be a fixed vertex. Let $d(s, v)$ denote the distance from vertex s to vertex v, i.e., the length of the shortest path from s to v. If we choose the vertex v that makes $d(s, v)$ as small as possible, subject to the requirement that $v \neq s$, then does every edge on the path from s to v have to be part of every minimum spanning tree of G?

(e) The same question as above, except now no two edges can have the same length.

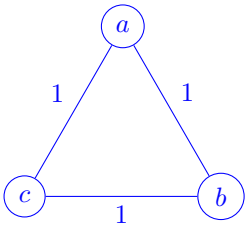
Solution:

(a) $\boxed{Yes}$ Consider a path $v_{i_1} \to \cdots \to v_{i_k}$. Since the graph is in topological order, we can conclude that $i_j < i_{j+1}$ for $j = 1, \ldots, k-1$, so by transitivity, $i_1 < i_k$.

(b) $\boxed{No}$ Consider the graph with 3 nodes a, b and c and edges $a \to b$, $b \to c$ and $c \to a$.

(c) With items $\boxed{B = \{a, b, c, d, e, f\}}$ and sets $\boxed{\{a, b, c\}, \{d, e, f\}, \{a, b, d, e\}}$ we would first choose {a,b,d,e} which would then require all three sets, while choosing the first two sets is sufficient.

(d) False. Consider the following counterexample:



Take $s = a$. Both $v = b$ and $v = c$ minimize $d(a, v)$, but neither edge is part of every MST: for instance, (a, b) and (b, c) form a minimum spanning tree that does not contain (a, c).

(e) True. First let's analyze the the definition. We need to find a vertex v that minimizes $d(s, v)$. Because the edge weights are positive, v has to be a neighbor a s. Or in other words, part (h) claims the lightest edge that is incident on s has to part of every minimum MST. Let us call this edge e. Assume, for contradiction, that e is not part of some MST T. Let us add e to T. This creates a cycle, which goes through e to s and exit s through another edge e'. We now remove e'. Removing an edge from a cycle keeps the graph connected and a tree. This creates a tree which is lighter than T which contradicts our assumptions that T is a MST.

4 Unique Shortest Path

Shortest paths are not always unique: sometimes there are two or more different paths with the minimum possible length. Show how to solve the following problem in $O((|V| + |E|) \log |V|)$ time.
Input: An undirected graph $G = (V, E)$; edge lengths $l_e > 0$; starting vertex $s \in V$.
Output: A Boolean array `usp[·]`: for each node u, the entry `usp[`u`]` should be `true` if and only if there is a *unique* shortest path from s to u. (Note: `usp[`s`] = true`.)
[Provide 3 part solution.]

Solution: Main Idea:
Suppose there are two different shortest paths from s to u. These two paths can either share the same last edge (the edge ending at u), or not. If they do, this can be detected by modifying Dijkstra's to detect if a node has been added to the known region previously with the same distance. If not there must be two different shortest paths to u's parent, which can be detected by propagating (for every edge (a, b)) `usp`(a) to `usp`(b) if DECREASEKEY(H, v) is called.

Pseudocode:
This can be done by slightly modifying Dijkstra's algorithm. The array `usp[·]` is initialized to `true` in the initialization loop. The main loop is modified as follows (lines 7-9 are added):

```
1: while H is not empty do
2:     u = DELETEMIN(H)
3:     for all (u, v) ∈ E do
4:         if dist(v) > dist(u) + l(u, v) then
5:             dist(v) = dist(u) + l(u, v)
6:             DECREASEKEY(H, v)
7:             usp[v] = usp[u]
8:         else if dist(v) = dist(u) + l(u, v) then
9:             usp[v] = false
```

Proof of Correctness:
By Dijkstra's proof of correctness, this algorithm will identify the shortest paths from the source u to the other vertices. For uniqueness, we consider some vertex v. Let p denote the shortest path determined by Dijkstra's algorithm. If there are multiple shortest paths, then take another path $p' \neq p$ and it will either share the same final edge (w, v) (for some vertex w) as p, or they have different final edges from p. In the former case, there must be multiple shortest paths from u to w. Using an inductive argument, which supposes that `usp` is already set correctly for all vertices that are closer than v, this will be detected in the first conditional statement when the algorithm explores from w and updates the distance to v by taking edge (w, v), as it will detect that there are multiple shortest paths to w. In the latter case, if the last edges of the two shortest paths are (w_1, v) and (w_2, v), then since edge lengths $l_e > 0$, both w_1 and w_2 must be visited before v, thus the algorithm will detect the existence of multiple shortest paths with the second conditional statement. The base case is true because there is a unique shortest path from the source s to itself.

Runtime:
The runtime analysis follows that of Dijkstra's, and will run in the required time when a binary heap is used for the priority queue.