Tutorial

Question 1

A depth-first forest classifies the edges of a graph into tree, back, forward, and cross edges. A breadth-first tree can also be used to classify the edges reachable from the source of the search into the same four categories. Prove that in a breadth-first search of a directed graph, the following properties hold:

1. *There are no forward edges.*
2. *For each tree edge $\langle u, v\rangle$, we have $d[v] = d[u] + 1$.*
3. *For each cross edge $\langle u, v\rangle$, we have $d[v] \leq d[u] + 1$.*
4. *For each back edge $\langle u, v\rangle$, we have $0 \leq d[v] \leq d[u]$.*

Background: Breadth First Search

Breadth First Search starts at a source s. Each vertex, u, has a distance, $d[u]$, from s. Intuitively, $d[s] = 0$. If the graph is not a connected graph (there is a path from each vertex to every other vertex), then vertices not reachable from s will have an infinite distance from s. That is, $d[u] = \infty$, where u is not reachable from s.

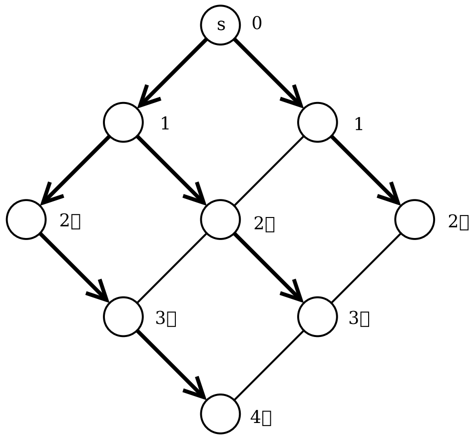

Figure 1: Example BFS Tree. Bold directed edges indicate tree edges that point from parent to child. Numbers indicate the distance from source s.

```
def bfs(V, adj, s):
    for each u in V:
        color[u] = WHITE
        d[u] = INFINITY
        p[u] = NIL
    color[s] = GRAY
    d[s] = 0
    Q = {s}
    while Q is not empty:
        u = head of Q
        for each v in adj[u]:
            if color[v] == WHITE:
                color[v] = GRAY
                d[v] = d[u] + 1
                p[v] = u
                append v to end of Q
        remove u from head of Q
        color[u] = BLACK
    return (d, p)
```

Background: Edge Classification

1. Tree Edge – A tree edge is an edge that is in the output tree of some graph search algorithm, such as the breadth first search.
2. Forward Edge – An edge not in the output tree that connects a vertex to a descendent in the output tree.
3. Back Edge – An edge of the original graph that connects a vertex to its ancestor in the output tree.
4. Cross Edge – All other edges. Edges not in the output tree that is neither a forward edge nor a back edge.

Answer

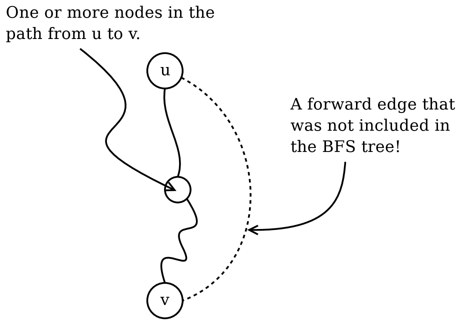

Figure 2: For a forward edge $\langle u, v\rangle$ to exist, the path from u to v in the BFS tree must contain at least one intermediate node. Thus the following condition must hold $d[u] + 1 < d[v]$.

1. There are no forward edges.

   For a forward edge $\langle u, v\rangle$ to exist, $\langle u, v\rangle$ must not be in the BFS tree and u is an ancestor of v in the BFS tree. Thus, $d[u] + 1 < d[v]$ must hold. That is, there must be at least one vertex in the predecessor path from v to u. Otherwise, it would imply that $\langle u, v\rangle$ was in the BFS tree and would not be considered a forward edge.

   Statement 1: When u is the head of the GRAY queue of discovered vertices, then it is made the ancestor of all adjacent vertices that are WHITE. Each is also made GRAY and added to the end of the queue of discovered vertices. For $\langle u, v\rangle$ to exist in the graph but not be included in the BFS tree, v could not have been WHITE when u was the head of the queue. Which means that v was either BLACK or GRAY.

   Statement 2: If v was BLACK, then v has been visited before u. Thus, $d[v] \leq d[u]$. So u cannot be an ancestor of v.

   Statement 3: If v was GRAY, then v has been discovered before u became the head of the queue. In fact, v is in the queue behind u when u was the head of the queue. Any new vertices discovered while u is the head of the queue will have a distance of $d[u] + 1$. But since v was in the queue before any of these newly discovered nodes, $d[v] \leq d[u] + 1$.

   Thus, whether v was BLACK or GRAY when u was the head of the queue, none of the conditions allow forward edges to exist. Proving that there are no forward edges in a BFS tree.

2. For each tree edge $\langle u, v\rangle$, we have $d[v] = d[u] + 1$.

   Following the BFS tree construction, if $\langle u, v\rangle$ is a tree edge, then $d[v] = d[u] + 1$, because u is a parent of v in the BFS tree.

3. For each cross edge $\langle u, v\rangle$, we have $d[v] \leq d[u] + 1$.

From statement 3, any edge $\langle u, v\rangle$ excluded from the BFS tree must have $d[v] \leq d[u] + 1$.

4. For each back edge $\langle u, v\rangle$, we have $0 \leq d[v] \leq d[u]$.

   If $\langle u, v\rangle$ is a back edge, then v must be an ancestor of u. So $d[v] < d[u]$. Since v may very well be the source s, $0 \leq d[v]$.

Question 2

There are two well-known algorithms for finding minimum spanning trees. Point out the difference between Prim's algorithm and Kruskal's algorithm in terms of the construction of the MST tree.

The construction of an MST using Prim's algorithm begins from a single vertex. From here, the tree is expanded until it covers all the vertices in the graph. In other words, there is only one partial tree during the construction of the MST from the very beginning to the end. However, Kruskal's algorithm starts from a forest of $|V|$ single vertex trees. At each step, it merges two trees in the forest, continuing until there is only one tree left; the MST.

Question 3

Assume that the weight assigned to each edge in graph $G = (V, E)$ is distinct. Let e be a maximum-weighted edge on a cycle of $G = (V, E)$. Show that a minimum spanning tree in $G' = (V, E - \{e\})$ is also a minimum spanning tree in G.

Let T' be a minimal spanning tree of G'. We claim that T' is also a minimal spanning tree of G. The only difference between G and G' is that G has the additional edge e. Thus, our claim will only be wrong if the minimal spanning tree of G contains the edge e. Let T be such a tree. Removing $e = \langle u, v\rangle$ from T disconnects the tree into two subtrees, one containing u and the other containing v. If there exist an edge of lower weight than e that connects the two subtrees, then T was not a MST!

Since e is part of a cycle, then by including e in T, at least one of the other edges in the cycle must have been excluded from T (because there are no cycles in trees). And we know that any edge of the cycle not used in T has a lower weight than e (because the weights are distinct). In other words, there exist an edge of lower weight than e that connects the two subtrees. Thus, any tree containing e cannot be a MST of G. Removing e from consideration, gives us G'. Hence, any MST of G' is also a MST of G.

Question 4

Given a directed weighted graph $G = (V, E)$ with source s, under what circumstances should the Bellman-Ford algorithm be used, instead of Dijkstra's

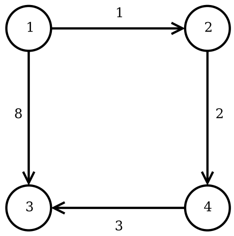

Figure 3: Example Graph

algorithm, to solve the single-source shortest paths problem with source s.

When there exist a negative edge in G; because Dijkstra's algorithm doesn't work for graphs with negative edges.

Question 5

Given a directed weighted graph $G = (V, E)$, let $d(v_i, v_j)$ be the distance in G from v_i to v_j in terms of the minimum number of edges used in the path, $v_i \in V$ and $v_j \in V$. Define the diameter D of G as $D = max_{v_i \in V, v_j \in V}\{d(v_i, v_j)\}$. What is the time complexity for finding all pairs of shortest paths in G if the repeated squaring approach is applied?

Background: Matrix Multiplication for Graphs

Using the graph depicted in Figure 3, let W be a weighted adjacency matrix of the graph G. Let the matrix $M^{(k)}$ contain the all pairs shortest paths of G of **at most** k edges.

$$W = \begin{pmatrix} 0 & 1 & 8 & \infty \\ \infty & 0 & \infty & 2 \\ \infty & \infty & 0 & \infty \\ \infty & \infty & 3 & 0 \end{pmatrix} = M^{(1)}$$

Multiplying $M^{(1)} \times W$ will give us $M^{(2)}$. This is the shortest path between any pair of vertices using **at most two edges**.

$$M^{(2)} = \begin{pmatrix} 0 & 1 & 8 & \mathbf{3} \\ \infty & 0 & \mathbf{5} & 2 \\ \infty & \infty & 0 & \infty \\ \infty & \infty & 3 & 0 \end{pmatrix}$$

Thus, in order to find all pairs of shortest paths in G, we need to find $M^{(n-1)}$; because that would give us all pairs shortest paths of G of at most $n-1$ edges. Hence, we calculate: $M^{(1)}, M^{(2)}, M^{(3)}, \ldots, M^{(n-1)}$, where $M^{(i)} = M^{(i-1)} \times W$.

With repeated squaring, we only need to calculate:

$$M^{(1)}, M^{(2)}, M^{(4)}, \ldots, M^{(2^{\lceil \log_2 (n-1) \rceil})},$$

where $M^{(2i)} = M^{(i)} \times M^{(i)}$.

Each multiplication is $O(n^3)$ time. Since the number of multiplication is $O(\log_2 n)$, the runtime of repeated squaring is $O(n^3 \log_2 n)$. If D is given, then the runtime is $O(n^3 \log_2 D)$.