WA1: Om Khadka

1. **Shortest Path Composition (10 points)** Consider a graph $G = (V, E, w : E \to \mathbb{R}^{>0})$ where V is a set of vertices, E is a set of (directed) edges, and w is a *weight function* that maps edges to weights where each weight is > 0. Let us define a path p to be a sequence of edges where the destination vertex of one edge is the source vertex of the next edge (if the next edge exists). Let us define the *cost* of an path the traditional way, i.e. the cost of a path p is the sum of the edge weights in p:

$$cost(p) = \sum_{e \in p} w(e)$$

Show that if we know a shortest path $p^* = a \overset{x}{\rightsquigarrow} b$ from vertex a to vertex b has cost x. If we know p^* passes through intermediary vertex c, then let $p_1 = a \overset{y}{\rightsquigarrow} c$, and let $p_2 = c \overset{z}{\rightsquigarrow} b$. Show that if $p^* = p_1 \cup p_2$, then p_1 is a shortest path from a to c, and that p_2 is a shortest path from c to b.

Answer:

The question will be solved by contradiction. Our new assumption will be:

If p^* is the shortest path from $a \to b$ with cost x, then p_1 isn't the shortest path from $a \to c$, with cost y OR p_2 isn't the shortest path from $c \to b$, with cost z.

Solving for p_1 is shown below:

$p_1^* = a^* \to c^*$, w/ cost y^*, shortest path from a^* to b^*	Since p_1 isn't the shortest path, there exists another path, p_1^*, that's shorter than p_1.
$p_1 = y$ $p_1^* = y^*$ $y^* < y$	From the given fact that we assume that p_1 is a longer path than p_1^*.
$p^{**} = y^* + z$ $p^* = y + z$	From given and the previous assumption.
$y^* + z < y + z$	Deduced from the previous.

A contradiciton forms here, as p^* should be shortest path, not p^{**}. And by symmetry, the same logic will be true for p^2. Therefore p_1, p_2 MUST be the shortest paths from $a \to c, c \to b$.

1. **Question 2: Iterative Deepening (10 points)** Consider an unweighted (e.g. a constant positive weighted) graph $G = (V, E)$ where V is a set of vertices and E is a set of (directed) edges. Recall that the expansion of all simple paths in a graph forms a tree, which is then expanded via a graph traversal algorithm. Recall that the way DFS traverses this expansion tree is by drilling down a path until it reaches a leaf vertex, then backing off to the parent and trying another branch. Also recall that *Depth-Limited* DFS (DLDFS) is just like the normal DFS algorithm except it is configured via a hyperparameter called the "depth" of the search. This value (a positive integer) provides an upper bound on the number of edges a path can be expanded before DFS arbitrarily backs off (regardless of whether the current vertex is a leaf or not).

   The *Iterative Deepening* algorithm is a single-source shortest-path algorithm that uses DLDFS as a subroutine. Iterative Deepening repeatedly calls DLDFS where each call is given a different depth limit: the first call to DLDFS uses a depth limit of 1, the second call to DLDFS uses a depth limit of 2, the third call to DLDFS uses a depth limit of 3, etc. Iterative Deepening continues to call DLDFS until a path to all vertices have been found (from the source vertex), or until it is determined any unreached vertices are no-longer reachable from the source node. Prove that when Iterative Deepening returns a path from the source to any other vertex, that this path is the shortest path between that pair of vertices.

   Answer:

   The statement 'Iterative Deepening (ID) returns a path from source vertex s to vertex t, that path will be the shortest' can be solved with induction.

   For the base case of $d = 1$:

$DLDFS(s, 1)$ explores all edges from s.	$DLDFS$ with a limit of 1 can only traverse one edge from the source.
If t is adjacent to s, $s \to t$ is a valid path with length of 1.	It's an unweighted graph, and only 1 edge away from the source vertex.

   For the inductive step, we solve the statement 'For all $k < d$, if $DLDFS(s, k)$ finds a path to v, then $s \to v$ has length k and is the shortest path to v'.

Suppose $DLDFS(s, d)$ finds a path p from $s \to t$.	Inductive step, with d being the depth limit.
$length(p) = d$	If a shorter path existed, it would've been returned from a previous call of $DLDFS$.
Suppose that there's a shorter path, p', with length $l < d$.	Contradictory assumption
Then, $DLDFS(s, l)$ would have found p'.	Proving how the contradictory statement is indeed contradictory. In this case, t it wasn't found until depth d, forming the contradiction.
No such shorter path exists.	p is the shortest path.
For all $d \leq 1$, if a path is found at depth d, it is the shortest path.	The conclusion of this proof and, thus, proving the initial statement.

1. **Question 3: Graph Diameters and the Longest-Shortest Path (10 points)** The diameter of an undirected, unweighted graph $G = (V, E)$ is defined to be the solution to the following optimization problem:

$$d = \max_{\substack{(u,v)\in V \\ u\neq v}} \min_{k\in\mathbb{Z}^{\geq 0}} cost(u \overset{\text{k}}{\rightsquigarrow} v)$$

which in english is "the largest of all the shortest paths in G" where $u \overset{\text{k}}{\rightsquigarrow} v$ is a path from vertex u to vertex v using exactly k edges, and the cost of a path is the sum of the edge weights used in that path (in our case $cost(u \overset{\text{k}}{\rightsquigarrow} v) = k$). Prove that the diameter of any unweighted, undirected graph is at most $|V|$ (the number of vertices in the graph).

Answer:

1. $G = (V, E)$	Given definition of the graph.
2. $d = \max_{\substack{(u,v)\in V \\ u\neq v}} \min_{k\in\mathbb{Z}^{\geq 0}} \text{cost}(u \overset{\text{k}}{\rightsquigarrow} v)$	Definition of diameter.
3. For any distinct u, $v \in V$, let $P =$ the shortest path from $u \to v$.	Consider any pair of distinct vertices.
4. P is simple.	From 3, P does not contain any cycles, as removing those cycles would create a shorter path, which would lead to a contradiction.
5. Let $k = \#$ of vertices in P. # of edges will be $k - 1$.	From 4, P is simple, and this is the definition of a simple path.
6. $k \leq \|V\|$	From 4, 5, a simple path can't be longer than the # of vertices it contains.
7. # of edges in $P \leq \|V\|$	From 6, # of edges is one less than vertices.
8. $\text{dist}(u, v) \leq \|V\| - 1$ for all u, v	From 7, the shortest path length can be at most $\|V\| - 1$.
9. $d = \max_{u,v} \text{dist}(u, v) \leq \|V\| - 1$	From 8, now the maximum over all potential paths can only be at most $\leq \|V\| - 1$.

This proves that, at a maximum, the diameter of an unweighted, undirected graph can only go up to $|V| - 1$. This also implies that $d \leq |V|$.

1. **Question 4: Dijkstras Algorithm and Shortest Paths (20 points)** Consider a graph $G = (V, E, w : E \to \mathbb{R}^{>0})$ where V is a set of vertices, E is a set of (directed) edges, and w is a *weight function* that maps edges to weights where each weight is > 0. If we were to search for the shortest path between a pair of vertices in such a graph, we would be forced to use dijkstras algorithm as BFS and Iterative Deepening assume a constant weighting. Prove that when dijkstras algorithm returns a path from the source to any other vertex, that this path is the shortest path between that pair of vertices.

Answer:

We begin by establishing some fundamentals about Dijkstra's algorithm itself. Note, the format $d[x]$ means the weight of the supposed vertex x.

1. Dijskta's algorithm, on the first iteration, holds an empty set, $S =$. Because of this, the algorithm's invariant is true.	Establishing the fact that Dijsktra's algorithm is primarily based on an invariant.
2. $d[s] = 0$, while $d[v] = \inf$	$d[s]$ is correct, as it's just the distance from traveling from the origin to the origin (just standing still). $d[v]$ being infinite represents that they're yet to be known.
3. Let u be the vertex extracted from a priority queue (a vertex with the current minimum $d[u] \in S$). u is added to S. Then, $d[u]$ is the shortest path weight from $s \to u$.	Invariant assumption. We hold that this is true for all previous iterations, and now will have to show how adding u to S preserves the invariant.
4. The way the shortest distance is calculated is from the following line: if $d[v] > d[u]+w(u, v)$ then $d[v] = d[u]+w(u, v)$	This ensures that the calculated shortest distance, $d[v]$, will always be $\geq$ the true shortest distance.

We can now solve by contradiction:

5. Suppose $d[u]$ is not the shortest path weight from $s \to u$. Thus, there exists another path, P, from $s \to u < d[u]$	Contradiction assumption
6. $s \in S$, $u \notin S$, path P must exit S at some point.	The path starts in S (which also has s). It end eventually ends outside of S (at u). Therefore, it must have a transition edge where it leaves S.
Let (x, y) be the 1st edge along P where $x \in S$ and $y \notin S$.	This is more concretely defining our contradictory assumption of $d[u]$ being wrong.
Let $P_s \to x$ be the subpath of P from $s \to x$.	The path of P is AT LEAST as long as any cut version of itself, as all edge weights are positive.
7. Let $f(x)$ be the true shortest path weight from $s \to v$.	This is just the result of the calculations that Dijkstra's algorithm does and assuming that the invariant is true for x.
From 5, $f(u) < d[u]$	
8. The weight of $P_s \to x + w(x, y \leq)$ the entire weight of P (which goes to u).	Chaining inequalities
From 6, $f(x) + w(x, y)f(u)$	Combining the contradiction assumption with 10.
9. $x \in S$. By 2, since $x \in S$, $d[x] = f(x)$. From 4, after x was added to S, we calculated all edges, including (x, y).	In this iteration, we chose u, not y, to extract from the priority queue. The algorithm always extracts the vertex with the smallest current distance.
Therefore, $d[y] \leq d[x] + w(x, y) = f(x) + w(x, y)$	Our initial assumption that $d[u]$ wasn't the shortest path weight has to be false from this. So, $d[u] = f(u)$, and thusly from 4, the invariant that all $u \in S$ have correct $d[u]$ holds.
From 7, this also means that $d[y] \leq f(x) + w(x, y)$.	
10. From 6,7, $d[y] \leq f(x) + w(x, y) \leq f(u)$	
11. From 5,8, $d[y] \leq f(u) \leq d(u)$	The only way to resolve the contradiction is to reject the initial contradiction assumption. Therefore $d[u]$ must be correct when added to S.
12. From 10, $d[u] \leq d[y]$	
13. Contradiction at 12	

Thus, this proves that when Dijstrka's algorithm return the shortest path weight from $s \to u$, $d[u]$, that it is indeed the shortest path and is correct.

For closure, we can also prove how the invariant ends, too:

13. Algorithm ends when $S = V$	The algorithm continues until all vertices in the priority queue have been processed.
14. Since the invariant was true each time a vertex was added to S, and from 13, thusly for each vertex $v \in V$, $d[v]$ is the weight of the shortest path from $s \to v$.	The invariant condition was true for every step of the loop, and is now true for the entire graph.