Solutions to Homework 6 Practice Problems

CS6515 Summer 2026

[DPV] Problem 4.14 – Shortest Path through a given vertex

Problem Statement

You are given a strongly connected directed graph G = (V,E) with positive edge weights along with a particular node $v_0 \in V$. Give an efficient algorithm for finding shortest paths between *all pairs of nodes*, with the one restriction that these paths must all pass through v_0.

(a) Algorithm:

Run Dijkstra's algorithm from v_0 to get all distances from v_0 to all vertices in V. Reverse the graph and run Dijkstra's again from v_0 - the output corresponds to the distances of the shortest path from all other vertices to v_0 in the original graph. For any pair of vertices $u, w \in V$, the shortest distance from u to w using a path through v_0 will be the sum of the two distances, dist$[u]$ in the reverse graph and dist$[w]$ in the original graph. Any single path $u \rightsquigarrow v_0 \rightsquigarrow w$ may be recovered using the `prev[]` arrays which result from each run of Dijkstra.

Note

While the problem asks for all pairs shortest paths, typically problems in this course would request a structure or function from which those paths could be derived, or else simply calculate the distances as above. We demonstrate this solution accordingly, showing how `dist[]` and `prev[]` can be used depending on the scenario. See runtime note for more.

(b) Justification of Correctness:

This algorithm works because our graph is strongly connected, meaning that a path exists between every pair of vertices. We also know that Dijkstra's, given a starting vertex and non-negative edge weights, will find the shortest path from that starting point to every other vertex. Reversing a directed graph essentially reverses the direction of the path between any pair of vertices. So, the shortest path from u to w which includes v_0 is the combination of the shortest path from u to v_0 in the reversed graph plus the shortest path from v_0 to w in the original graph.

(c) Runtime Analysis:

Each round of Dijkstra's takes $O((m+n)\log(n))$ which simplifies to $O(m \log n)$ since the graph is strongly connected. Building the reverse graph takes $O(n+m)$ time. Obtaining the necessary structures (`dist[]` and `prev[]`) from which the shortest paths can be obtained is thus $O(m \log n)$.

Note

For calculating pairwise distance or recontructing the path, it takes $O(n^2)$ to iterate all pairs. Calculating the distance is constant time using `dist[]` for an overall $O(n^2)$ runtime. Reconstructing each path using `prev[]` takes $O(n)$, for an overall $O(n^3)$ runtime.

[DPV] Problems 5.1, 5.2 (Practice fundamentals of MST designs)

5.1

(a) The cost is 19

(b) There are 2 possible MSTs

(c)

Edge included	Cut
AE	{A} and {B,C,D,E,F,G,H}
EF	{A,E} and {B,C,D,F,G,H}
BE	{A,E,F} and {B,C,D,G,H}
FG	{A,B,E,F} and {C,D,G,H}
GH	{A,B,E,F,G} and {C,D,H}
CG	{A,B,E,F,G,H} and {C,D}
GD	{A,B,C,E,F,G,H} and {D}

5.2 (a) - Prim

Vertex incl.	Edge included	(accum)	Cost A	B	C	D	E	F	G	H
{}		0	**0/-**	∞/-	∞/-	∞/-	∞/-	∞/-	∞/-	∞/-
A		0		**1/A**	∞/-	∞/-	4/A	8/A	∞/-	∞/-
B	AB	1			**2/B**	∞/-	4/A	6/B	6/B	∞/-
C	BC	3				3/C	4/A	6/B	**2/C**	∞/-
G	CG	5				**1/G**	4/A	1/G		1/G
D	GD	6					4/A	**1/G**		1/G
F	GF	7					4/A			**1/G**
H	GH	8					**4/A**			
E	AE	12								

5.2 (b) - Kruskal

Here are the values for the parent pointer π at each iteration of Kruskals. From this you should be able to deduce the disjoint-sets.

Union	Values of π for each vertex
Start	`[ A, B, C, D, E, F, G, H ]`
(A,B)	`[ B, B, C, D, E, F, G, H ]`
(F,G)	`[ B, B, C, D, E, G, G, H ]`
(D,G)	`[ B, B, C, G, E, G, G, H ]`
(G,H)	`[ B, B, C, G, E, G, G, G ]`
(C,G)	`[ B, B, G, G, E, G, G, G ]`
(B,C)	`[ B, G, G, G, E, G, G, G ]`
(A,E)	`[ G, G, G, G, G, G, G, G ]`

[DPV] Problem 5.9 (Statements about MSTs)

(a) FALSE. Consider a graph where a vertex is adjacent to a single edge.

(b) TRUE. Consider the order in which edges would be processed by Kruskal's.

(c) TRUE. A minimum weight edge would be a candidate for at least one possible MST.

(d) TRUE. This is assured by the *Cut Property*.