IV. Section (iv) Dijkstra's Algorithm

One of the best-known & most important algorithms in graph theory / complex networks relates to finding the most efficient (i.e. shortest) path between two vertices.

The problem, for instance, of finding the most efficient route between two cities (i.e. the "Google Maps" problem) is something we have each, inevitably, encountered.

A less explicit (but nonetheless equally ubiquitous) instance of the "shortest path" problem pertains to message routing on the internet. In this way, we can think of vertices as representing computers (or servers), so that neighboring vertices are in some sense "linked" via their ability to send/receive messages. The commonly deployed link-state routing protocol uses a shortest path variant algorithm (i.e. Dijkstra's). In the exposition that follows, we present a version of **Dijkstra's Algorithm** (1959) that simultaneously solves the shortest path problem for all vertices in a graph with respect to a "root" vertex.

Formal Algorithm (Dijkstra):

Let G be an undirected, simple, weighted graph. Edge weights are required to be non-negative. Consider a vertex u. We introduce the following sets & labels:

- Let $S_T(u)$ be the set of vertices to which a shortest path from vertex u has been found after step T.

- Each vertex v is assigned a label $L(v) = (L_1(v), L_2(v))$ in which $L_1(v)$ is the vertex preceding v in the shortest (u,v)-path found so far, and $L_2(v)$ is the total weight of that path.

- Let $R_T(u) = S_T(u) \cup \bigcup_{v \in S_T(u)} N(v)$, with $N(v)$ denoting the neighbor sets of v. Put another way, $R_T(u)$ consists of vertices in $S_T(u)$ & their neighbors.

Notation Summary:

$S_T(u)$: set of vertices with known shortest path from u.

$(L_1(v), L_2(v))$: label for previous vertex in path (left) & path length (right).

$R_T(u)$: set of "known" vertices & their neighbors.

The Algorithm (in 3 steps):

(1) **Initialization.**

Set: $T \to 0$, $S_0(u) = \{u\}$ and for all $v \in V(G)$,

$$L(v) \to \begin{cases} (u,0) & \text{if } v = u \\ (-,\infty) & \text{else} \end{cases}$$

(2) For every vertex $y \in R_T(u) \setminus S_T(u)$, consider the vertices $N'(y)$ that are neighbors of y that lie in $S_T(u)$, i.e., $N'(y) = N(y) \cap S_T(u)$. Select $x \in N'(y)$ for which $L_2(x) + w(\langle x,y \rangle)$ is minimal. Finally, set $L(y) \to (x, L_2(x) + w(e))$

(3) Let $z \in R_T(u) \setminus S_T(u)$ for which $L_2(z)$ is minimal. Set $S_{T+1}(u) \to S_T(u) \cup \{z\}$. If $S_{T+1}(u) = V(G)$ we are done. Otherwise, update T: $T \to T+1$, compute $R_T(u)$ & repeat (2).

Algorithm Summary:

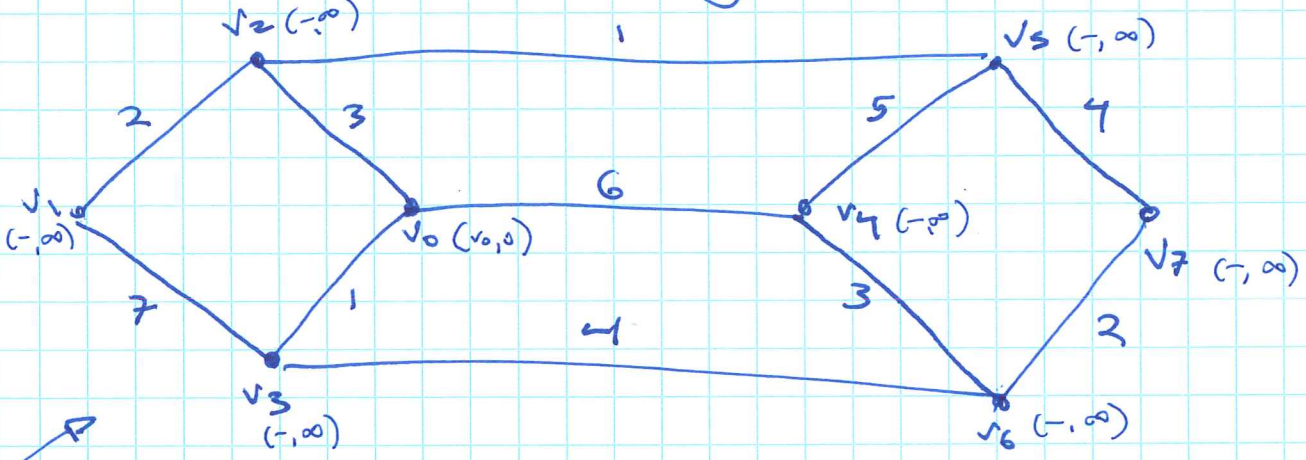
(1) Initialize parameters, set labels $L(v)$ to ∞ to all vertices except u.

(2) Update labels based on "minimal neighbors."

(3) Update set of "known vertices" $S_T(u)$, T.

We now apply Dijkstra's algorithm to a directed graph. Following this example we present a succinct, pseudo-code variant of the algorithm.

Ex. Let G be the following digraph.



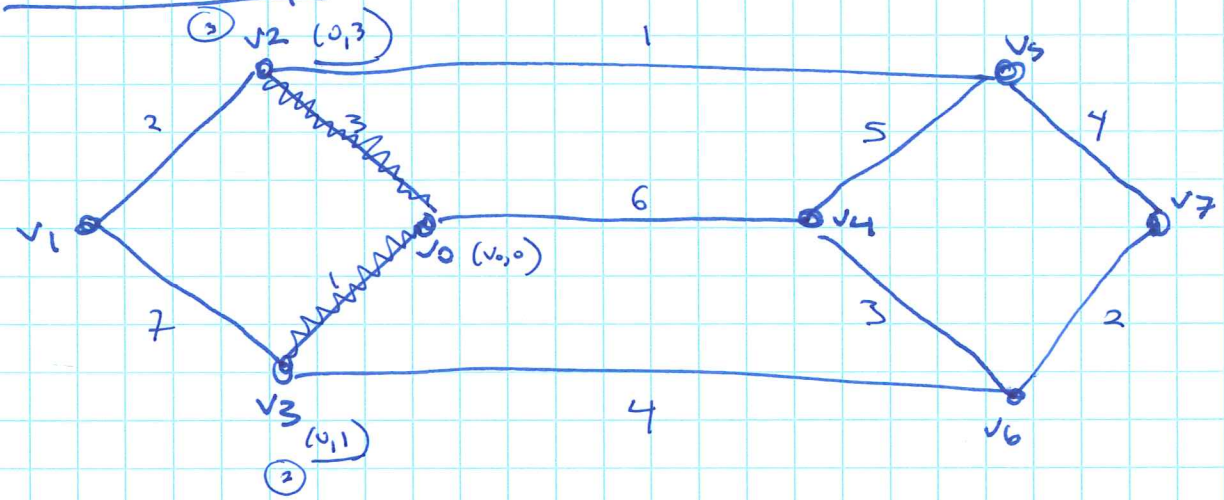
Step 1: Set $t \to 0$, $S(v_0) = \{v_0\}$, label each vertex in G with appropriate $L(v) = (L_1(v), L_2(v))$ label.

Step 2: We consider the vertex closest to the "root" vertex v_0. This vertex is $\boxed{v_3}$. Next we add v_3 to the set $S(v_0) = \{v_0, v_3\}$; note that the label for v_3 is accordingly updated to: $(1, 4)$.

(1): distance from v_0; (2): previous vertex in path.

Step 3: Continuing, we identify the vertex closest to v_0 that can be reached from any vertex in $S(v_0)$. This vertex is $\boxed{v_2}$, which is then added to $S(v_0) = \{v_0, v_3, v_2\}$. Next we update the label for v_2 to $(0, 3)$.

After 3 steps:



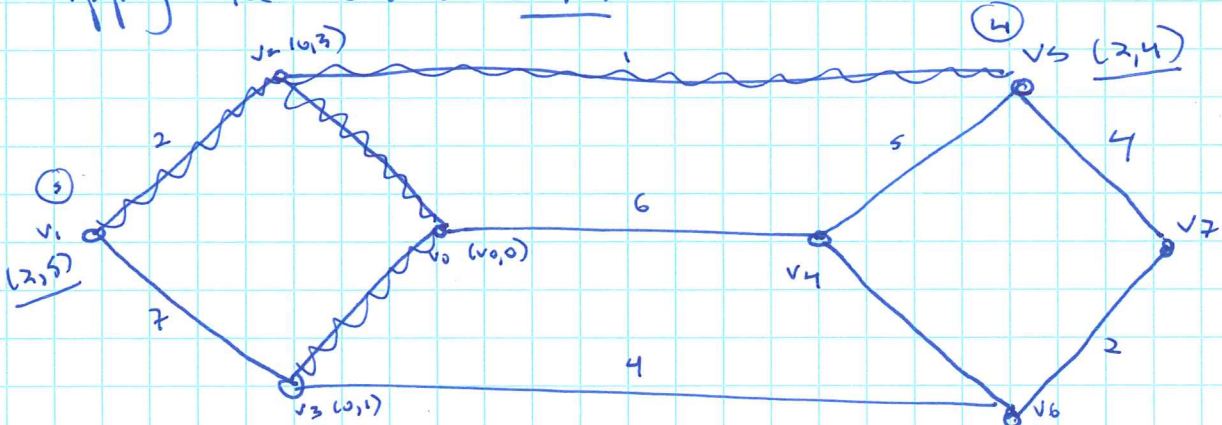
Step 4: The next vertex to add is $\boxed{v_5}$: with $S(v_0)$ now set to: $\{v_0, v_2, v_3, v_5\}$, it receives label: $(2,4)$.

Step 5: At this juncture the set of "reachable" vertices (neighbor-wise) for $S(v_0) = \{v_0, v_2, v_3, v_5\}$ consists of the set: $\{v_1, v_4, v_6, v_7\}$. The respective minimum distance from v_0 for each of these vertices is as follows:

v_1 - (5 via v_2), v_4 - (6 via v_0), v_6 - (5 via v_3),

v_7 - (8 via v_5). Either minimum-reachable vertex will do; we choose $\boxed{v_1}$, making $S(v_1) = \{v_0, v_1, v_2, v_3, v_5\}$ & apply the label: $(2,5)$.

After 5 steps

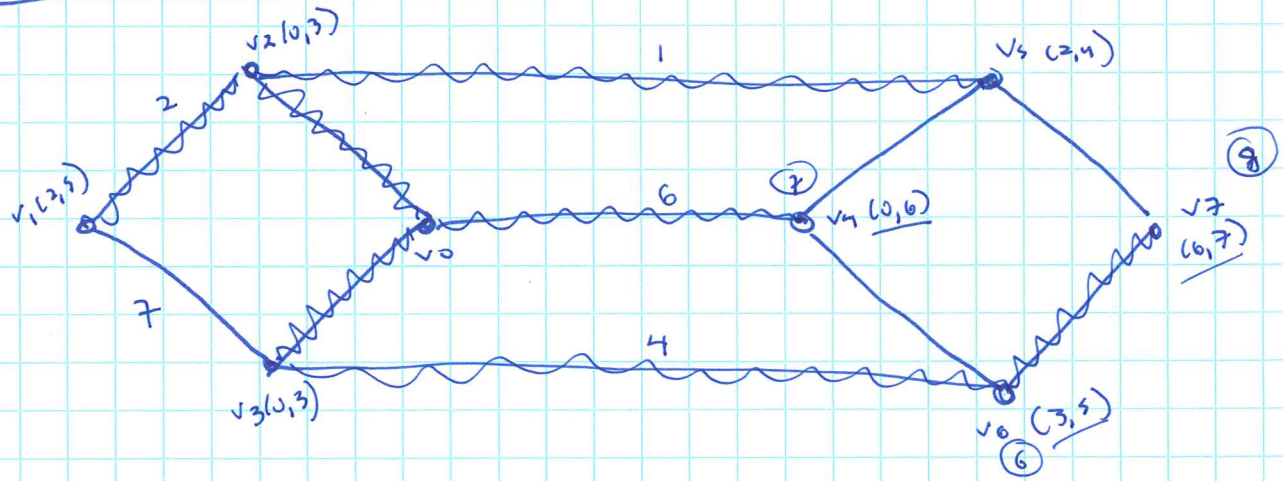


Step 6: As previously mentioned, vertex v_6 is also now a minimal distance from v_0, so we add it to the queue, giving: $S(v_0) = \{v_0, v_1, v_2, v_3, v_5, v_6\}$, applying the label: $(3,5)$.

Step 7: Only two unvisited vertices remain: $\{v_4, v_7\}$. The minimum distance to each is: v_4 - (6 via v_6), v_7 - (7 via v_6), respectively; consequently we add v_4 to $S(v_0)$ with label: $(0,6)$.

Step 8: Lastly, we add v_7 with label: $(6,7)$. This completes the algorithm.

Result of Dijkstra's algorithm:



Note that Dijkstra's algorithm results in a spanning tree rooted at v_0. We now know the minimum distance between v_0 and any other vertex; in addition this minimum path can be "read" from the Dijkstra encoding.

Pseudo-code version of The Algorithm:

$S(u) \to \{u\}$

$L(u) \to (u, 0)$, else $L(v) \to (-, \infty)$

while $S(u) \neq V(G)$ **do:**

$R(u) \to S(u) \cup \bigcup_{v \in S(u)} N(v)$

for all $y \in R(u) \setminus S(u)$ **do:**

for all $x \in N(y) \cap S(u)$ **do:**

if $L_2(x) + w(\langle x, y \rangle) < L_2(y)$ **then:**

$L(y) \to (x, L_2(x) + w(\langle x, y \rangle))$

end if

end for

end for

select $v \notin S(u)$ where $L_2(v)$ is minimal

$S(u) \to S(u) \cup \{v\}$

end while.