Dijkstra's Algorithm

- The Problem:

  Given a graph whose edges have non-negative weights, find the least-weight path (a.k.a. the shortest path) from vertex v_0 to each other vertex in the graph.

- The Algorithm:

  1. `Let` $t = 0, S = \{v_0\}$`, and` $d_i = \begin{cases} 0 & \texttt{if } i = 0 \\ \infty & \texttt{if } i > 0 \end{cases}$
  2. `For each` $v_i \in V(G) \setminus S$ `with an edge from` v_t `to` v_i`,`
     `if` $d_t + wt(v_t, v_i) < d_i$ `then let` $d_i = d_t + wt(v_t, v_i)$ `and let` $p_i = v_t$`.`
     `Let` $v_t \in V(G) \setminus S$ `be a vertex such that` $d_t \leq d_i \ \forall v_i \in V(G) \setminus S$ `(i.e.` t `gets updated).`
     `Let` $S = S \cup \{v_t\}$`.`
  3. `Repeat (2) until either` $V(G) \setminus S = \emptyset$ `or` $d_i = \infty \ \forall v_i \in V(G) \setminus S$`.`

  What the algorithm does is build a set, S, of vertices for which the shortest distance from v_0 is known; the distance for each vertex not in S is considered tentative until the vertex gets added to S. At each iteration, the set S has one vertex added to it, namely a vertex of $V(G) \setminus S$ having shortest tentative distance from v_0 (this is the vertex selected since its tentative distance can not be improved any further).

  The algorithm terminates either when the shortest distance to each vertex is determined, or when $V(G) \setminus S$ consists entirely of vertices which cannot be reached from v_0 (as would occur in a disconnected graph).

- The Algorithm's Efficiency:

  Let $n = |V(G)|$ denote the number of vertices in the graph. We determine the algorithm's efficiency by determining the number of operations (such as assignments, additions, and comparisons) required by the algorithm, and then expressing this number as a function of the size of the input to the algorithm (i.e. as a function of n).

  Step (1) requires $1 + 1 + n$ operations.

  The iteration of steps (2) and (3) occurs $n - 1$ times. Each iteration requires several operations:

  - up to $n + 2 + 3$ operations (comparing at most n vertices with v_t to find qualifying vertices, followed by 2 operations in the `if` statement, followed by 3 operations if the `if` statement is satisfied.
  - up to $n + 2$ operations are involved in updating t and S.
  - up to $1 + n$ operations are involved in testing for the algorithm's termination

We obtain an upper bound on the number of operations:

$$\begin{aligned}(2+n)+(n-1)((n+5)+(n+2)+(1+n)) &= 2+n+(n-1)(3n+8) \\ &= 3n^2+6n-6\end{aligned}$$

Observe that the number of operations is a polynomial in n (as opposed to being an exponential function, such as 2^n). Hence the algorithm is said to be a polynomial-time algorithm, since the time (i.e. the number of operations) required for the algorithm to complete is polynomially bounded by the size of the input.

Also note that when n is sufficiently large the $3n^2$ term dominates. And in the $3n^2$ term, it is the n^2 which dominates. We thus say that the algorithm's running time is $O(n^2)$ ("big oh of n^2").