Dijkstra's Algorithm

- **Dijkstra's algorithm**: finds shortest (minimum weight) path between a particular pair of vertices in a weighted directed graph with *nonnegative* edge weights
  - Solves the "one vertex, shortest path" problem

- **Basic algorithm concept**:
  - For each vertex, keep track of the currently known best way to reach it (distance, previous vertex)
  - Iterate until best way is found
Example Application

- Dijkstra's algorithm can be used to find the shortest route between one city and any other
  - vertices represent cities
  - edge weights represent driving distances between pairs of cities connected by a direct road
Dijkstra pseudocode

Dijkstra(v1, v2):
    for each vertex v:  // Initialization
        v's distance := infinity.
        v's previous := none.
    v1's distance := 0.
    List := {all vertices}.

    while List is not empty:
        v := remove List vertex with minimum distance.
        mark v as known.
        for each unknown neighbor n of v:
            dist := v's distance + edge (v, n)'s weight.

            if dist is smaller than n's distance:
                n's distance := dist.
                n's previous := v.

        reconstruct path from v2 back to v1,
        following previous pointers.
Example: Initialization

Distance(source) = 0

Pick vertex in List with minimum distance.

Distance (all vertices but source) = \(\infty\)
Example: Update neighbors' distance

Distance(B) = 2
Distance(D) = 1
Example: Remove vertex with min. distance

Pick vertex in List with minimum distance, i.e., D
Example: Update neighbors

Distance(C) = 1 + 2 = 3
Distance(E) = 1 + 2 = 3
Distance(F) = 1 + 8 = 9
Distance(G) = 1 + 4 = 5
Example: Continued...

Pick vertex in List with minimum distance (B) and update neighbors

Note: distance(D) not updated since D is already known and distance(E) not updated since it is larger than previously computed.
Example: Continued...

Pick vertex List with minimum distance (E) and update neighbors

No updating
Example: Continued...

Pick vertex List with minimum distance (C) and update neighbors

Distance(F) = 3 + 5 = 8
Example: Continued...

Distance(F) = min(8, 5+1) = 6

Pick vertex List with minimum distance (G) and update neighbors
Pick vertex not in S with lowest cost (F) and update neighbors
Correctness

- Dijkstra’s algorithm is a greedy algorithm
  - Makes choices that currently seem the best
  - In general, locally optimal does not always mean globally optimal (think hill-climbing), but in this case, it is.

- Correct because maintains following two properties:
  - For every known vertex, recorded distance is shortest distance to that vertex from source vertex
  - For every unknown vertex \( v \), its recorded distance is shortest path distance to \( v \) from source vertex, considering only currently known vertices and \( v \)
If the path to $v$ is the next shortest path, the path to $v'$ must be at least as long (if it were shorter, it would be picked over $v$). Therefore, any path through $v'$ to $v$ cannot be shorter!
The simplest implementation of the Dijkstra's algorithm stores vertices in an ordinary linked list or array

- Good if the graph is dense (lots of edges: \(|E| \sim O(|V|^2)|\))

- Initialization (setting to infinity, unknown) \(O(|V|)\)
- While loop \(O(|V|)\)
  - Find and remove min distance vertex \(O(|V|)\)
- Potentially \(O(|E|)\) distance updates
  - Update costs \(O(1)\)
- Reconstruct path \(O(|E|)\)

- Total time \(O(|V|^2| + |E|) = O(|V|^2)\)
Time Complexity: Priority Queue

- For sparse graphs (i.e. $|E| \sim O(|V|)$), Dijkstra's implemented more efficiently by priority queue

- Initialization $O(|V|)$ using $O(|V|)$ buildHeap
- While loop $O(|V|)$
  - Find and remove min distance vertex $O(\log |V|)$ using deleteMin
- Potentially $O(|E|)$ distance updates
  - Update costs $O(\log |V|)$ using decreaseKey
- Reconstruct path $O(|E|)$

- Total time $O(|V|\log |V| + |E|\log |V|) = O(|E|\log |V|)$
- $|V| = O(|E|)$ assuming a connected graph
Exercise

- Use Dijkstra's algorithm to determine the lowest cost path from vertex A to all of the other vertices in the graph. Keep track of previous vertices so that you can reconstruct the path later.