Section 7.4: Dijkstra's Algorithm (Version 2 with heap)

Determines the length of a shortest path from a source vertex **`start`** in a connected weighted graph *G* to each of the other vertices of *G*.

`adj` is an adjacency list representation of the graph
`adj[i]` – pointer to first object in linked list of type `Node`, represents the nodes adjacent to vertex `i`
    `int vertex` – a vertex adjacent to `i` (the label or index of the vertex, e.g., 1, 2, …, n)
    `int weight` – weight of edge between nodes `i` and `vertex`
    `Node next` – reference to next node in linked list
`start` – index (or label) of start vertex
`predecessor` – stores predecessor of each vertex along a shortest path
`h` – min heap of vertices (keyed by weights of shortest path from source), has operations:
    `h.init(key, n)` builds the heap `h` using the values in key (array of size `n`)
    `h.minimum()` returns item in `h` with smallest key
    `h.delete()` deletes item in `h` with smallest key
    `h.isIn(w)` returns true if vertex `w` is in `h` and false otherwise
    `h.keyval(w)` returns the shortest path weight (key value) of vertex `w`
    `h.decrease(w, wgt)` changes the key for vertex `w` to `wgt` (a smaller value) and sifts up to restore heap property

```
dijkstra(adj, start, predecessor)
{
  n = adj.last
  for i = 1 to n
    key[i] = ∞
  key[start] = 0
  predecessor[start] = start
  h.init(key, n)              // builds minheap of vertices by key
  for i = 1 to n              // process each vertex once
  {
    v = h.minimum()           // heap_smallest
    min_cost = h.keyval(v)    // weight of path from start to v
    h.delete()                // heap_delete
    ref = adj[v]
    while (ref != null)       // scan adjacency list of vertex v
    {
      w = ref.vertex          // inspect edge from v to w
      if (h.isIn(w) && (min_cost + ref.weight < h.keyval(w)))
      {
        predecessor[w] = v
        h.decrease(w, min_cost + ref.weight)      // updateKey
      }
      ref = ref.next
    }
  }
}
```

Running time: