Dijkstra’s algorithm, the idea. Maintain three sets of vertices:

- The known, consisting of vertices whose shortest paths have been found.
- The frontier, consisting of those vertices for whom a path, but not necessarily a shortest path, has been found from the starting vertex.
- The unknown, consisting of the rest of the vertices.

The algorithm operates in a few steps:

- At the start, the starting vertex is the only known vertex, and it is in the frontier.
- We make “progress” by selecting the frontier vertex closest to the starting vertex, call it $u$. At this point, we’ve found the shortest path from the starting vertex to $u$.
- We then look at each neighbor of $u$ and add it to the frontier or update its information.

Some general things to think about/talk about:

- What does the method do? Explain what the role of the different parameters is, what is returned and how the method operates.
- Show some examples. Note that the Dijkstra’s algorithm is for weighted graphs. An interesting example is when there are multiple different paths from the starting vertex to a given vertex. Show how the algorithm makes sure that the only the shortest path is found.
What is the running time of the method with respect to $|V|$ the number of vertices and $|E|$ the number of edges?

In the worst case, there are $|V|$ calls to $\text{pop}$ (when the graph is connected) and there are $|E|$ calls to $\text{reduce_priority}$ (either implicitly in $\text{push}$ or explicitly).

Some specific things to think about/talk about:

- What purpose does the priority queue serve?
- What do the three different if statements check?
- In my notes above, I mention Dijkstra’s keeps three different sets. Where are these three sets in the code?
- What is stored in the $\text{parents}$ map?
- When we call $\text{reduce_priority}$ what have we found? What is the purpose of the call?
map<int,int> shortest_paths(int start, const map<int,list<pair<int,int> > > & graph) {
    map<int,int> parents;
    priorityqueue62 frontier;

    parents[start]=start;
    frontier.push(start, 0);

    while (!frontier.is_empty()) {

        int v = frontier.top_serialnumber();
        int p = frontier.top_priority();
        frontier.pop();

        for (the neighbors (n,w) of v)
            if (n == parents[v])
                ; // do nothing
            else if (n is not in the frontier and has not been visited) {
                parents[n] = v;
                frontier.push(n, p + w);
            }else if (p + w < frontier.get_priority(n)) {
                parents[n] = v;
                frontier.reduce_priority(n, p + w);
            }
    } // end while

    return parents;
}