Thao Nguyen and Joshua Nowak

Dijkstra’s Algorithm Report

The graph was implemented as an adjacency list. shortestPath is a method that takes std::string startLabel, std::string endLabel, and std::vector<std::string> &path as arguments, and returns an unsigned long value for the distance of the path. Dijkstra’s algorithm finds the shortest path between two vertices in the following six steps.

**Step 1**: Three unordered\_maps from the Standard Template Library are declared to store relevant data about each vertex. One unordered\_map, named “distances,” stores unsigned long values to record each vertex’s distance from the start vertex. Another unordered\_map, named “visited,” stores bool values to record whether the algorithm has already visited a vertex. A third, named “predecessors,” stores a string label of the previous vertex in the path from the start vertex to the current one. All three unordered\_maps use string keys to access information about a vertex by its label.

**Step 2**: The unordered\_maps “distances” and “visited” are iterated over. To represent infinity as the starting estimate of each vertex’s distance from the start vertex, their values in “distances” are set to the maximum unsigned long value with the function std::numeric\_limits<unsigned long>::max(). Only the start vertex has its distance set to 0 instead, as there is no distance between the start vertex and itself. The bool value for all vertices in “visited” is set to false.

**Step 3**: A priority\_queue from the Standard Template Library is initialized to take pairs of unsigned long values and strings. This priority\_queue will store the distances and labels of vertices; vertices with smaller distances are given greater priority. An unsigned long value of 0 and the string label of the start vertex are pushed to the priority\_queue.

**Step 4**: While the priority\_queue is not empty, the top element is popped from the priority queue. If the vertex with the corresponding label has already been visited (true in “visited”), the loop continues to the next iteration. If not, the current vertex is marked as visited and the method proceeds to Step 5.

**Step 5**: The incident edges of the current vertex are iterated over to access all adjacent vertices. If an adjacent vertex has not been visited, and the “distances” value of the current vertex plus the weight of the incident edge is less than the “distances” value of the adjacent node, then relaxation occurs and the data of the adjacent vertex are pushed to the priority\_queue. In relaxation, the “distances” value of the adjacent vertex is updated to be the “distances” value of the current vertex plus the weight of the incident edge. After relaxation, the adjacent vertex is pushed to the priority queue and its predecessor is updated to be the current vertex. Once all incident edges have been iterated over, the loop repeats from Step 4.

**Step 6**: When the priority\_queue is empty, the loop terminates. To construct the path from the start vertex to the end vertex, “predecessors” is iterated through backwards. The predecessor of the end vertex is pushed to the back of “path,” then the predecessor replaces the end vertex as the current label and has its predecessor accessed. This repeats until the current label is the label of the start vertex. “Path” is then reversed by std::reverse(path.begin(), path.end()) so that it is ordered from from the start vertex to the end vertex, and the “distances” value of the end vertex is returned.

In the case of g.shortestPath("1", "5", path) in the given example, the three unordered\_maps are declared, all vertices have their “distances” values initialized as std::numeric\_limits<unsigned long>::max() and “visited” values initialized as false, the “distances” value for vertex 1 is set to 0, and vertex 1 is pushed to the priority queue.

Vertex 1 is popped from the priority queue and visited. Adjacent vertices 2, 3, and 6 have their distance values updated with the values of their incident edges, are pushed to the priority queue, and have their predecessors initialized as vertex 1. The priority queue is now ordered 2, 3, 6.

Vertex 2 is popped from the priority queue and visited. Adjacent vertex 1’s data does not change because 1 has already been visited. Adjacent vertex 3’s data does not change because the distance value of vertex 2 plus the weight of the incident edge (7 + 10) is not less than the distance value of vertex 3 (9). Adjacent vertex 4 has its distance value updated with the distance value of vertex 2 plus the weight of the incident edge (7 + 15 = 22), is pushed to the priority queue, and has its predecessor initialized as vertex 2. The priority queue is now ordered 3, 6, 4.

Vertex 3 is popped from the priority queue and visited. Adjacent vertices 1 and 2’s data do not change because they have already been visited. For vertex 4 relaxation occurs because the distance value of vertex 3 plus the weight of the incident edge (9 + 11) is less than the distance value of vertex 4 (22). Vertex 4 has its distance value updated to 20 and its predecessor is updated to vertex 3. Relaxation also occurs for vertex 6. Its distance value is updated to 11 and its predecessor is updated to vertex 3. The priority queue is now ordered 6, 4.

Vertex 6 is popped from the priority queue and visited. Adjacent vertices 1 and 3’s data do not change because they have already been visited. Vertex 5 has its distance value updated to 20, is pushed to the priority queue, and has its predecessor updated to vertex 6. The priority queue is now ordered 4, 5.

Vertex 4 is popped from the priority queue and visited. Adjacent vertices 2 and 3’s data do not change because they have already been visited. Relaxation does not occur for adjacent vertex 5. The priority queue now only contains 5.

Vertex 5 is popped from the queue and visited. Adjacent vertices 6 and 4 have already been visited. Now that the priority queue is empty, the loop terminates. Starting from the end vertex and following the predecessor of each vertex, vertices 5, 6, 3, and 2 are pushed to the back of “path.” Vector “path” is then reversed, and the distance value of vertex 5, unsigned long 20 is returned.