Data Structures
Djikstra Algorithm

Cain Susko

Queen's University
School of Computing

April 1, 2022

Finding the Shortest Path

it is useful to know the shortest path in an a graph is useful as it allows us to move as efficiently as possible in a graph. A graph will normally be wieghted if you are asked to find the shortest path, the objective is to find the path from a to b with the smallest sum of weights.
One can also be taked to find the shortest path between any 2 verticies in a graph.

Djikstra's Algorithm

This algorithm determines the shortest path in G from vertex s to **all other** verticies in the graph.

Input	G, s
Output	$dist$

where $dist$ is an array containing the optimal distances from s to every other vertex in G.

Data Structure

Djikstra uses 3 different data structures to carry out its operation.

1. A $dist$ array (min-heap) that tracks the **current best known cost** to get from s to every other vertex in the graph. For each vertex i in $dist$:
    (a) set the distance from s to be equal to 0
    (b) set the distance from all other verticies to ∞
2. An array called $done$ that holds boolean values denoting if a vertex has been fully prosessed. For each vertex i, set $done[i]$ to false.
3. An array called $parent$ that holds the parent for each vertex.

Algorithm

The Djikstra Algorithm is as follows:

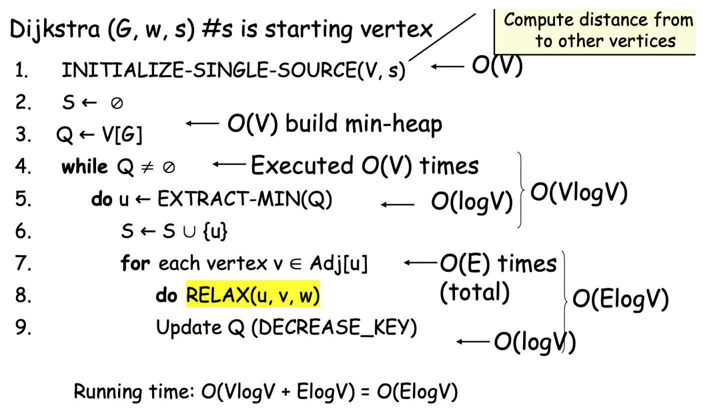
```
while !done {
        u = the closest unprocessed vertex to s

        for v in notDoneVertx{
                if edgeExists.(u,v){
                        if dist[v] > dist[u] + weight(u,v){
                                dist[v] = dist[u] + weight(u,v)

                                parent[v] = u
                        }
                }
        }
}
```

Relaxation Step

Relaxing the edge (u, v) means to test whether we can improve the shortest path to v by going through u

Time Complexity

The Time complexity analysis is as follows:



```
Dijkstra (G, w, s) #s is starting vertex
1.  INITIALIZE-SINGLE-SOURCE(V, s)
2.  S ← ∅
3.  Q ← V[G]
4.  while Q ≠ ∅
5.      do u ← EXTRACT-MIN(Q)
6.          S ← S ∪ {u}
7.          for each vertex v ∈ Adj[u]
8.              do RELAX(u, v, w)
9.              Update Q (DECREASE_KEY)

    Running time: O(VlogV + ElogV) = O(ElogV)
```

Compute distance from to other vertices
O(V)
O(V) build min-heap
Executed O(V) times
O(logV)
O(VlogV)
O(E) times (total)
O(ElogV)
O(logV)

Explaination

Let the node at which we are starting be called the initial node. Let the distance of node Y be the distance from the initial node to Y. Dijkstra's algorithm will initially start with infinite distances and will try to improve them step by step.

1. Mark all nodes unvisited. Create a set of all the unvisited nodes called the unvisited set.

2. Assign to every node a tentative distance value: set it to zero for our initial node and to infinity for all other nodes. The tentative distance of a node v is the length of the shortest path discovered so far between the node v and the starting node. Since initially no path is known to any other vertex than the source itself (which is a path of length zero), all other tentative distances are initially set to infinity. Set the initial node as current.

3. For the current node, consider all of its unvisited neighbors and calculate their tentative distances through the current node. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one. For example, if the current node A is marked with a distance of 6, and the edge connecting it with a neighbor B has length 2, then the distance to B through A will be $6 + 2 = 8$. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, the current value will be kept.

4. When we are done considering all of the unvisited neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will never be checked again.

5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.

6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new current node, and go back to step 3.

When planning a route, it is actually not necessary to wait until the destination node is "visited" as above: the algorithm can stop once the destination node has the smallest tentative distance among all "unvisited" nodes (and thus could be selected as the next "current"). [1]

[1] Wikipedia