Dijkstra's Algorithm

Definition

Dijkstra's Algorithm is a greedy algorithm that finds the shortest path from a particular source node to every other node in a graph. The algorithm is essentially a modified breadth first search that uses a **priority queue** instead of a FIFO queue.

- Works on the basis that **each subpath is the shortest path**.
- Produces a **shortest path tree**
    - It differs from a *minimum spanning tree* because the shortest distance between two vertices might not include all the vertices of the graph. (i.e It does not span a graph).
- Dijkstra’s algorithm has the **same structure as Prim’s algorithm**.
    - The main difference is in the interpretation of the key values, the distance values.
    - The algorithm updates the distance values as they're added **one a time**. NOTE: at the beginning the source is initially `0`, while the other nodes distance values are initially `∞`.
        - if a vertex is added to the tree, only the distance values of its **neighbors** (outside the tree) are affected and these are updated. Source
        -
- **WTF is the Relaxtion Process?** Relaxation is when you update the cost of all vertices connected to a vertex, if the costs would be improved by including the path via *v*. Source
    - Whenever we face a ***tense*** node (i.e. when the tentative shortest path is incorrect because there's a path that is shorter), we ***relax*** the edge.
        - For example: `an edge u→v is tense if dist(u) + w(u→v) < dist(v).` Source

Okay but what's the more practical explaination?

Dijkstra's algorithm simply finds the shortest path between **any** two vertices in a graph.

- From a starting vertex we **visit each neighbor** and **find the shortest subpath**.

Pseudocode

```
function dijkstra(source):
        InitSSSP(source) //initializes distances to neighbors to ∞
```

```
        Insert(source, 0)

        while the priority queue is not empty
                u = minimum vertex
                for all edges u→v
                        if u→v is tense
                                Relax(u→v)
                                if v is in the priority queue
                                        DecreaseKey(v, dist(v))
                                else
                                        Insert(v, dist(v))
```

Source

Algorithm Description

Dijkstra's is based on a **priority queue** and tentative distances as key values.

1. We loop through the priority queue, while it isn't empty.
2. Choosing the minimum vertex before traversing through its edges.
3. Relaxing all tense edges we find.
4. If the neighboring vertex is in the priority queue we invoke the ***binary heap operation*** `DecreaseKey` which *decreases the value of the key given in the first argument to the value of the second argument.* Source: (Gopal, 301)
5. Otherwise, you call the ***binary heap operation*** for insertion.

TIme Complexity

The worst case time complexity of Dijkstra's is `O(E log V)`.
Operations breakdown:

- FInding and updating each adjacent vertex's weight in the min heap is `O(log V) + O(1)` which is `O(log V)`
- Then updating all adjacent edges is `E * log V`
- Thus, the time complexity for the algorithm is `O(E log V)`