Dijkstra's Shortest Path Algorithm

The shortest path between u and v is denoted $\delta(u,v)$, if there is no path, then $\delta(u,v) = \infty$

1 Can a shortest path contain a cycle?

A directed cycle is:

- Positive: if its edge weights sum up to a positive number
- Negative: if its edge weights sum up to a negative number

If there is a positive cycle in the graph, it will not be contained in any shortest path between u and v so we can assume that the shortest paths we find contain no positive cycles.

However if there is a negative cycle between u and v, then $\delta(u,v) = -\infty$ so we shall assume that the graphs we consider do not contain negative cycles.

2 Single-Source Shortest Paths

- Aim: to describe an algorithm that solves the single-source shortest paths problem, i.e. an algorithm that finds the shortest path from a specific source vertex
- This is a generalization of BFS
- So the output of the algorithm should be two arrays d, π where for each vertex v:
  - $d(v) = \delta(s,v)$
  - $\pi(v)$ is the predecessor of v

3 Relaxation

- Assume that the weight on every edge is non-negative
- We do not directly compute the entry $d(v) = \delta(s,v)$
- Instead, at every step, $d(v)$ is an estimate for $\delta(s,v)$
  - Initially, $d(v) = \infty$, and it always remains $d(v) \geqslant \delta(s,v)$
  - $d(v)$ is updated (i.e. it decreases) as shorter paths are found
  - At the end of the algorithm we have $d(v) = \delta(s,v)$

Listing 1 Initialise-Single-Source(G,s)

```
for each vertex v∈V(g) do
    d(v)=∞
    π(v)=NIL
d(s)=0
```

The process of relaxing an edge (u,v):

- Test whether we can improve the shortest path from s to v that we found so far, by going through u
- If yes, then update $d(v)$ and $\pi(v)$
  - Decrease the estimate $d(v)$
  - Update the predecessor $\pi(v)$ to u
- The algorithm first calls initialise-single-source and then it repeatedly relaxes the appropriate edges (according to the weight function w)

Listing 2 Relax(u,v,w)

```
if d(v)>d(u)+w(u,v) then
    d(v)=d(u)+w(u,v)
    π(v)=u
```

4 Dijkstra's Algorithm

- Initialisation: distance to source A is 0, $S = \emptyset$, Q=V
- S stores the vertices v for which we already found $\delta(A, v)$
- Q stores all the other vertices
- While q is not empty
  - Remove from Q the vertex u for which d(u) is minimum
  - Add this vertex to s
  - Relax all edges leaving u

Listing 3 Dijkstra(G,w,s)

```
Initialise-Single-Source(G,s)
S=∅
Q=V(G)
while Q≠ ∅ do
    u=Extract-Min(Q)
    S=S∪{u}
    for each vertex v ∈ Adj(u) do
        Relax(u,v,w)
```

4.1 Runtime

Initialisation is done in $O(V)$ time - two operations per vertex

Finding the vertex v in Q with minimum d(v) takes $O(V)$ time and this is done v times

- To find the minimum, just scan the set Q
- To compute the new vertex in S, find the new minimum of Q

Relaxation takes in total $O(E)$ time as every edge is relaxed once

The total running time is $O(V + V^2 + E) = O(V^2)$

However using a more sophisticated implementation for extracting the minimum for Q it can run in $O(V \log V + E)$ time

5 Properties of shortest paths and relaxation

Definition: Triangle inequality

For all edges (u,v) we have $\delta(s, v) \leqslant \delta(s, u) + w(u, v)$

Definition: Optimal Substructure

Any subpath of a shortest path is also a shortest path

Definition: Upper bound property

For every vertex v, we have $d(v) \geqslant \delta(s, v)$

Definition: No-path property

If $\delta(s, v) = \infty$ then we have $d(v) = \infty$ at every iteration

Definition: Convergence property

If there is a shortest path from s to v including the edge (u,v) and if $d(u) = \delta(s, u)$, then we obtain $d(v) = \delta(s, v)$ when (u, v) is relaxed

6 Correctness of Dijkstra's algorithm

We need to prove the loop invariant always remains true.

At the start of each iteration of the while loop, $d(v) = \delta(s, v)$ for every $v \in S$

- Initialisation: at the start of the algorithm S is empty, so the loop invariant is trivially true
- Maintenance: we need to show that $d(u) = \delta(s, u)$ when u is added to S
- Termination: at the end, S contains every vertex, which implies that $d(v) = \delta(s, v)$ for all vertices v in the graph