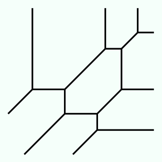
Tropical Semirings

A general method for declaratively solving graph problems

Simon Zeng

UWCSC

November 24, 2022



Handout

- You may obtain a copy of these notes and code samples on my website
- Code samples are in Haskell and were tested on GHC 9.2.4

Links

- https://simonzeng.com/tropical.pdf
- https://simonzeng.com/tropical.hs

Dijkstra's Shortest Path Algorithm

- Q: How do we traditionally get the shortest path between two nodes on a graph?

Dijkstra's Shortest Path Algorithm

- Q: How do we traditionally get the shortest path between two nodes on a graph?
- A: *Dijkstra's algorithm*

Pseudocode

```
function Dijkstra(Graph, source):
    dist[source] <- 0
    create vertex priority queue Q
    for each vertex v in Graph.Vertices:
        if v != source
            (dist[v], prev[v]) <- INFINITY, UNDEFINED
        Q.add_with_priority(v, dist[v])
    while Q is not empty:
        u <- Q.extract_min()
        for each neighbor v of u:
            alt <- dist[u] + Graph.Edges(u, v)
            if alt < dist[v]:
                (dist[v], prev[v]) <- alt, u
                Q.decrease_priority(v, alt)
    return dist, prev
```

Dijkstra's Shortest Path Algorithm

- Q: How do we traditionally get the shortest path between two nodes on a graph?
- A: *Dijkstra's algorithm*

Pseudocode

```
function Dijkstra(Graph, source):
    dist[source] <- 0
    create vertex priority queue Q
    for each vertex v in Graph.Vertices:
        if v != source
            (dist[v], prev[v]) <- INFINITY, UNDEFINED
        Q.add_with_priority(v, dist[v])
    while Q is not empty:
        u <- Q.extract_min()
        for each neighbor v of u:
            alt <- dist[u] + Graph.Edges(u, v)
            if alt < dist[v]:
                (dist[v], prev[v]) <- alt, u
                Q.decrease_priority(v, alt)
    return dist, prev
```

- Very classic, but:
    - Uses lots of state and mutation
    - Hard to tell what's going on from just reading the code

Core of Dijkstra's Algorithm

- When going from some node u to some node v:

Core of Dijkstra's Algorithm

- When going from some node u to some node v:
    - Get the best distance from source to u...

Core of Dijkstra's Algorithm

- When going from some node u to some node v:
  - Get the best distance from source to u...
  - ...**add** the weight of the edge uv...

Core of Dijkstra's Algorithm

- When going from some node u to some node v:
  - Get the best distance from source to u...
  - ...**add** the weight of the edge uv...
  - ...compare against existing best distance of v...

Core of Dijkstra's Algorithm

- When going from some node u to some node v:
  - Get the best distance from source to u...
  - ...**add** the weight of the edge uv...
  - ...compare against existing best distance of v...
  - ... store the **minimum** between our number and what v already has

Core of Dijkstra's Algorithm

- When going from some node u to some node v:
  - Get the best distance from source to u...
  - ...**add** the weight of the edge uv...
  - ...compare against existing best distance of v...
  - ... store the **minimum** between our number and what v already has

Pseudocode

```
alt <- dist[u] + Graph.Edges(u, v)
if alt < dist[v]:
    dist[v] <- alt
```

Core of Dijkstra's Algorithm

- When going from some node u to some node v:
  - Get the best distance from source to u...
  - ...**add** the weight of the edge uv...
  - ...compare against existing best distance of v...
  - ... store the **minimum** between our number and what v already has

Pseudocode

```
alt <- dist[u] + Graph.Edges(u, v)
if alt < dist[v]:
  dist[v] <- alt
```

- The rest of Dijkstra's tells us only **when** we look at a particular node

Core of Dijkstra's Algorithm

- When going from some node u to some node v:
  - Get the best distance from source to u...
  - ...**add** the weight of the edge uv...
  - ...compare against existing best distance of v...
  - ... store the **minimum** between our number and what v already has

Pseudocode

```
alt <- dist[u] + Graph.Edges(u, v)
if alt < dist[v]:
    dist[v] <- alt
```

- The rest of Dijkstra's tells us only **when** we look at a particular node
- Dijkstra's is just node ordering boilerplate around this core operation

A Functional Kernel

- The original calculates, then compares, then (sometimes) sets

Original Pseudocode

```
alt <- dist[u] + Graph.Edges(u, v)
if alt < dist[v]:
    dist[v] <- alt
```

- The comparison+set can be written as a single function call

Refined pseudocode

```
dist[v] <- min(dist[v], dist[u] + Graph.Edges(u, v))
```

- This is the core of path algorithms!
- Remember this for later