Tropical Semirings

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

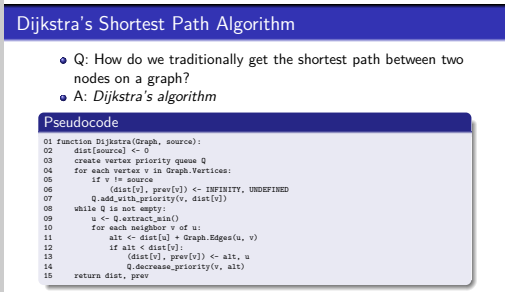
Links

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

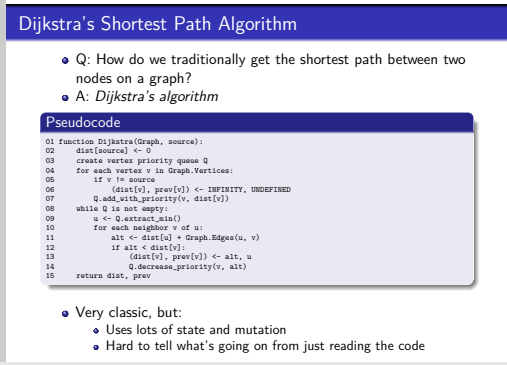
- Greeting
- Functional programming talk
  - Not lambda
  - Not purity
  - Not monads
- Solving real problems declaratively
  - Graph problems!
- We develop an algebraic method that elegantly describes a class of graph problems and their solutions
- These notes are available on my website

Dijkstra's Shortest Path Algorithm
- Q: How do we traditionally get the shortest path between two nodes on a graph?

- Q: read
- (next, then read)
- Everyone love's Dijkstra's
- Set up priority queue, put things into it, take things out of it, stop iterating based on it
- (next, then read)
- It's a very imperative algorithm
- CS 341 shows a few other graph path algorithms
  - from that you'd think they're all inherently imperative
- But: graph problems are **not** inherently imperative!
- Let's zoom in to the meat



- Q: read
- (next, then read)
- Everyone love's Dijkstra's
- Set up priority queue, put things into it, take things out of it, stop iterating based on it
- (next, then read)
- It's a very imperative algorithm
- CS 341 shows a few other graph path algorithms
  - from that you'd think they're all inherently imperative
- But: graph problems are **not** inherently imperative!
- Let's zoom in to the meat



- Q: read
- (next, then read)
- Everyone love's Dijkstra's
- Set up priority queue, put things into it, take things out of it, stop iterating based on it
- (next, then read)
- It's a very imperative algorithm
- CS 341 shows a few other graph path algorithms
  - from that you'd think they're all inherently imperative
- But: graph problems are **not** inherently imperative!
- Let's zoom in to the meat

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

```
11 alt <- dist[u] + Graph.Edges(u, v)
12 if alt < dist[v]:
13     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

```
11 alt <- dist[u] + Graph.Edges(u, v)
12 if alt < dist[v]:
13     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

```
11 alt <- dist[u] + Graph.Edges(u, v)
12 if alt < dist[v]:
13     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

```
11 alt <- dist[u] + Graph.Edges(u, v)
12 if alt < dist[v]:
13     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