2022-11-25

Tropical Semirings

Tropical Semirings

A general method for declaratively solving graph problems

Simon Zeng

UWCSC

November 25, 2022

- 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

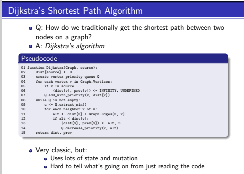
Dijkstra's Shortest Path Algorithm

- 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
```

- 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...

Tropical Semirings

The Essence of the Path Algorithm

2022-11-25

Core of Dijkstra's Algorithm

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

Original Pseudocode

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

A Functional Kernel

Original Pseudocode

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

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

A Functional Kernel

Original Pseudocode

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

- The original calculates, then compares, then (sometimes) sets
- The comparison+set can be written as a single function call

A Functional Kernel

Original Pseudocode

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

- The original calculates, then compares, then (sometimes) sets
- The comparison+set can be written as a single function call

Refined pseudocode

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

A Functional Kernel

Original Pseudocode

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

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

The Algebra of the Path Algorithm

Essence of the path algorithm

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

└─The Algebra of the Path Algorithm

The Algebra of the Path Algorithm

Essence of the path algorithm

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

- There are only two operations here: `min` and $+$

The Algebra of the Path Algorithm

Essence of the path algorithm

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

- There are only two operations here: `min` and $+$
- Q: What algebraic structures have two operations?

└─The Algebra of the Path Algorithm

The Algebra of the Path Algorithm

Essence of the path algorithm

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

- There are only two operations here: `min` and $+$
- Q: What algebraic structures have two operations?
- A: Rings, fields, and related structures

└─The Algebra of the Path Algorithm

The Algebra of the Path Algorithm

Essence of the path algorithm

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

- There are only two operations here: `min` and $+$
- Q: What algebraic structures have two operations?
- A: Rings, fields, and related structures
- Let's define a ring-like structure $(R, +, \cdot)$ such that:

└─The Algebra of the Path Algorithm

The Algebra of the Path Algorithm

Essence of the path algorithm

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

- There are only two operations here: `min` and $+$
- Q: What algebraic structures have two operations?
- A: Rings, fields, and related structures
- Let's define a ring-like structure $(R, +, \cdot)$ such that:
  - The underlying set R is $\mathbb{R}$

└─The Algebra of the Path Algorithm

The Algebra of the Path Algorithm

Essence of the path algorithm

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

- There are only two operations here: `min` and $+$
- Q: What algebraic structures have two operations?
- A: Rings, fields, and related structures
- Let's define a ring-like structure $(R, +, \cdot)$ such that:
  - The underlying set R is $\mathbb{R}$
  - The "addition" operation $+$ is the function `min` that takes the minimum of it's two arguments

└─The Algebra of the Path Algorithm

The Algebra of the Path Algorithm

Essence of the path algorithm

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

- There are only two operations here: `min` and $+$
- Q: What algebraic structures have two operations?
- A: Rings, fields, and related structures
- Let's define a ring-like structure $(R, +, \cdot)$ such that:
  - The underlying set R is $\mathbb{R}$
  - The "addition" operation $+$ is the function `min` that takes the minimum of it's two arguments
  - The "multiplication" operation $\cdot$ is the usual addition on reals

└The Algebra of the Path Algorithm

The Algebra of the Path Algorithm

Essence of the path algorithm

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

- There are only two operations here: `min` and $+$
- Q: What algebraic structures have two operations?
- A: Rings, fields, and related structures
- Let's define a ring-like structure $(R, +, \cdot)$ such that:
  - The underlying set R is $\mathbb{R}$
  - The "addition" operation $+$ is the function `min` that takes the minimum of it's two arguments
  - The "multiplication" operation $\cdot$ is the usual addition on reals

Essence of the path algorithm, algebraically

$$dist[v] = dist[v] + dist[u] \cdot Graph.Edges(u, v)$$