

2022-11-24

# Tropical Semirings

Tropical Semirings

A general method for declaratively solving graph problems

Simon Zeng

UWISC

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

# Tropical Semirings

## └ The essence of the path algorithm

### └ Dijkstra's Shortest Path Algorithm

- 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

## Tropical Semirings

└ The essence of the path algorithm

└ Dijkstra's Shortest Path Algorithm

## 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

```

1: function Dijkstra(Graph, source):
2:   dist[source] ← 0
3:   create vertex priority queue Q
4:   for each vertex v in Graph.Vertices:
5:     if v != source
6:       dist[v] ← ∞
7:       dist[v], v ← DEQUEUE, ENQUEUE
8:   while Q is not empty:
9:     u ← Q.dequeue()
10:    for each neighbor v of u:
11:      alt ← dist[u] + length(u, v)
12:      if alt < dist[v]:
13:        dist[v] ← alt
14:        Dijkstra_priorityQ.enqueue(v, alt)
15:  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

## Tropical Semirings

└ The essence of the path algorithm

└ Dijkstra's Shortest Path Algorithm

- 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

```

1: function Dijkstra(Graph, source):
2:   dist[source] ← 0
3:   create vertex priority queue Q
4:   for each vertex v in Graph.Vertices:
5:     if v != source
6:       dist[v] ← ∞
7:       insert v into Q
8:   while Q is not empty:
9:     u ← Q.extract_min()
10:    for each neighbor v of u:
11:      alt ← dist[u] + length(u, v)
12:      if alt < dist[v]:
13:        dist[v] ← alt
14:        decrease_key(Q, v, alt)
15:   return dist, prev

```

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

2022-11-24

# Tropical Semirings

- └ The essence of the path algorithm
  - └ Core of Dijkstra's Algorithm

Core of Dijkstra's Algorithm

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

2022-11-24

# Tropical Semirings

- └ The essence of the path algorithm
  - └ 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  $u$ ...

# Tropical Semirings

- └ The essence of the path 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  $uv$ ...



# Tropical Semirings

└ The essence of the path algorithm

└ 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  $uv$ ...
  - ... compare against existing best distance of  $v$ ...

# Tropical Semirings

└ The essence of the path algorithm

└ 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  $uv$ ...
  - ... compare against existing best distance of  $v$ ...
  - ... store the **minimum** between our number and what  $v$  already has

# Tropical Semirings

- └ The essence of the path algorithm
  - └ 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  $uv$ ...
  - ... compare against existing best distance of  $v$ ...
  - ... store the **minimum** between our number and what  $v$  already has

### Pseudocode

```
14 add  $v \leftarrow \text{dist}[u] + \text{weight}(u, v)$   
15 if  $\text{dist}[v] > \text{dist}[u] + \text{weight}(u, v)$   
16  $\text{dist}[v] \leftarrow \text{dist}[u] + \text{weight}(u, v)$ 
```

## Tropical Semirings

- └ The essence of the path algorithm
  - └ 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  $uv$ ...
  - ... compare against existing best distance of  $v$ ...
  - ... store the **minimum** between our number and what  $v$  already has

## Pseudocode

```

14 dist ← dist[u] + Graph.Edge(u, v)
15 if dist < dist[v]
16   dist[v] ← dist

```

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

## Tropical Semirings

└ The essence of the path algorithm

└ 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  $uv$ ...
  - ... compare against existing best distance of  $v$ ...
  - ... store the **minimum** between our number and what  $v$  already has

## Pseudocode

```

14 dist ← dist[u] + Graph.Edge(u, v)
15 if dist < dist[v]
16   dist[v] ← dist

```

- 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

## Tropical Semirings

└ The essence of the path algorithm

└ A Functional Kernel

## A Functional Kernel

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

## Original Pseudocode

```
11 dist ← dist[s] + Graph.Edges(s, v)
12 if dist < dist[v]
13   dist[v] ← dist
```

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

## Refined pseudocode

```
dist[v] ← min(dist[s] + Graph.Edges(s, v))
```

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