CSCI3055U – PROGRAMMING LANGUAGES

Final Project

WHAT WAS THE PROBLEM I CHOSE?

I chose to implement Dijsktra's algorithm in the language I researched, Rust.

WHAT IS DIJKSTRA'S ALGORITHM?

Dijkstra's algorithm is a shortest-path finding algorithm for weighted graphs. When you input a graph and a source node into the algorithm, the algorithm will return a set of paths from the source to every other node in the graph it is connected to. These paths will have the least possible cost for traversal.

BASIC STEPS OF THE ALGORITHM

1. Create a set of nodes, and set every non-source node to have a distance of infinity and no parent.
2. Set the source node distance to 0.
3. Iterate through the set of nodes, selecting the smallest distance node each time. For the first iteration this will be the source node.
4. Remove the selected node from the set, and iterate over each neighbouring node.
5. For each neighbour, compare its distance with the selected node's distance plus the edge weight between them.
6. If the neighbour's distance is greater than the selected node's + the weight, set the neighbours distance as the selected node distance + the weight, and set the parent as the selected node.
7. Once the set of nodes is empty, return the distance and parent of each node.

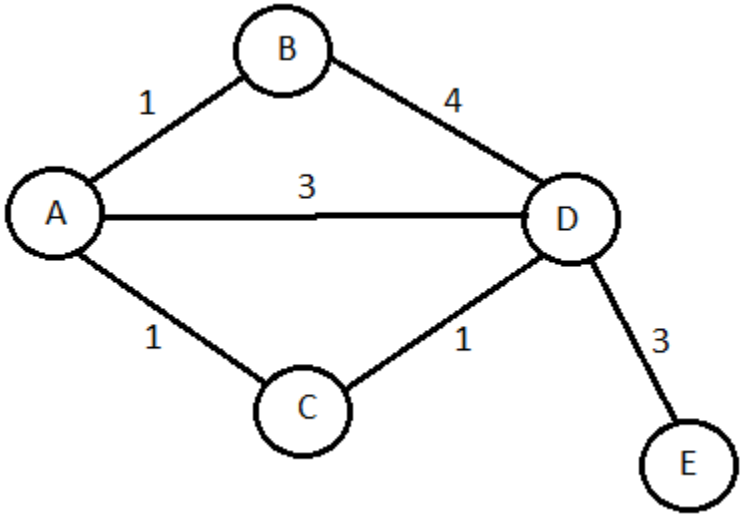
PSEUDOCODE

Pseudocode from Wikipedia

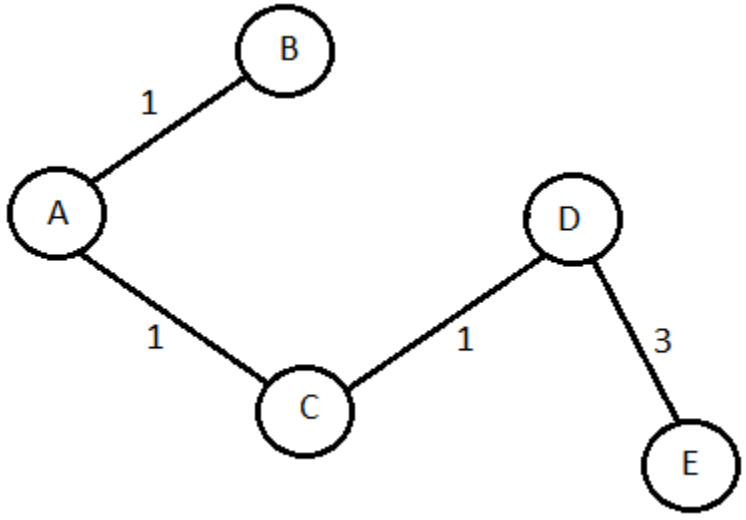
```
1  function Dijkstra(Graph, source):
2
3      create vertex set Q
4
5      for each vertex v in Graph:             // Initialization
6          dist[v] ← INFINITY                  // Unknown distance from source to v
7          prev[v] ← UNDEFINED                 // Previous node in optimal path from source
8          add v to Q                          // All nodes initially in Q (unvisited nodes)
9
10     dist[source] ← 0                        // Distance from source to source
11
12     while Q is not empty:
13         u ← vertex in Q with min dist[u]    // Source node will be selected first
14         remove u from Q
15
16         for each neighbor v of u:           // where v is still in Q.
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]:               // A shorter path to v has been found
19                 dist[v] ← alt
20                 prev[v] ← u
21
22     return dist[], prev[]
```

MY IMPLEMENTATION

I used the following graph for testing my implementation.



My implementation returned the following shortest path graph.



CREATING THE GRAPH

Originally, I had started creating my own graph implementation to use. Populating the graph using my own implementation worked fine, but I started running into problems when trying to implement traits.

I wasn't experienced or knowledgeable enough about Rust in order to implement traits that worked properly in Rust's type system. I kept getting tons of errors when implementing methods like neighbours(), edge_weight() and so on.

Eventually I decided to use a third party graph implementation. I looked for something similar to networkx from python, and found petgraph. (https://github.com/bluss/petgraph)

CREATING THE GRAPH

Then I started trying to do my implementation by using a text file to read in the node and edge information. I could read in teh file information easily enough, but I was getting errors trying to input the information into the Graph struct from petgraph.

Reading in information from files in Rust was giving me String values, but String did not implement the copy trait which was required by the Graph implementation. So I tried converting the Strings to reference strings, &str. This worked but I got more errors due to the lifetime of the &strs.

The references to the strs died whenever I tried to return the Graph or pass it to another function. So eventually I decided to just manually code the graph, since I was using a small test graph.

CREATING THE GRAPH

I ended up with the following function.

```
fn def_lines<'a>() -> Vec<&'a str>
{
        let mut lines: Vec<&'a str> = Vec::new();
        lines.push("A|B|1");
        lines.push("A|D|3");
        lines.push("A|C|1");
        lines.push("C|D|1");
        lines.push("B|D|4");
        lines.push("D|E|3");
        lines
}
```

The <'a> was to manually increase the lifetime of the string references so they could be used in other functions. I tried using <'a> when reading from the file, but that didn't work with the String type due to an error with the size trait of referenced Strings.

So my implementation could now add nodes and edges to the graph. I then started noticing another problem, that the Graph struct in petgraph was making duplicate nodes.

CREATING THE GRAPH

The Graph struct also didn't support calling a specific node by its label, so I switched to the GraphMap implementation in petgraph instead. GraphMap worked much better, as it didn't allow duplicate nodes and nodes could be called by their labels. So I wrote this function to populate the graph.

```
fn generate_graph<'a>(lines: Vec<&'static str>) -> UnGraphMap<&'static str, i32>
{
        let mut G: UnGraphMap<&str,i32> = UnGraphMap::new();
        for line in lines
        {
                let vals: Vec<&str> = line.split('|').collect();
                let weight = match vals[2].parse::<i32>()
                {
                        Ok(weight) => weight,
                        Err(_) => panic!("can't be converted to i32"),
                };
                G.add_edge(vals[0],vals[1],weight);
        }
        G
}
```

So I created a graph G that was an UnGraphMap, that mapped &str to node labels and i32 to edge weights. I then iterated through each edge entry in the Vec argument, and added the edge and node info into the graph. Creating the graph was now finished.

CREATING THE ALGORITHM

Then I started working on implementing the algorithm itself. I first started with declaring a hashmap to map &str to i32 to track distance of each node, a hashmap to map &str to &str to keep track of parents for each node, as well as a HashSet to store each node and a Vec to sort node distances from least to greatest.

```
fn dijkstra(G: UnGraphMap<&str, i32>,source: &str) -> ()
{
        let mut dist: HashMap<&str, i32> = HashMap::new();
        let mut parent: HashMap<&str, &str> = HashMap::new();
        let mut S: HashSet<&str> = HashSet::new();
        let mut sorted: Vec<&str> = Vec::new();
```

And filled them using this code:

```
for v in G.nodes()
{
        dist.insert(v,i32::max_value());
        parent.insert(v,v);
        S.insert(v);
        sorted.push(v);
}
dist.insert(source, 0);
```

CREATING THE ALGORITHM

Then I quickly sorted them from least to greatest distance:

```
let mut swapped = true;
let n = sorted.len();
while swapped == true
{
        swapped = false;
        for i in 1..n
        {
                if dist[sorted[i-1]] > dist[sorted[i]]
                {
                        let s = sorted[i];
                        sorted[i] = sorted[i-1];
                        sorted[i-1] = s;
                        swapped = true;
                }
        }
}
```

Then I implemented the main part of the algorithm

```
while S.is_empty() == false
{
        let u = sorted.remove(0);
        S.remove(u);
        for v in G.neighbors(u)
        {
                let edge = G.edge_weight(u,v);
                let e = edge.unwrap();
                let du = dist[u];
                if dist[v] > du + e
                {
                        dist.insert(v,(du + e));
                        parent.insert(v,u);
                        S.insert(u);
                        sorted.push(u);
                }
        }
}
```

CREATING THE ALGORITHM

The only problem I really had here was figuring out how to handle the Option<i32> return value from the edge_weight() method, and a bug that was causing the last calculated path to have the wrong distance.

Then I wrote some code to output the shortest paths to the terminal.

```
        for n in G.nodes()
        {
                if n != source
                {
                        println!("From {} to {}",source,n);
                        println!("{}",print_path(source,n,&parent, &dist));
                }
        }
fn print_path(source: &str, dest: &str, parent: &HashMap<&str, &str>, dist: &HashMap<&str, i32>) -> Strin
{
        let mut path = String::new();
        let mut p = parent[dest];
        path.push_str(dest);
        path.push_str(">-");
        while p != source
        {
                path.push_str(p);
                path.push_str(">-");
                p = parent[p];
        }
        path.push_str(source);
        path = reverse(path);
        path.push_str(" With a total cost of ");
        path += &dist[dest].to_string();
        path
}
```

CREATING THE ALGORITHM

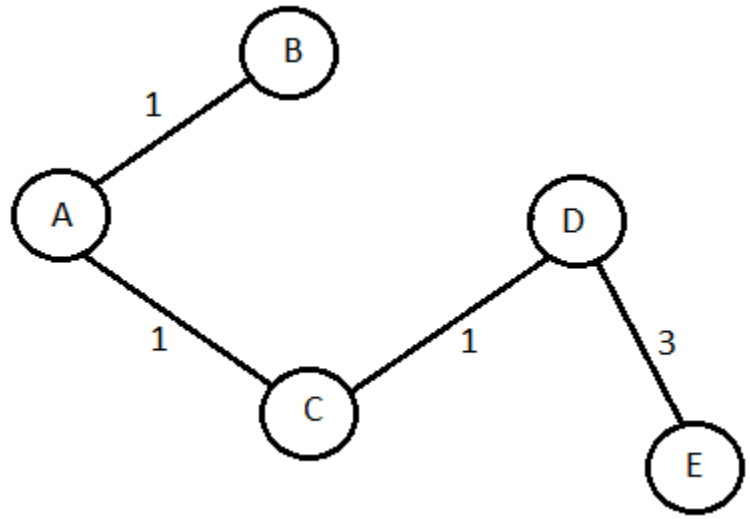
And I had to write my own reverse function.

```
fn reverse(mut s: String) -> String
{
        let n = s.len();
        let mut s2 = String::new();
        for i in 0..n
        {
                let c = s.pop().unwrap();
                s2.push(c);
        }
        s2
}
```

Final output:

```
From A to B
A->B With a total cost of 1
From A to D
A->C->D With a total cost of 2
From A to C
A->C With a total cost of 1
From A to E
A->C->D->E With a total cost of 5
```

Which corresponds with the shortest path tree. So I can conclude my implementation is correct.



SUITABILITY OF RUST FOR MY PROBLEM

I think Rust is definitely a suitable language to implement Dijkstra's algorithm in. The only problems I had while doing my implementation rose from my inexperience and lack of knowledge about Rust. Somebody who is more proficient at Rust would definitely be able to create a more optimized and clean implementation of the algorithm.

Git Logs

commit f178edc263598485c00c98102d7205ac9693ee16
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Fri Dec 9 15:01:44 2016 -0500

Fixed error where last path had wrong cost. Removed unneeded code

commit 6456db8f3fecf66748138fa8243efbe6c47a9eba
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Fri Dec 9 13:22:08 2016 -0500

Algorithm now calculates shortest paths for every node in the graph

commit 10fdd8a68706ee6868d42109faa4af079a9bcd6a
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Fri Dec 9 12:49:48 2016 -0500

Algorithm now computes shortest distance for each node from source

commit b5684c5601b1f4a13eecd301aab77384961a0ed9
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Fri Dec 9 12:00:22 2016 -0500

Added sort algorithm to sort nodes by distance

commit 7ef41678bdb10223cd9aa707bb7fb13ad2ca9a9a
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Sun Dec 4 12:18:44 2016 -0500

added parent hashmap and pseudocode for relaxedge

commit 4e10178a238b4385bfb274f7bb173e11b78ee122
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Sun Dec 4 12:14:49 2016 -0500

implemented the initializer for algorithm

commit 741df43ae884b7d74270fbc611823f5c63ff1f6c
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Sat Dec 3 17:05:42 2016 -0500

Built loop that creates the String to be returned from dijkstra function

commit 8eb2289ab3200968d077f69dce8f4937c090e804
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Sat Dec 3 16:50:39 2016 -0500

Due to type ownership problems, cut out the file and instead manually inputted the edge data

commit 0d9ce428a8a416d258a38fe49ba4ee7d0cf1f1b2
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Sat Dec 3 16:17:38 2016 -0500

Tried fixing lifetime problem

commit 2dc3f70f620ed2608cbd67e178159cbbd4837a7b
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Sat Dec 3 15:32:43 2016 -0500

Switched to graphmap, trying to get it functional

commit f0ebe3600ec4fd659b75aaf6822375b439577fe7
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Sat Dec 3 13:48:24 2016 -0500

Graph gets constructed but duplicates nodes

commit 633a4e1c4b99dbc6522c1fb8af4e81922bf39ae0
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Sat Dec 3 13:29:01 2016 -0500

Fixed adding edge

Git Logs

commit 5aabd6ab65c092f9e515cb4a69077f28cb592f58
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Sat Dec 3 13:27:52 2016 -0500

Textfile data parsing correctly, can add nodes to graph. Error when adding edge to graph.

commit 4633f744723fe882f5f292114b9f2a23c066779d
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Sat Dec 3 12:58:38 2016 -0500

Fixed try! macro error when opening file

commit fe3c457627db5482d276de4469b36138ed952a54
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Fri Dec 2 14:18:02 2016 -0500

Built dependencies

commit 09ab70051f34356620d79be5216a78cc3a8857d7
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Fri Dec 2 14:11:47 2016 -0500

Found better graph library: petgraph by bluss, https://github.com/bluss/petgraph

commit 8faefbe3bf588a669b37cd314df63b3876ad1ca3
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Fri Dec 2 14:01:05 2016 -0500

Adding dependency for rust-graph crate created by gsingh93. Link to his repository: https://github.com/gsingh93/rust-graph

commit 2b7c86390b4ee3bec93fdbc391a733e8a18ae5ff
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Fri Dec 2 13:57:34 2016 -0500

Removing old directory

commit 89b0b01b9dedbf536d473116b1b2c723ca7c7e0e
Merge: fc62916 b5a4d00
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Fri Dec 2 13:51:10 2016 -0500

Merge branch 'master' of https://github.com/joshualemmon/Prog-Lang-Final-Project
Changing to cargo project

commit fc62916481af4bb695575babdd2ff280bb42e5cf
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Fri Dec 2 13:47:57 2016 -0500

Changing to cargo project

commit b5a4d00508cd232a51f0e005a5bedda33b85c2af
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Fri Dec 2 13:47:20 2016 -0500

Changing to cargo project

commit 9085b39c986949810cf790548be49d732903be23
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Fri Dec 2 13:45:26 2016 -0500

Remade project using cargo

commit 4a3ac6281bb058c7fa6c3d5cd0234570adbdef3d
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Thu Dec 1 14:28:42 2016 -0500

made getting filename from args more optimised

commit 57519de862d8ba1409863e506e9cefb736655675
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Thu Dec 1 12:41:50 2016 -0500

program can now read in a filename as an argument

Git Logs

commit 8e789c1a5efb36ca2d7659d35211d0b7bbcbf6ce
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Thu Dec 1 12:00:27 2016 -0500

Now have working Edge struct and Graph struct that can be iterated over

commit 5a461069da798f75b4c300d6136a81e9c7b0a534
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Thu Dec 1 11:50:07 2016 -0500

Fixed struct implementations, now trying to do some testing to see how well they work

commit 68df8d43e7561cf6e6246b163005e6adb9f6a06d
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Thu Dec 1 11:28:48 2016 -0500

Added a struct tiple to describe edges, created a struct to describe a graph

commit c3a71ce841bd2ee3d413b87951d269b260535db1
Author: joshualemmon <joshua.lemmon@uoit.net>
Date: Tue Nov 29 12:07:31 2016 -0500

initial commit

commit bdb81885ab78a0085f908dba2cc49f059d3f517b
Author: Joshua Lemmon <joshualemmon@users.noreply.github.com>
Date: Sun Nov 27 21:52:19 2016 -0500

Initial commit