Algoritmo de Dijkstra

Victor Gerardo Rodríguez Barragán

04 de Septiembre del 2023

Problematica

El ejercicio consiste en encontrar el camino mas corto para llegar de un nodo A hacia un nodo B e imprimirlo en pantalla como una lista.

Solucion

Para resolver el problema el cual consiste en encontrar el camino mas corto desde un nodo inicial hacia un nodo final se utilizo una cola de prioridad para ir almacenando los nodos que se van visitando y una tabla de hash para almacenar los nodos que ya fueron visitados, tambien se utilizo una tabla de hash para almacenar el camino mas corto desde el nodo inicial hasta el nodo final.

Codigo

```rust
use std::collections::BinaryHeap;
use std::collections::HashMap;
use std::collections::HashSet;

#[derive(Debug, PartialEq, Eq, PartialOrd, Ord, Clone, Copy)]
struct Vertice<'a> {
    id: &'a str,
    distance: i32,
}

impl<'a> Vertice<'a>{
    // Metodo para crear un nuevo vertice
    fn new_graph(id: &'a str, distance: i32) -> Self {
        Vertice { id, distance }
    }
}

pub fn dijkstra<'a>(
    nodo: &HashMap<&'a str, Vec<(&'a str, i32)>>,
    inicio: &'a str,
    fin: &'a str,
    ) -> Vec<&'a str>
{
    let mut distancias: HashMap<&str, i32>
        = nodo.keys().map(|&x| (x, i32::max_value())).collect();
    let mut visitas: HashSet<&str> = HashSet::new();
    let mut prioridades: BinaryHeap<Vertice> = BinaryHeap::new();
    let mut shortest_path: HashMap<&str, &str> = HashMap::new();

    let mut path = vec![fin];
```

```
    let mut current = fin;

    distancias.insert(inicio, 0);
    prioridades.push(Vertice::new_graph(inicio, 0));

    // Encontrar el camino mas corto desde el inicio hasta el fin
    while !prioridades.is_empty() {
        let Vertice { id, distance } = prioridades.pop().unwrap();
        if id == fin {
            break;
        }
        if visitas.contains(id) {
            continue;
        }
        visitas.insert(id);
        for (vecino, peso) in &nodo[id] {
            let peso = distance + peso;
            if peso < distancias[vecino] {
                distancias.insert(vecino, peso);
                shortest_path.insert(vecino, id);
                prioridades.push(Vertice::new_graph(vecino, peso));
            }
        }
    }

    // Acomodar el camino mas corto en un vector
    while current != inicio {
        current = shortest_path[current];
        path.push(current);
    }

    // Regresar el camino mas corto
    path
}
```

Finalmente llamamos a la funcion dijkstra desde el main, y ahi es donde se crea el grafo y se le pasa como parametro a la funcion dijkstra junto con el nodo inicial y el nodo final.

```
use std::collections::HashMap;

mod dijkstra;

fn main() {
    // Ejemplo
    let mut graph: HashMap<&str, Vec<(&str, i32)>> = HashMap::new();
```

```
    graph.insert("A", vec![("B", 3), ("D", 8)]);
    graph.insert("B", vec![("D", 5), ("E", 6)]);
    graph.insert("D", vec![("B", 5), ("E", 3), ("F", 2)]);
    graph.insert("E", vec![("F", 1), ("C", 9)]);
    graph.insert("F", vec![("E", 1), ("C", 3)]);
    graph.insert("C", vec![("E", 9), ("F", 3)]);

    let inicio = "A";
    let fin = "C";
    let res = dijkstra::dijkstra(&graph, inicio, fin);
    println!("El camino mas corto desde {} hasta {} es: {:?}", inicio, fin, res);
}
```

Resultado

```
~/Documents/ceti/ceti6sem/ia/1p/rs-graphs ᚠ|master* took 4s
> cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.00s
     Running `target/debug/grafos`
El camino mas corto desde A hasta C es: ["C", "F", "D", "A"]
```

Conclusion

La teoria del algoritmo de Dijkstra es “en teoria” sencilla, pero en la practica se me hizo mucho mas compleja, tuve que aplicar estructuras de datos que me complicaron un poco el codigo, pero al final logre implementar la practica. Una de las cosas que mas me costo trabajo fue implementar el algoritmo como tal en Rust, apesar de haber usado el lenguaje con anterioridad nunca habia implementado un algoritmo como este, al final use muchas cosas como heap binarios, tablas de hash, etc. que me ayudaron a la implementacion del algoritmo, porque con puros Vectores deduje que seria muy complicado, me gustaria mejorar el codigo a lo mejor a lo mejor usando dict o algun otro tipo que facilite y haga mas legible el codigo.