Nama: Isep Lutpi Nur  
NPM: 2113191079  
Tugas: Komunikasi Data Minggu 14

# Implementasi Alogritma Djikstra

Program untuk menentukan jalur tercepat menggunakan algoritma djikstra, Tugas besar mata kuliah Komunikasi data semester 3 Menggunakan Bahasa pemrograman Javascript.

Kode Sumber: https://github.com/iseplutpinur/algoritma*dijkstra*  
*Lihat Halaman: <a href="https://iseplutpinur.github.io/algoritma*dijkstra/">https://iseplutpinur.github.io/algoritma\_dijkstra/

## Kode

// Nama : Isep Lutpi Nur  
// NPM : 2113191079  
// Matkul : Komunikasi Data  
// Dosen : Nanang Hunaifi, ST, MM  
  
  
// membuat queue untuk keluar masuknya vertex / node  
class PriorityQueue {  
 constructor() {  
 this.values = [];  
 }  
 // menambah queue  
 enqueue(val, priority) {  
 this.values.push({ val, priority });  
 this.sort();  
 }  
 // menghapus atau mengeluarkan queue  
 dequeue() {  
 return this.values.shift();  
 }  
 // mensorting queue yg lebih pendek  
 sort() {  
 this.values.sort((a, b) => a.priority - b.priority);  
 }  
}  
  
// membuat graph untuk short path djikstra  
class WeightedGraph {  
 constructor() {  
 this.adjacencyList = {};  
 }  
 // function membuat vertex tempat destinasi  
 addVertex(vertex) {  
 if (!this.adjacencyList[vertex]) this.adjacencyList[vertex] = [];  
 }  
 // function membuat edge dan panjang jarak penghubung ke vertex lain  
 addEdge(vertex1, vertex2, weight) {  
 this.adjacencyList[vertex1].push({ node: vertex2, weight });  
 this.adjacencyList[vertex2].push({ node: vertex1, weight });  
 }  
 // Short-Path  
 Dijkstra(start, finish) {  
 const nodes = new PriorityQueue();  
 const distances = {};  
 const previous = {};  
 let path = []; // tempat menembalikan nodes terakhir  
 let smallest;  
 // membangun initial state  
 for (let vertex in this.adjacencyList) {  
 if (vertex === start) {  
 distances[vertex] = 0;  
 nodes.enqueue(vertex, 0);  
 } else {  
 distances[vertex] = Infinity;  
 nodes.enqueue(vertex, Infinity);  
 }  
 previous[vertex] = null;  
 }  
  
 // menentukan panjang path yg dikunjungi  
 while (nodes.values.length) {  
 smallest = nodes.dequeue().val;  
 if (smallest === finish) {  
 // selesai sampai tujuan mengembalikan nilai terekhir  
 while (previous[smallest]) {  
 path.push(smallest);  
 smallest = previous[smallest];  
 }  
 break;  
 }  
 if (smallest || distances[smallest] !== Infinity) {  
 for (let neighbor in this.adjacencyList[smallest]) {  
 // mencari tetangga dari node  
 let nextNode = this.adjacencyList[smallest][neighbor];  
 //menjumlah tetangga node  
 let candidate = distances[smallest] + nextNode.weight;  
 let nextNeighbor = nextNode.node;  
 if (candidate < distances[nextNeighbor]) {  
 //update jarak terkecil antara node dan tetangga  
 distances[nextNeighbor] = candidate;  
 //update previous - mendapatkan jarak antar tetanggaa sebelumnya  
 previous[nextNeighbor] = smallest;  
 //enqueue hasil queue dengan hasil yg baru  
 nodes.enqueue(nextNeighbor, candidate);  
 }  
 }  
 }  
 }  
 return path.concat(smallest).reverse();  
 }  
}  
  
// inisialisasi graph  
const graph = new WeightedGraph();  
  
// 1.   
graph.addVertex("A");  
graph.addVertex("B");  
graph.addVertex("C");  
graph.addVertex("D");  
  
// 2.  
graph.addEdge("A", "B", 1);  
graph.addEdge("A", "C", 6);  
graph.addEdge("B", "D", 3);  
graph.addEdge("C", "D", 1);  
  
// 3.  
console.log(graph.Dijkstra("A", "D"));  
// hasil [ 'A', 'B', 'D' ]

## Cara Menggunakan

### Dengan Penulisan langsung / Coding

1. Inisialisasi Variable dengan class WeightedGraph()

const graph = new WeightedGraph();

1. Menambah label vertex dengan menggunakan method addVertex(labelvertex).

graph.addVertex("A");  
graph.addVertex("B");  
graph.addVertex("C");  
graph.addVertex("D");

1. Menambah bobotEdge denan method addEdge(vertex1, vertex2, bobot).

graph.addEdge("A", "B", 1);  
graph.addEdge("A", "C", 6);  
graph.addEdge("B", "D", 3);  
graph.addEdge("C", "D", 1);

1. Memanggil methood Djikstra(node awal, node tujuan).

console.log(graph.Dijkstra("A", "D"));  
// hasil [ 'A', 'B', 'D' ]

### Dengan GUI/ Halaman Web

Contoh Kasus

### Cara kerja

1. Ketika graph di inisialisasi maka constructor akan mendeklarasikan adjacencyList atau daftar node yang terhubung untuk perhitungan, dideklarasikan dengan tipe data objek.
2. Ketika method addVertex dijalankan dengan parameter label vertex atau node akan ditambahkan ke properti adjacencyList dengan tipe data array yang nantinya array tersebut akan memuat detal data bobot edge vertex.
3. Setelah selesai menambahkan label dari vertex, bobot edge vertex akan ditambahkan sesuai dengan labelnya masing masing kedalam properti adjacencyList[vertex] masing masng.
4. setelah label dan bobot tiap tiap node/ vertex didapat maka pencarian jalur tecepat dapat dilakukan dengan cara memanggil method Djikstra(awal, tujuan), dengan dua parameter yaitu parameter awal dan parameter tujuan, method ini mengembalikan/return hasil dari pencarian jalur tercepat dengan tipe data array contoh [ 'A', 'B', 'D' ].
5. Proses diidalam method Djikstra.  
   a. Membangun initial state atau node keberangkatan Seperti contoh node keberangkatan d sini adalah 'A' maka priority nya 0, dan yang lain infinity dan node keberangkatan di sortir menjadi array paling rendah atau 0, untuk nantinya di dequeu atau diambil sementara.  
   b. menentukan path yang dikunjungi dengan perulangan, perulangan nya akan berhenti jika nilai label dengan nilai terkecil sama dengan label tujuan. \*di point d dibawah  
   c. variable samllest akan di isi nilai keberangkatan yang telah di sort dari node. contoh disini node awalnya 'A' maka variable samllest akan berisi nilai 'A'.   
   D. Kemudian smallest di cek apakah sama dengan nilai tujuan, jika sama maka dilakukan perulangan dengan parameter previous. didalam perulangan dilakukan push ke variable path dengan nilai smallest