

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 Aplikasi: 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 });
  }
}
```

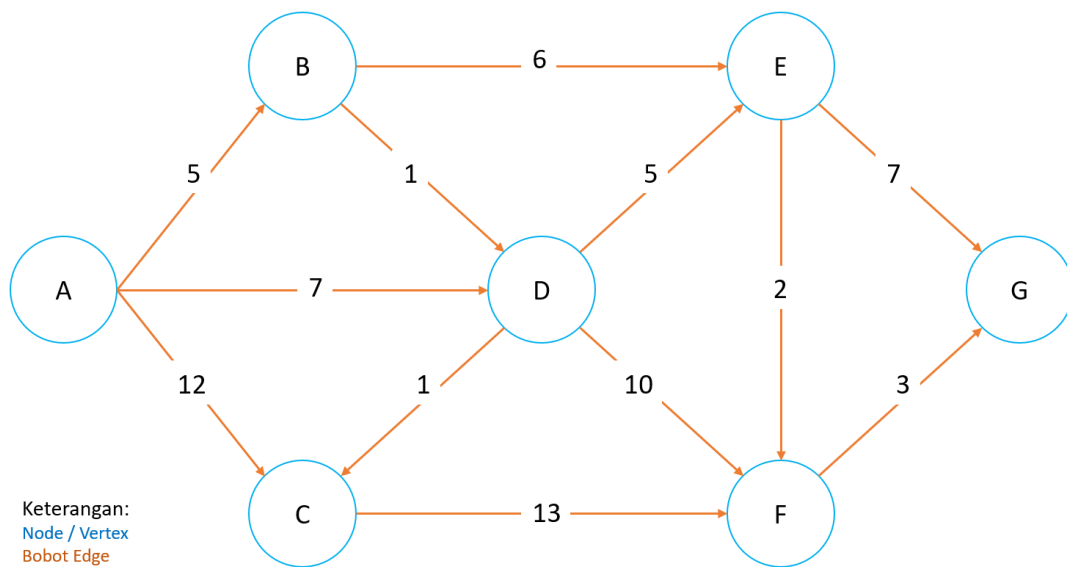
```

    }
    // 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 terakhir
                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 tetangga
                        sebelumnya
                        previous[nextNeighbor] = smallest;
                        //enqueue hasil queue dengan hasil yg baru
                        nodes.enqueue(nextNeighbor, candidate);
                    }
                }
            }
        }
        return path.concat(smallest).reverse();
    }
}

```

Contoh Kasus



Penyelesaian Dengan Penulisan langsung / Coding

1. Inisialisasi Variable dengan class WeightedGraph()

```
const graph = new weightedGraph();
```

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

```
graph.addVertex("A");  
graph.addVertex("B");  
graph.addVertex("C");  
graph.addVertex("D");  
graph.addVertex("E");  
graph.addVertex("F");  
graph.addVertex("G");
```

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

```
graph.addEdge("A", "B", 5);  
graph.addEdge("A", "C", 12);  
graph.addEdge("A", "D", 7);  
  
graph.addEdge("B", "D", 1);  
graph.addEdge("B", "E", 6);  
  
graph.addEdge("C", "F", 13);  
  
graph.addEdge("D", "C", 1);  
graph.addEdge("D", "E", 5);  
graph.addEdge("D", "F", 10);
```

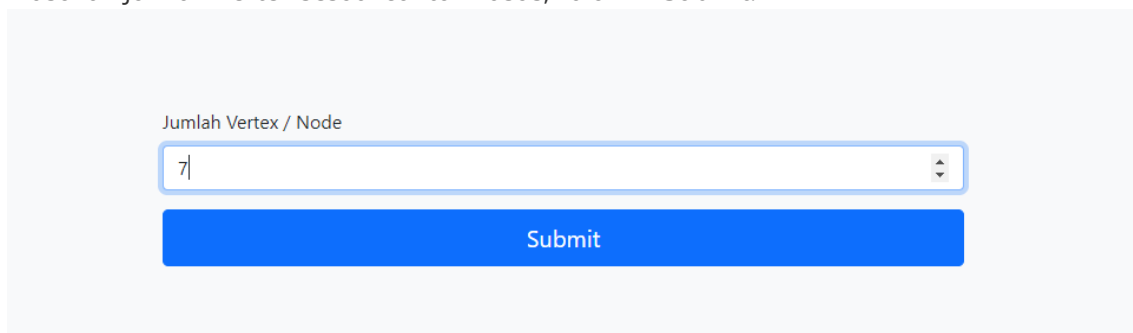
```
graph.addEdge("E", "F", 2);  
graph.addEdge("E", "G", 7);  
  
graph.addEdge("F", "G", 3);
```

4. Memanggil method Dijkstra(node awal, node tujuan).

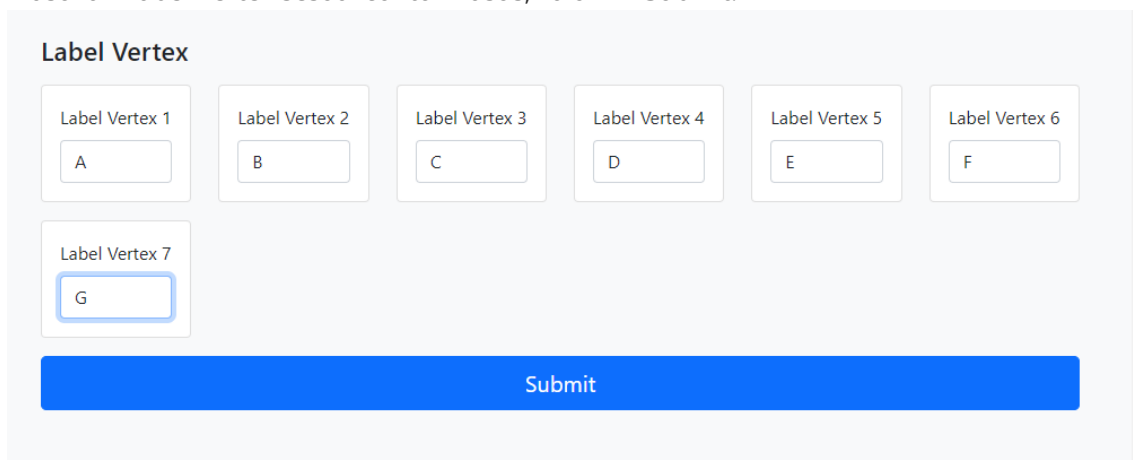
```
console.log(graph.Dijkstra("A", "D"));  
// hasil [ 'A', 'B', 'E', 'F', 'G' ]
```

Dengan GUI/ Halaman Web

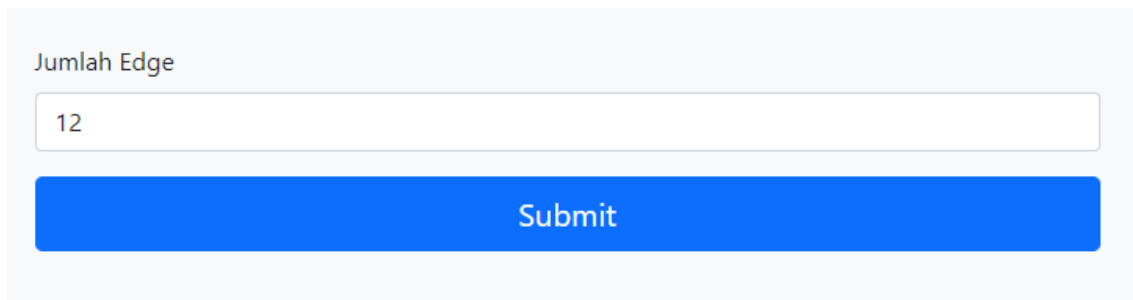
1. Kunjungi halaman: https://iseplutpinur.github.io/algorithm_dijkstra/
2. Masukan Jumlah Vertex sesuai contoh kasus, Lalu Klik Submit.



3. Masukan Label Vertex sesuai contoh kasus, Lalu Klik Submit.



4. Masukan Jumlah edge atau sambungan antara node/vertex sesuai contoh kasus, Lalu Klik Submit.



5. Masukan Bobot tiap tiap ede sesuai dengan contoh kasus, lalu klik Submit. *Pada pilihan vertex 2 dan bobot edge akan terbuka jika Vertex 1 Telah di pilih.

Bobot edge

Edge 1	Edge 2	Edge 3	Edge 4	Edge 5	Edge 6	Edge 7	Edge 8	Edge 9	Edge 10	Edge 11	Edge 12
Vertex 1 A	Vertex 1 A	Vertex 1 A	Vertex 1 B	Vertex 1 B	Vertex 1 C	Vertex 1 D	Vertex 1 D	Vertex 1 D	Vertex 1 E	Vertex 1 E	Vertex 1 F
Vertex 2 B	Vertex 2 C	Vertex 2 D	Vertex 2 D	Vertex 2 E	Vertex 2 F	Vertex 2 C	Vertex 2 E	Vertex 2 F	Vertex 2 F	Vertex 2 G	Vertex 2 G
Bobot Edge 5	Bobot Edge 12	Bobot Edge 7	Bobot Edge 1	Bobot Edge 6	Bobot Edge 13	Bobot Edge 1	Bobot Edge 5	Bobot Edge 10	Bobot Edge 2	Bobot Edge 7	Bobot Edge 3

Submit

6. Pilih Vertex/ Node awal dan Vertex/ Node tujuan, Lalu klik Hitung, Maka kolom jalur tercepat akan menampilkan hasil.

Hitung Jalur Tercepat

Vertex Awal	Vertex Tujuan	Jalur Tercepat
A	G	A -> B -> E -> F -> G
Hitung		

7. Untuk Menggunakan kembali, Klik Tombol reset.

Reset

Cara kerja

1. Ketika graph di inialisasi maka constructor akan mendeklarasikan adjacencyList atau daftar node yang terhubung untuk perhitungan, dideklarasikan dengan tipe data objek.

```
class weightedGraph {  
  constructor() {  
    this.adjacencyList = {};  
  }  
}  
  
.....  
  
const graph = new weightedGraph();
```

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 detail data bobot edge vertex.

```
// function membuat vertex tempat destinasi  
addVertex(vertex) {  
  if (!this.adjacencyList[vertex]) this.adjacencyList[vertex] = [];  
}  
  
graph.addVertex("A");  
graph.addVertex("B");  
graph.addVertex("C");  
graph.addVertex("D");  
graph.addVertex("E");  
graph.addVertex("F");  
graph.addVertex("G");
```

3. Setelah selesai menambahkan label dari vertex, bobot edge vertex akan ditambahkan sesuai dengan labelnya masing masing kedalam properti adjacencyList[vertex] masing masng.

```
// 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 });  
}  
  
graph.addEdge("A", "B", 5);  
graph.addEdge("A", "C", 12);  
graph.addEdge("A", "D", 7);  
  
graph.addEdge("B", "D", 1);  
graph.addEdge("B", "E", 6);  
  
graph.addEdge("C", "F", 13);  
  
graph.addEdge("D", "C", 1);  
graph.addEdge("D", "E", 5);
```

```
graph.addEdge("D", "F", 10);

graph.addEdge("E", "F", 2);
graph.addEdge("E", "G", 7);

graph.addEdge("F", "G", 3);
```

4. setelah label dan bobot tiap tiap node/ vertex didapat maka pencarian jalur tercepat dapat dilakukan dengan cara memanggil method Dijkstra(awal, tujuan), dengan dua parameter yaitu parameter awal dan parameter tujuan, method ini mengembalikan/return hasil dari pencarian jalur tercepat dengan tipe data array.

```
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);
  }
}

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 terakhir
      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();
}

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

```

5. Proses didalam method Dijkstra.

1. 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 dequeue atau diambil sementara.

2. menentukan path yang dikunjungi dengan perulangan, perulangan nya akan berhenti jika nilai label dengan nilai terkecil sama dengan label tujuan. *di point d dibawah
3. 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'.
4. 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