

IF2211 Strategi Algoritma

IMPLEMENTASI ALGORITMA UCS DAN A*
UNTUK MENENTUKAN
RUTE TERPENDEK

Laporan Tugas Kecil 3

Disusun untuk memenuhi tugas mata kuliah IF2211 Strategi Algoritma
pada Semester 2 (dua) Tahun Akademik 2022/2023



Oleh

Akbar Maulana Ridho 13521093

Arsa Izdihar Islam 13521101

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2023

DAFTAR ISI

DAFTAR ISI	1
BAB I	
DESKRIPSI MASALAH	2
1.1 Permasalahan Pencarian Pasangan Titik Terdekat	2
1.2 Spesifikasi Program	3
BAB II	
LANDASAN TEORI	4
2.1 Haversine Distance	4
2.2 Algoritma Uniform Cost Search	4
BAB III	
IMPLEMENTASI	6
3.1 Library Yang Digunakan	6
3.2 Implementasi Kelas Dasar	6
3.3 Algoritma Pencarian Rute Terpendek dengan Algoritma UCS dan A*	10
3.4 Implementasi Tampilan Peta	12
3.5 Implementasi Bonus	17
3.5.1 Pengambilan Data Map	17
3.5.2 Implementasi Tampilan Peta	17
BAB IV	
UJI COBA	23
4.1 Hasil Uji Coba Arbitrary Graph	23
4.1.1 Uji Coba 8 Simpul dengan Directed Graph	23
4.1.2 Uji Coba 10 Simpul dengan Directed Graph	23
4.1.3 Uji Coba 20 Simpul dengan Undirected Graph	24
4.1.4 Uji Coba 8 Simpul tanpa Edge	25
4.2 Hasil Uji Coba Peta Kota Bandung	26
4.2.1 Uji Coba 1	26
4.2.2 Uji Coba 2	28
4.2.3 Uji Coba 3	29
4.2.4 Uji Coba 4	29
LAMPIRAN	31
Lampiran 1 Link Repository GitHub	31
Lampiran 2 Tabel Check List Poin	31
DAFTAR REFERENSI	32

BAB I

DESKRIPSI MASALAH

1.1 Permasalahan Pencarian Pasangan Titik Terdekat

Algoritma UCS (Uniform cost search) dan A* (atau A star) dapat digunakan untuk menentukan lintasan terpendek dari suatu titik ke titik lain. Pada tugas kecil 3 ini, anda diminta menentukan lintasan terpendek berdasarkan peta Google Map jalan-jalan di kota Bandung. Dari ruas-ruas jalan di peta dibentuk graf. Simpul menyatakan persilangan jalan atau ujung jalan. Asumsikan jalan dapat dilalui dari dua arah. Bobot graf menyatakan jarak antar simpul. Jarak antar dua simpul dapat dihitung dari koordinat kedua simpul menggunakan rumus jarak euclidean atau dapat menggunakan ruler di Google Map, atau cara lainnya yang disediakan di Google Map.



Gambar 1.1.1 Ilustrasi Permasalahan Pencarian Rute Terpendek

(Sumber: Spesifikasi Tucil 3)

Langkah pertama di dalam program ini adalah membuat graf yang merepresentasikan peta. Berdasarkan graf yang dibentuk, lalu program menerima input simpul asal dan simpul tujuan, lalu menentukan lintasan terpendek antara keduanya menggunakan algoritma UCS dan A*. Lintasan terpendek dapat ditampilkan pada peta/ graf. Nilai heuristik yang dipakai adalah jarak garis lurus dari suatu titik ke tujuan.

1.2 Spesifikasi Program

Program yang dibangun menggunakan algoritma UCS dan A* untuk mencari rute terpendek. Program ini ditulis dalam bahasa Typescript dan ditranspile ke Javascript. Berikut adalah spesifikasi program.

1. Program menerima input file graf, jumlah simpul minimal 8 buah.
2. Program dapat menampilkan peta/ graf
3. Program menerima input simpul asal dan simpul tujuan
4. Program dapat menampilkan lintasan terpendek beserta jaraknya antara simpul asal dan simpul tujuan.
5. Antarmuka program bebas, apakah pakai GUI atau command line aja.

Spesifikasi bonus.

Bonus nilai diberikan jika dapat menggunakan Google Map API untuk menampilkan peta, membentuk graf dari peta, dan menampilkan lintasan terpendek di peta. Simpul graf diperoleh dari peta dengan mengklik ujung jalan atau persimpangan jalan, lalu jarak antara kedua simpul dihitung langsung dengan rumus Euclidean.

BAB II

LANDASAN TEORI

2.1 Haversine Distance

Haversine formula adalah rumus yang bisa digunakan untuk menghitung jarak terdekat antara dua titik pada permukaan berbentuk bola. Misalkan φ_1 adalah garis lintang (latitude) 1 dan λ_1 adalah garis bujur (longitude). Fungsi haversine didefinisikan sebagai berikut.

$$\text{hav}(\theta) = \sin^2(\theta/2) = (1 - \cos(\theta))/2$$

Jarak antar dua titik pada permukaan bola dapat dirumuskan sebagai berikut.

$$d = 2r \arcsin\left(\sqrt{\text{hav}(\varphi_2 - \varphi_1) + (1 - \text{hav}(\varphi_1 - \varphi_2) - \text{hav}(\varphi_1 + \varphi_2)) \cdot \text{hav}(\lambda_2 - \lambda_1)}\right)$$

dengan r adalah jari-jari bola (dalam hal ini bumi).

2.2 Algoritma Uniform Cost Search

Algoritma Uniform Cost Search (UCS) adalah salah satu algoritma pencarian rute terpendek yang bersifat *uninformed search*. Pada dasarnya, algoritma ini merupakan algoritma Breadth First Search (BFS) dan Iterative Deepening Search (IDS) dengan langkah terpendek. Algoritma ini melakukan proses pencarian yang dimulai dari root node, lalu melakukan percabangan berdasarkan cumulative cost terkecil. Algoritma ini diimplementasikan menggunakan struktur data priority queue. Berikut adalah langkah umum pada algoritma UCS.

1. Masukkan node awal ke prioqueue
2. Keluarkan elemen dengan prioritas tertinggi. Jika elemen yang dikeluarkan merupakan node tujuan, tampilkan total cost dan hentikan proses pencarian. Bila tidak, cek apakah node masuk ke dalam daftar node yang telah dikunjungi.
3. Jika tidak, masukkan semua children node tersebut ke dalam prioqueue, dengan nilai cumulative cost-nya merupakan penjumlahan nilai sebelumnya dengan cost menuju node yang saat ini dikunjungi.

2.3 Algoritma A*

Algoritma A* merupakan algoritma pencarian rute terpendek yang menggunakan pendekatan heuristik, sehingga merupakan algoritma yang bersifat informed search. Bisa dibilang, algoritma ini merupakan perluasan dari algoritma

UCS. Ide awal algoritma ini adalah untuk menghindari perluasan jalur yang sudah dirasa mahal.

Bila pada UCS kita menghitung cost untuk bergerak ke suatu node, algoritma A* tidak hanya menghitung cost tersebut, tetapi juga menghitung nilai estimasi/ heuristiknya. Cost functionnya adalah sebagai berikut.

$$g(n) = f(n) + h(n)$$

Dengan $f(n)$ adalah cost sampai saat ini yang diperlukan untuk mencapai n , $h(n)$ adalah estimasi cost untuk mencapai goal dari n , dan $g(n)$ adalah estimasi total cost dari path yang melalui n menuju goal. Fungsi heuristik h harus selalu underestimate nilai asli dari h agar hasil pencarian rute bisa dijamin optimal dan lengkap.

BAB III

IMPLEMENTASI

3.1 Library Yang Digunakan

Berikut adalah daftar tools/ library yang kami gunakan untuk menyelesaikan tugas kecil ini.

1. Bahasa pemrograman Typescript/ Javascript
2. Library React dan Daisy UI untuk front end
3. Cytospace untuk menampilkan graf
4. Priority Queue untuk implementasi priority queue
5. React google maps api dan use places autocomplete untuk menampilkan peta Google Maps
6. Haversine distance untuk menghitung jarak antar dua titik pada bidang berbentuk bola

3.2 Implementasi Kelas Dasar

Kami mengimplementasikan struktur koordinat dan kelas dasar untuk graf, seperti Graf dan Node.

Implementasi node berisi struktur koordinat dan fungsi untuk menghitung jarak antar dua titik dengan haversine distance. Implementasi Node direpresentasikan dengan kelas Node, yang memiliki nama, data, dan daftar edge-nya. Implementasi graf direpresentasikan dengan kelas Graph, yang memiliki daftar nodes serta beberapa fungsi pembantu.

coordinates.ts

```
import haversineDistance from 'haversine-distance'

export interface LatLngCoordinate {
    lat: number
    lng: number
}

export const calculateHaversineDistance = (a: LatLngCoordinate, b: LatLngCoordinate): number => {
    return haversineDistance(a, b)
}
```

nodes.ts

```
export type BasicNodeData = {
    minEdge: number
}
```

```

}

export type BasicNode = Node<BasicNodeData, number>

export class Node<T, U> {
    id: number
    data: T
    name: string
    adjacent: Map<number, {
        weight: U
    }>
    constructor(id: number, data: T, name?: string) {
        this.id = id
        this.data = data
        this.adjacent = new Map()
        if (name) {
            this.name = name
        } else {
            this.name = id.toString()
        }
    }
    get idString() {
        return this.id.toString()
    }
    addEdge(node: Node<T, U>, weight: U) {
        this.adjacent.set(node.id, {
            weight,
        })
    }
    removeEdge(node: Node<T, U>) {
        this.adjacent.delete(node.id)
    }
}

```

graph.ts

```

export type BasicGraph = Graph<BasicNodeData, number>

export class Graph<TData, TWeight> {
    nodes: Map<number, Node<TData, TWeight>>
    directed: boolean
    count: number

    constructor(directed = false) {
        this.nodes = new Map()
        this.directed = directed
        this.count = 0
    }
}

```

```

}

static fromString(str: string) {
  const graph = new Graph<BasicNodeData, number>(true)

  const lines = str.split(/\r?\n/)

  if (lines.length < 1) {
    throw new Error('Invalid input')
  }
  let lineIdx = 0

  const nodeCount = parseInt(lines[lineIdx++])
  if (isNaN(nodeCount)) {
    throw new Error('Invalid nodes count')
  }
  if (lines.length < nodeCount * 2 + 1) {
    throw new Error('Invalid input')
  }

  const names = lines.slice(lineIdx, lineIdx + nodeCount)

  const nodes: BasicNode[] = []

  for (let i = 0; i < nodeCount; i++) {
    nodes.push(graph.addNode({
      minEdge: Infinity,
    }, names[i]))
  }

  for (let i = 0; i < nodeCount; i++) {
    const line = lines[i + nodeCount + 1].split(/\s+/)
    for (let j = 0; j < nodeCount; j++) {
      const weight = parseInt(line[j])
      if (isNaN(weight) || weight < 0) {
        throw new Error(`Invalid weight at line ${i} column ${j}`)
      }
      if (weight > 0) {
        graph.addEdge(nodes[i], nodes[j], weight)
      }
    }
  }
  // calculate minimum edge weight of each node as node data for heuristic
  purpose
  nodes.forEach((node) => {
    let min = Infinity
    node.adjacent.forEach((value) => {
      if (value.weight < min) {
        min = value.weight
      }
    })
  })
}

```

```

        node.data.minEdge = min
    })
    return graph
}

addNodeWithId(id: number, data: TData, name?: string) {
    this.count++
    const newNode = new Node<TData, TWeight>(id, data, name)
    this.nodes.set(id, newNode)
    return newNode
}

addNode(data: TData, name?: string) {
    return this.addNodeWithId(this.count+1, data, name)
}

addEdge(from: Node<TData, TWeight>, to: Node<TData, TWeight>, weight: TWeight,
) {
    from.addEdge(to, weight)
    if (!this.directed) {
        to.addEdge(from, weight)
    }
}

getVisualizeData() {
    const elements: cytoscape.ElementDefinition[] = []

    this.nodes.forEach((node) => {
        elements.push({
            data: {
                id: node.idString,
                label: node.name
            }
        })
    })

    this.nodes.forEach((node) => {
        node.adjacent.forEach((value, key) => {
            const neighbor = this.nodes.get(key)!
            const undirected = neighbor.adjacent.get(node.id)?.weight ===
value.weight
            elements.push({
                data: {
                    source: node.idString,
                    target: key.toString(),
                    weight: value.weight,
                    id: `${node.idString},${key.toString()}`,
                    curve: undirected ? 'straight' : 'bezier',
                    arrow: undirected ? 'none' : 'triangle',
                }
            })
        })
    })
}

```

```

        })
    }

    return elements
}
}

```

3.3 Algoritma Pencarian Rute Terpendek dengan Algoritma UCS dan A*

Pada implementasi yang kami buat, algoritma UCS dan A* diimplementasikan menjadi satu algoritma yang sama, dengan pilihan antara UCS atau A* ditentukan melalui parameter pemanggilan fungsi.

Berikut adalah skema pencarian rute terpendek yang diimplementasikan.

1. Masukkan node asal ke queue
2. Ulangi hingga queue kosong atau cost semua elemen pada queue sudah lebih besar dibandingkan dengan solusi sekarang
 - a. Dequeue node dengan prioritas tertinggi
 - b. Jika sudah terdapat solusi dan semua elemen pada queue memiliki cost yang lebih besar dibandingkan dengan cost solusi sekarang, hentikan perulangan
 - c. Jika node yang sedang dikunjungi merupakan node tujuan dan jika tidak ada solusi sekarang atau cost solusi pada tahap pencarian ini lebih kecil daripada solusi sekarang, simpan solusi tersebut sebagai solusi terbaik saat ini
 - d. Sisanya, iterasi setiap simpul yang bersisian dengan simpul yang saat ini sedang dikunjungi. Pada kasus UCS, total cost adalah cost yang diperlukan untuk bergerak dari simpul awal menuju simpul saat ini. Pada kasus A*, total cost adalah cost yang diperlukan untuk bergerak dari simpul awal menuju simpul saat ini ditambah dengan estimasi cost dari simpul sekarang menuju simpul tujuan.

Pada uji kasus graf biasa (tanpa informasi koordinat/ lokasi), fungsi heuristik yang dipilih adalah nilai cost terkecil dari simpul sekarang menuju semua simpul yang bertetangga dengan simpul tersebut. Pada uji kasus peta dunia, fungsi heuristik yang dipilih adalah jarak haversine dari simpul sekarang menuju simpul tujuan.

Berikut adalah kode implementasi algoritma pencarian rute

```

type NodeC = Node<LatLngCoordinate, number> | BasicNode
type GraphC = Graph<LatLngCoordinate, number> | BasicGraph

function getHeuristic<T extends NodeC | BasicNode>(from: T, to: T): number {

```

```

if ('minEdge' in from.data && 'minEdge' in to.data) {
    // if the nodes are the same, heuristic value is 0
    if (from.id == to.id) return 0

    // else, heuristic value is the minimum edge weight of the node
    return from.data.minEdge
}

else if ('lat' in from.data && 'lat' in to.data) {
    return calculateHaversineDistance(from.data, to.data)
}

return 0
}

export class SearchNode {
    visited: Set<NodeC>

    constructor(public node: NodeC, public fx: number, beforeVisited: Set<NodeC>, public hx: number = 0) {
        this.visited = new Set(beforeVisited)
        this.visited.add(node)
    }

    get value() {
        return this.fx + this.hx
    }
}

const compareNodes: ICompare<SearchNode> = (a, b) => {
    return a.value - b.value
}

export function runAlgorithmRaw(graph: GraphC, start: NodeC, end: NodeC, isAstar = false) {
    const startTime = performance.now()
    const queue = new PriorityQueue(compareNodes)

    queue.enqueue(new SearchNode(start, 0, new Set()))

    let bestSolution: SearchNode | null = null
    let everythingElseBigger = false

    while (!queue.isEmpty() && !everythingElseBigger) {
        const searchNode = queue.dequeue()
        if (bestSolution && searchNode.value > bestSolution.value) {
            // No need to continue, we already have a better solution
            everythingElseBigger = true
            continue
        }
    }
}

```

```

const node = searchNode.node

if (node.id === end.id) {
  if (!bestSolution || searchNode.value < bestSolution.value) {
    bestSolution = searchNode
  }
  continue
}
node.adjacent.forEach((value, key) => {
  const nextNode = graph.nodes.get(key)
  if (nextNode && !searchNode.visited.has(nextNode)) {
    const fx = searchNode.fx + value.weight
    const hx = isAstar ? getHeuristic(nextNode, end) : 0
    queue.enqueue(new SearchNode(nextNode, fx, searchNode.visited, hx))
  }
})
}
return {solution: bestSolution, time: performance.now() - startTime}
}

export function runAlgorithm(graph: GraphC, start: NodeC, end: NodeC, isAstar = false) {
  const bestSolution = runAlgorithmRaw(graph, start, end, isAstar)

  if (!bestSolution.solution) {
    return null
  }
  return {...bestSolution, edges: getEdgesFromResult(bestSolution.solution)}
}

function getEdgesFromResult(searchNode: SearchNode) {
  const visited = searchNode.visited
  const edges: `${number},${number}>[]` = []
  let prev: Node<any, number> | null = null
  visited.forEach((node) => {
    if (prev) {
      edges.push(`${prev.id},${node.id}`)
    }
    prev = node
  })
  return edges
}

```

3.4 Implementasi Tampilan Peta

Berikut adalah halaman tampilan tempat yang diimplementasikan dengan React.

datatypes.h

```
function NormalPage({ navigate }: { navigate: (path: string) => void }) {
  const inputRef = useRef<HTMLInputElement>(null)
  const containerRef = useRef<HTMLDivElement>(null)
  const [graph, setGraph] = useState<BasicGraph | null>()
  const [start, setStart] = useState<number | undefined>(undefined)
  const [end, setEnd] = useState<number | undefined>(undefined)
  const [cy, setCy] = useState<cytoscape.Core | null>(null)
  const [isAStar, setIsAStar] = useState(false)
  const [distance, setDistance] = useState<number | null>(null)
  const [timeExec, setTimeExec] = useState<number | null>(null)

  function visualizeGraph(graph: BasicGraph) {
    if (!graph || !containerRef.current) return
    const elements = graph.getVisualizeData()
    const cy = cytoscape({
      container: containerRef.current!,
      elements,
      wheelSensitivity: 0.1,
      style: [
        {
          selector: 'node',
          style: {
            label: 'data(label)',
            backgroundColor: '#d1d5db',
            color: '#d1d5db',
          },
        },
        {
          selector: 'edge',
          style: {
            ... (omitted)
          },
        },
        {
          selector: '.solution',
          style: {
            'line-color': 'red',
            'target-arrow-color': 'red',
            width: 3,
          },
        },
      ],
    })
    setCy(cy)
  }

  function updateSolution(solution: SearchNode | null, solutionEdges: ${number}, ${number}[], timeExecution: number) {
    if (!cy) return
    cy.edges().forEach((edge) => {
```

```

        if (solutionEdges.includes(edge.id() as any) || (edge.data('arrow') ===
'none' && solutionEdges.includes(edge.id().split(',').reverse().join(',') as
any))) {
            edge.addClass('solution')
        } else {
            edge.removeClass('solution')
        }
    })
}

if (solution) {
    setDistance(solution.fx)
} else {
    setDistance(0)
}
setTimeExec(timeExecution)
}

return (
    <div className="h-screen w-full max-w-screen-2xl mx-auto flex flex-col
lg:flex-row items-center px-4 lg:space-x-4">
    <div className="flex flex-col w-full max-w-md">
        <div className="my-4">
            <BackButton onClick={() => navigate('home')} />
        </div>
        <input
            ref={inputRef}
            type="file"
            accept=".txt, text/plain"
            className="file-input file-input-accent"
            onChange={(e) => {
                const file = e.target.files?.item(0)
                if (file) {
                    const reader = new FileReader()
                    reader.onload = () => {
                        if (!reader.result) return
                        let graph: BasicGraph
                        if (typeof reader.result === 'string') {
                            graph = Graph.fromString(reader.result)
                        } else {
                            const arrayBuffer = reader.result as ArrayBuffer
                            const uint8Array = new Uint8Array(arrayBuffer)
                            const textDecoder = new TextDecoder()
                            const text = textDecoder.decode(uint8Array)
                            graph = Graph.fromString(text)
                        }
                        setDistance(null)
                        setTimeExec(null)
                        setGraph(graph)
                        setStart(1)
                        setEnd(graph.nodes.size)
                        visualizeGraph(graph)
                    }
                    reader.readAsText(file)
                }
            }}
        </input>
    </div>

```

```

        }
    } }
/>
<div className="form-control">
    <label htmlFor="algorithm" className="label">
        Algorithm
    </label>
    <select name="algorithm" className="select select-accent select-sm"
onChange={(e => setIsAStar(e.target.value === 'A*'))}>
        <option value="UCS">UCS</option>
        <option value="A*">A*</option>
    </select>
</div>

{graph && cy && (
<>
    <div className="grid grid-cols-2 gap-2">
        <div className="form-control">
            <label htmlFor="start" className="label">
                Start
            </label>
            <select
                name="start"
                value={start}
                onChange={(e) => {
                    setStart(+e.target.value)
                    setDistance(null)
                    setTimeExec(null)
                }}
                className="select select-accent select-sm"
            >
                {[...graph.nodes.keys()].map((nodeId) => (
                    <option value={nodeId} key={nodeId}>
                        {graph.nodes.get(nodeId)?.name}
                    </option>
                ))}
            </select>
        </div>
        <div className="form-control">
            <label htmlFor="end" className="label">
                End
            </label>
            <select
                name="end"
                value={end}
                onChange={(e) => {
                    setEnd(+e.target.value)
                    setDistance(null)
                    setTimeExec(null)
                }}
                className="select select-accent select-sm"
            >

```

```

        >
          { [...graph.nodes.keys()].map((nodeId) => (
            <option value={nodeId} key={nodeId}>
              {graph.nodes.get(nodeId)?.name}
            </option>
          )))
        </select>
      </div>
    </div>
    {start && end && (
      <button
        className="btn btn-accent btn-sm mt-4"
        onClick={() => {
          const solution = runAlgorithm(
            graph,
            graph.nodes.get(start)!, 
            graph.nodes.get(end)!, 
            isAStar
          )
          if (!solution) {
            return alert('No solution found')
          }
          updateSolution(solution.solution, solution.edges,
            solution.time)
        } }
      >
        Run
      </button>
    )}
  </>
)
)
{distance !== null && timeExec !== null ? <div className="card w-full bg-base-100 shadow-1">
  <div className="card-body">
    <h3 className="card-title">Waktu Eksekusi</h3>
    <p>{timeExec.toFixed(4)} ms</p>
    <h3 className="card-title">Jarak</h3>
    <p>{distance} satuan jarak</p>
  </div>
</div> : <></>}
</div>
<div
  className="w-full aspect-video border border-gray-200 rounded mt-4 mb-5 bg-gray-800"
  ref={containerRef}
></div>
</div>
)
}
export default NormalPage

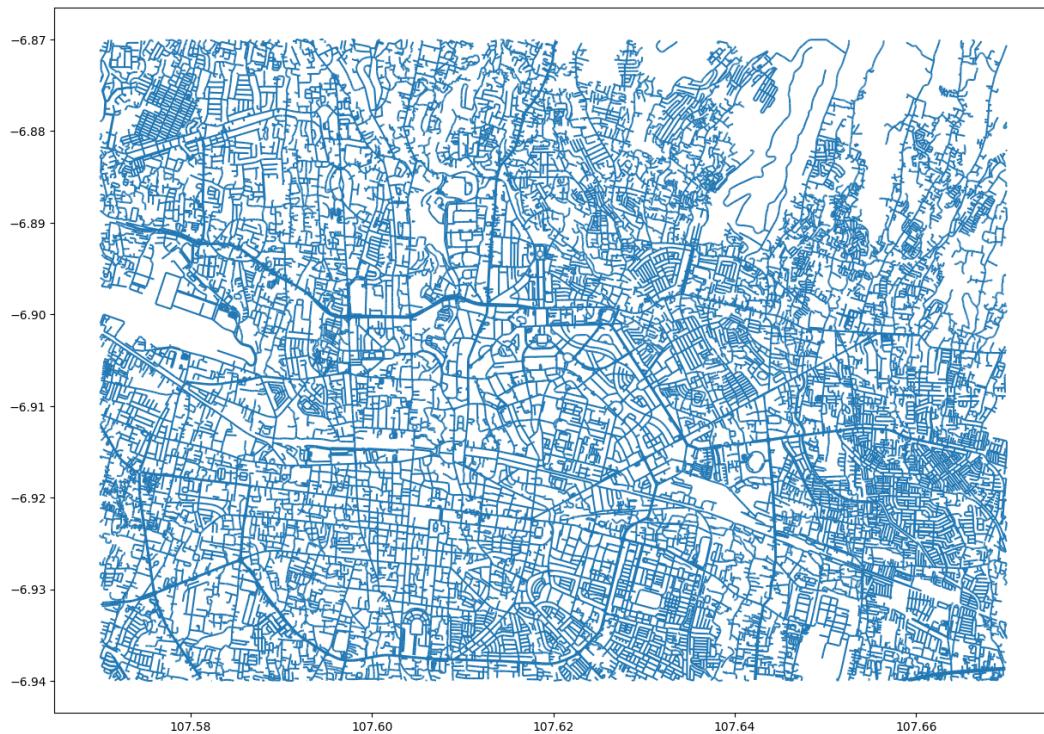
```

3.5 Implementasi Bonus

Kami melakukan sedikit improvisasi terkait spesifikasi bonus yang berkaitan dengan implementasi menggunakan Google Maps API. Alih-alih pengguna memasukkan sendiri simpul peta, kami mengambil sendiri seluruh data jalan yang ada di kota Bandung.

3.5.1 Pengambilan Data Map

Data jalan di Kota Bandung diambil dari data open source Open Street Maps. Kami mengunduh data pulau jawa lalu melakukan filter berdasarkan longitude dan latitude untuk mendapatkan peta Kota Bandung. Daerah yang diambil berada dalam garis lintang (latitude) -6.87 hingga -6.94 dan garis bujur (longitude) 107.57 hingga 107.67. Berikut adalah tampilan road network dari daerah yang kami ekstrak data jalannya.



Gambar 3.5.1.1 Peta Jalan Sebagian Kota Bandung

Sebagai catatan, kami tidak mengekstrak semua data jalan di Kota Bandung untuk menjaga ukuran dataset tidak terlalu berlebihan. Pada data tersebut, terdapat 90584 simpul dan 97259 sisi.

3.5.2 Implementasi Tampilan Peta

Untuk memasukkan titik awal dan titik tujuan, pengguna bisa memasukkan nama tempat dan memilih pilihan yang sesuai. Koordinat lokasi yang diinginkan pengguna akan dicari berdasarkan hasil pencarian tersebut. Lalu, akan dicariakan node terdekat dengan koordinat yang dimaksud. Setelah titik asal

dan tujuan ditentukan, proses pencarian rute terpendek akan dijalankan dan rute terpendek akan digambarkan pada peta. Berikut adalah kode implementasinya.

map-solver.ts

```
interface MapFormat {
    nodes: number[][]
    edges: number[][][]
}

export const loadGraphFromMap = (): Graph<Coordinate, number> => {
    const graph = new Graph<Coordinate, number>()

    fetch('/bandung.json').then(res => res.json()).then(data => {
        const map = data as MapFormat

        map!.nodes.forEach(node => {
            graph.addNodeWithId(node[0], {
                lat: node[1],
                lng: node[2]
            })
        })

        map!.edges.forEach(edge => {
            const from = graph.nodes.get(edge[0])!
            const to = graph.nodes.get(edge[1])!
            graph.addEdge(from, to, edge[2])
        })
    })

    return graph
}

export const closestNode = (graph: Graph<Coordinate, number>, target: Coordinate): Node<Coordinate, number> | null => {
    let closest: Node<Coordinate, number> | null = null
    let closestDistance: number = Number.POSITIVE_INFINITY
    graph.nodes.forEach(node => {
        if (closest === null) {
            closest = node
            closestDistance = calculateHaversineDistance(node.data, target)
        } else {
            const distance = calculateHaversineDistance(node.data, target)
            if (distance < closestDistance) {
                closest = node
                closestDistance = distance
            }
        }
    })
}
```

```
        return closest
    }
```

MapPage.tsx

```
function MapPage({navigate}: { navigate: (path: string) => void }) {
    const GMAPS_API_KEY = import.meta.env.VITE_GMAPS_API_KEY
    const libraries: Libraries = ['places']

    const {isLoaded} = useJsApiLoader({
        googleMapsApiKey: GMAPS_API_KEY,
        libraries,
        region: 'ID'
    })

    const center = useMemo(() => ({lat: -6.89148, lng: 107.6084704}), [])

    const [source, setSource] = useState<LatLngCoordinate | null>(null)
    const [destination, setDestination] = useState<LatLngCoordinate | null>(null)
    const [algorithm, setAlgorithm] = useState<'UCS' | 'A*'>('UCS')
    const [paths, setPaths] = useState<LatLngCoordinate[] | null>(null)
    const [distance, setDistance] = useState<number | null>(null)
    const [timeExec, setTimeExec] = useState<number | null>(null)

    const graph = loadGraphFromMap()
    const [isLoading, setIsLoading] = useState(false)

    const handleSearch = async () => {
        if (source === null || destination === null) {
            setPaths(null)
            setDistance(null)
            setTimeExec(null)
            return
        }

        const sourceNode = closestNode(graph, source)
        const destNode = closestNode(graph, destination)

        if (sourceNode === null || destNode === null) {
            setDistance(null)
            setTimeExec(null)
            setPaths(null)
            return
        }
        setIsLoading(true)
        const result = runAlgorithmRaw(graph, sourceNode, destNode, algorithm === 'A*')
        setIsLoading(false)
        if (result.solution === null) {
```

```

        setDistance(null)
        setTimeExec(null)
        setPaths(null)
        return
    }

    const coordinatePaths: LatLngCoordinate[] = []
    for (const path of result.solution.visited.values()) {
        if ('lat' in path.data) {
            coordinatePaths.push(path.data)
        }
    }
    setPaths(coordinatePaths)
    setDistance(result.solution.fx)
    setTimeExec(result.time)
}

const options = {
    strokeColor: '#FF0000',
    strokeOpacity: 0.8,
    strokeWeight: 2,
    fillColor: '#FF0000',
    fillOpacity: 0.35,
    clickable: false,
    draggable: false,
    editable: false,
    visible: true,
}

const sourceOptions: MarkerOptions = {
    title: 'Tempat awal'
}

const destOptions: MarkerOptions = {
    title: 'Tempat tujuan'
}

return (
    <div className="h-screen w-full max-w-screen-2xl mx-auto flex flex-col lg:flex-row items-center px-4 lg:space-x-4">
        <div className="flex flex-col w-full max-w-md p-8">
            <div className="my-4">
                <BackButton onClick={() => navigate('home')}/>
            </div>
            {!isLoaded ? <></> : (
                <div className="mx-auto w-full">
                    <div className="form-control">
                        <label htmlFor="algorithm" className="label">Algoritma</label>
                        <select name="algorithm" className="select select-accent" value={algorithm}>
                            <option value="ucs" selected>UCS</option>
                            <option value="astar">A*</option>
                            <option value="dijkstra">Dijkstra</option>
                            <option value="floyd">Floyd-Warshall</option>
                            <option value="bellmanFord">Bellman-Ford</option>
                            <option value="prim">Prim</option>
                            <option value="kruskal">Kruskal</option>
                        </select>
                    </div>
                </div>
            )}
        </div>
    </div>
)

```

```

'A*' }) >
    <option value="UCS">UCS</option>
    <option value="A*>A*</option>
  </select>
</div>
<PlaceSearch searchLabel="Apa tempat asal anda?">
placeholder="Tempat asal"
setResult={x => {
  setSource(x)
  setPaths(null)
  setDistance(null)
  setTimeExec(null)
}
} />
<PlaceSearch searchLabel="Apa tempat tujuan anda?">
placeholder="Tempat tujuan"
setResult={x => {
  setDestination(x)
  setPaths(null)
  setDistance(null)
  setTimeExec(null)
}
} />

<button className={'btn max-w-xs my-4 btn-accent' + (isLoading ? 'loading' : '')}>
  disabled={source === null || destination === null || isLoading}
  onClick={e => handleSearch()}>Mulai Pencarian
</button>

</div>
) }

{distance !== null && timeExec !== null ? <div className="card w-full bg-base-100 shadow-1">
  <div className="card-body">
    <h3 className="card-title">Waktu Eksekusi</h3>
    <p>{timeExec.toFixed(4)} ms</p>
    <h3 className="card-title">Jarak</h3>
    <p>{distance} meter</p>
  </div>
</div> : <></>}
</div>
 {!isLoaded ? <div className="w-full"></div> : (
  <GoogleMap mapContainerClassName="w-full h-full" center={center}>
  zoom={14}>
    {source === null ? <></> : (
      <MarkerF position={source} options={sourceOptions}>
    )
    {destination === null ? <></> : (
      <MarkerF position={destination} options={destOptions}>
    )
  )
)
)
}

```

```
        )
      {
        paths === null ? <></> : (
          <PolylineF path={paths} options={options}/>
        )
      }
    </GoogleMap>
  )
</div>
)
}

export default MapPage
```

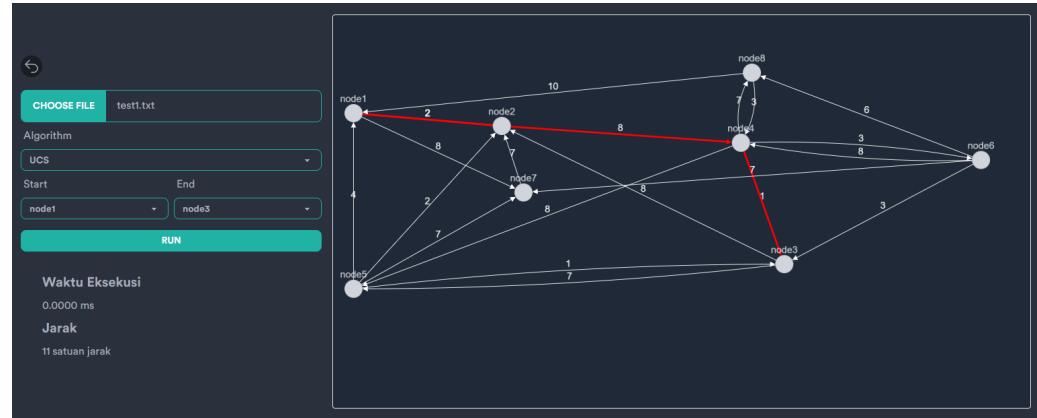
BAB IV

UJI COBA

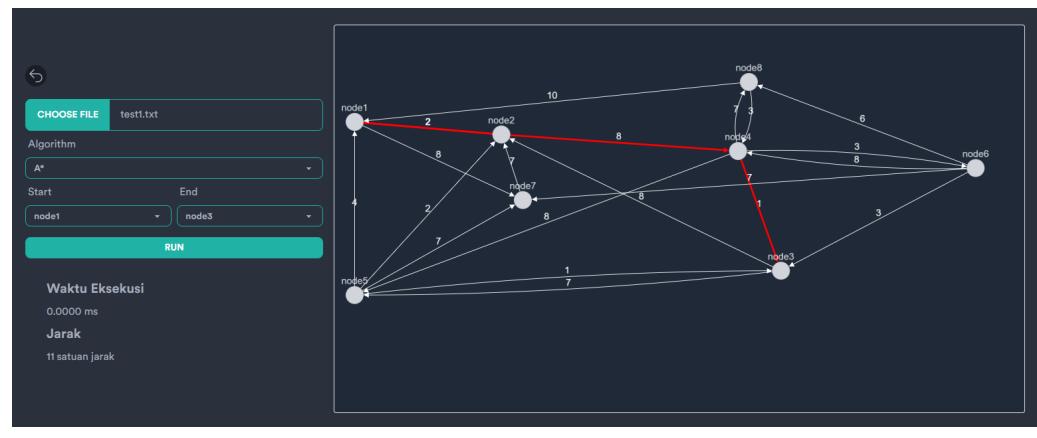
4.1 Hasil Uji Coba Arbitrary Graph

4.1.1 Uji Coba 1 dengan 8 simpul pada Directed Graph

Dengan Algoritma UCS

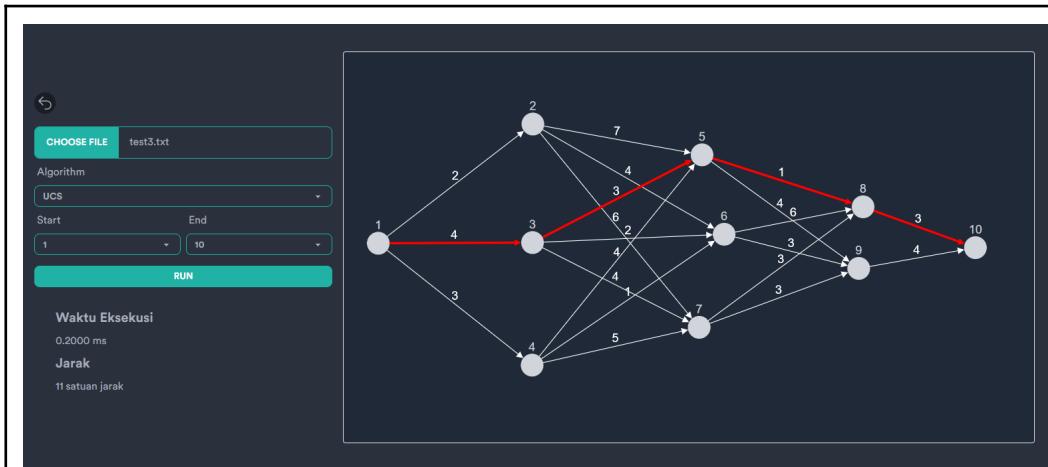


Dengan Algoritma A*

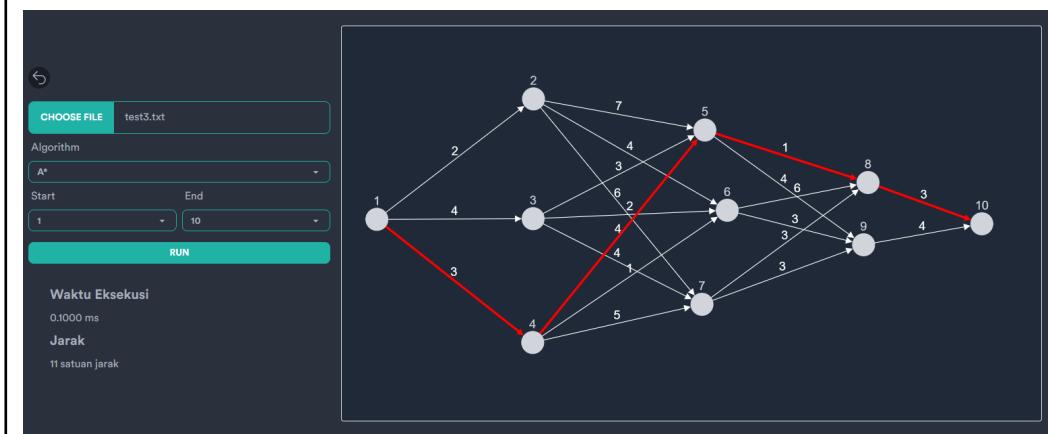


4.1.2 Uji Coba 2 dengan 10 simpul pada Directed Graph

Dengan Algoritma UCS



Dengan Algoritma A*

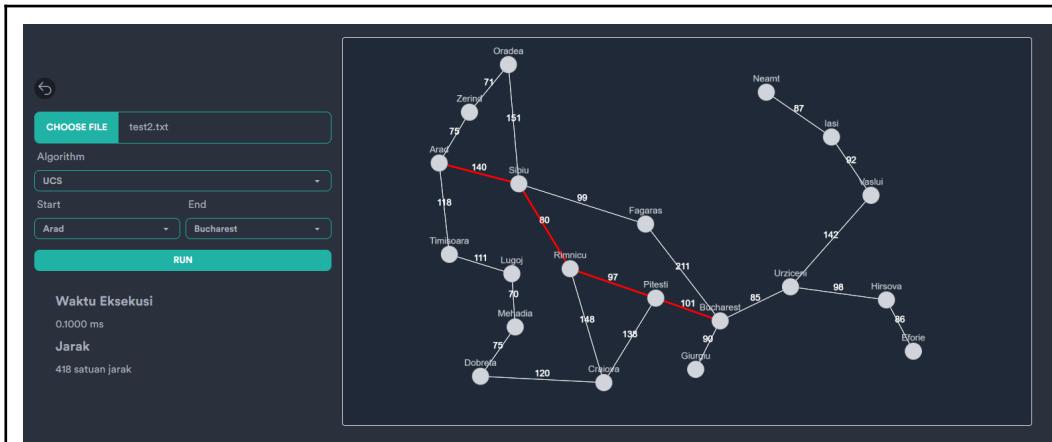


Komentar

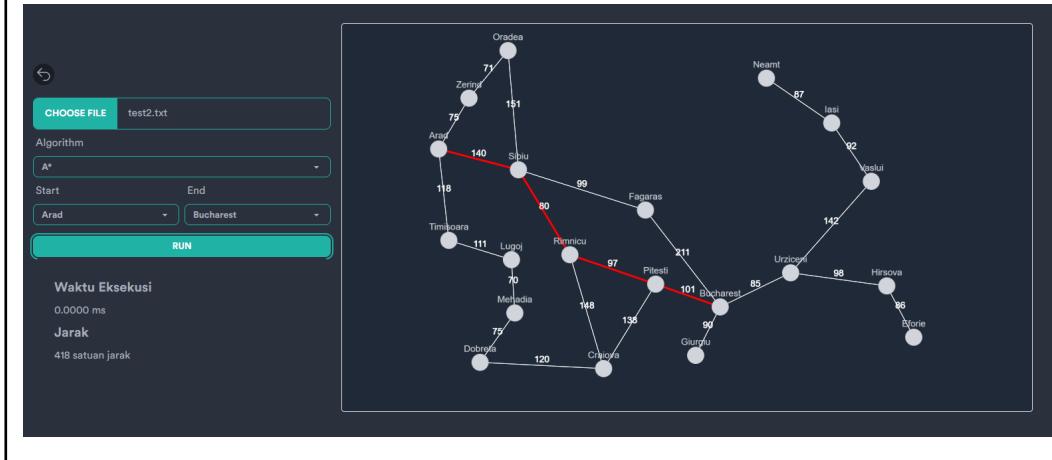
Rute dari kedua algoritma dapat berbeda, tetapi jarak yang ditempuh tetap sama.

4.1.3 Uji Coba 3 dengan 20 simpul pada Undirected Graph

Dengan Algoritma UCS

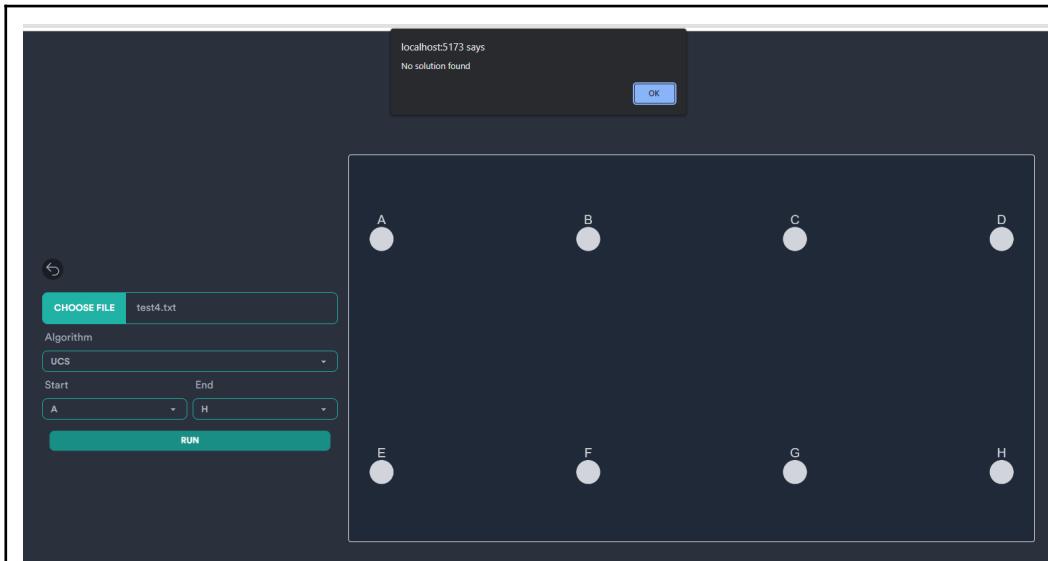


Dengan Algoritma A*

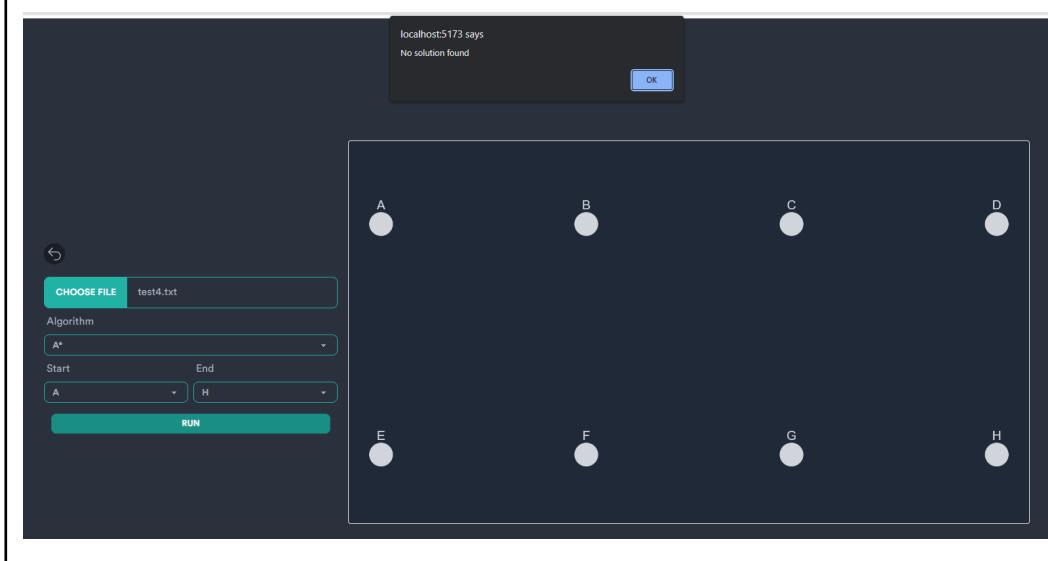


4.1.4 Uji Coba 4 dengan 8 simpul tanpa Edge

Dengan Algoritma UCS



Dengan Algoritma A*



4.2 Hasil Uji Coba Peta Kota Bandung

Berdasarkan beberapa percobaan berikut, terlihat bahwa Algoritma A* (informed search) jauh lebih efisien dibandingkan dengan Algoritma UCS. Pada satu kasus, perbedaan waktu eksekusinya sangat signifikan, sedangkan pada kasus lain Algoritma A* berhasil menemukan rute dalam kurun waktu yang masuk akal, sedangkan Algoritma UCS tidak.

4.2.1 Uji Coba 1

Dari Unpad Dipatiukur menuju Museum Geologi Bandung

Dengan Algoritma UCS

Algoritma
UCS

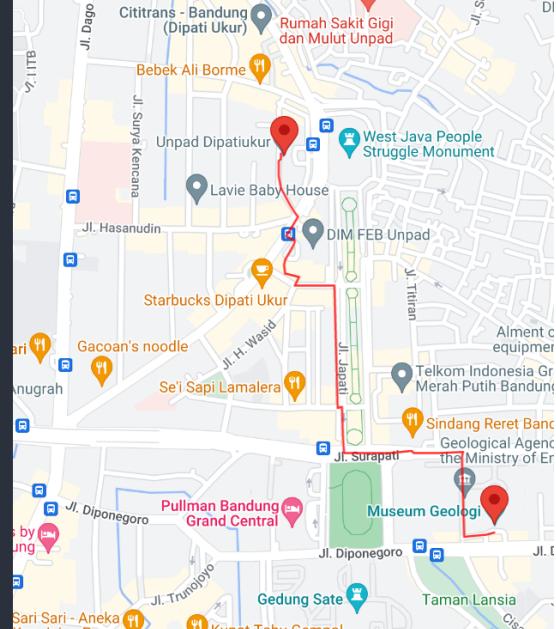
Apa tempat asal anda?
Unpad Dipatiukur, Jalan Dipati Ukur, Lebak Gede

Apa tempat tujuan anda?
Museum Geologi, Jalan Diponegoro, Cihaur Geu

MULAI PENCARIAN

Waktu Eksekusi
17370.1000 ms

Jarak
1344.8350000000003 meter



Dengan Algoritma A*

Algoritma
A*

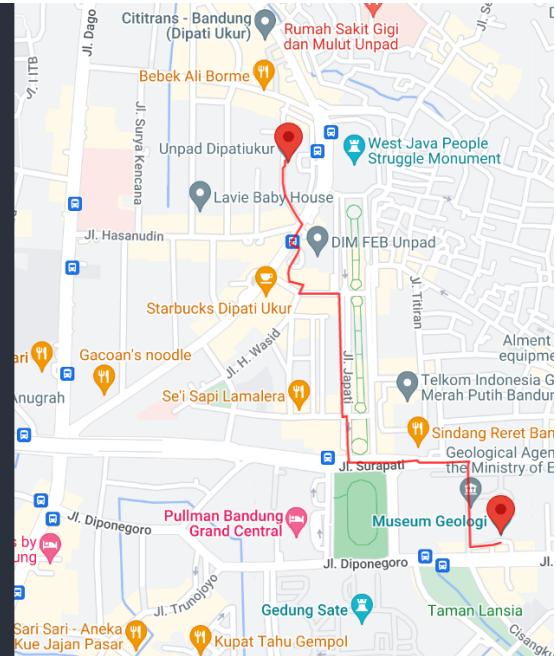
Apa tempat asal anda?
Unpad Dipatiukur, Jalan Dipati Ukur, Lebak Gede

Apa tempat tujuan anda?
Museum Geologi, Jalan Diponegoro, Cihaur Geu

MULAI PENCARIAN

Waktu Eksekusi
50.6000 ms

Jarak
1344.8350000000003 meter



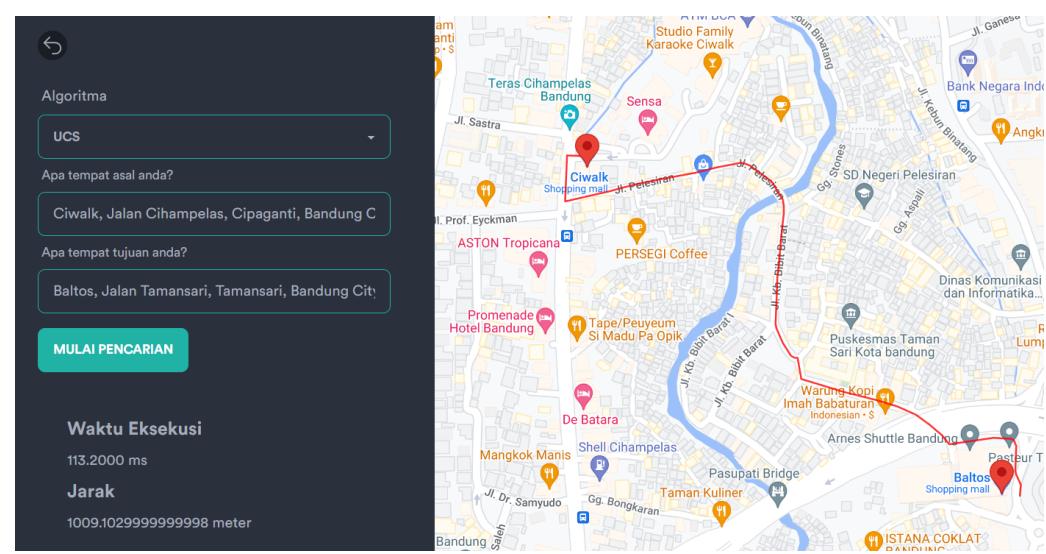
Keterangan:

Algoritma A* mampu menemukan rute terdekat dalam kurun waktu 50ms, dibandingkan dengan Algoritma UCS yang memerlukan waktu 17 detik.

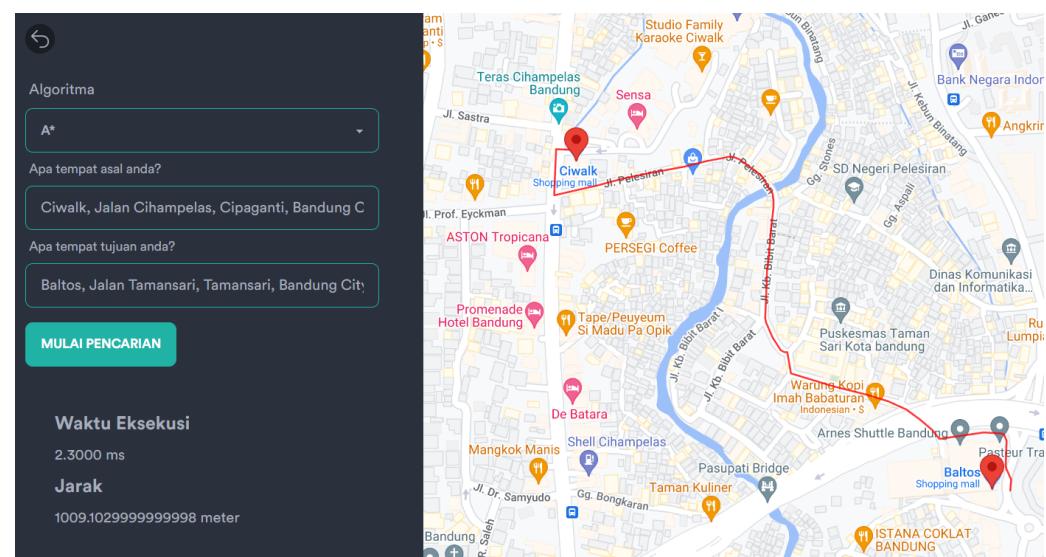
4.2.2 Uji Coba 2

Dari Ciwalk menuju Baltos

Dengan Algoritma UCS



Dengan Algoritma A*

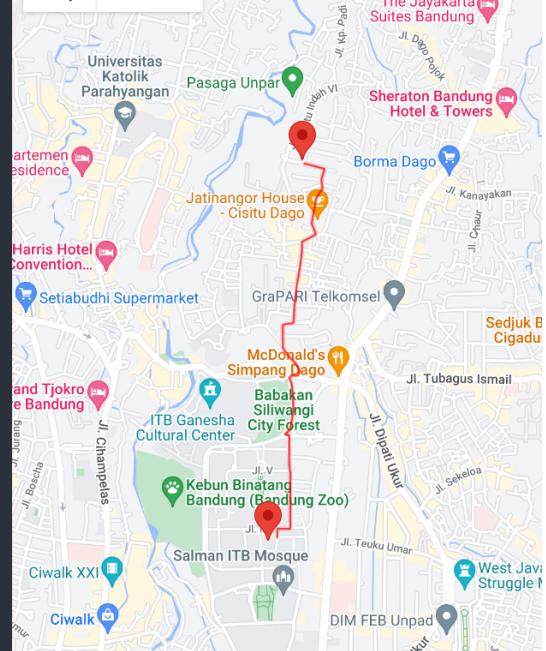


Keterangan:

Algoritma A* mampu menemukan rute terdekat dalam kurun waktu 2ms, dibandingkan dengan Algoritma UCS yang memerlukan waktu 100ms.

4.2.3 Uji Coba 3

Dari Jalan Cisitu Indah 2 No 14 menuju Pusat Institut Teknologi Bandung

Algoritma UCS	
Website unresponsive (crash)	
Algoritma A*	
Keterangan: Kasus cukup sulit, sehingga Algoritma UCS tidak dapat menemukan rute terpendek dalam waktu yang wajar, sedangkan Algoritma A* membutuhkan waktu 23ms.	

4.2.4 Uji Coba 4

Dari Alun-Alun Kota Bandung menuju Gedung Sate

Algoritma UCS	
Website unresponsive (crash)	
Algoritma A*	

Algoritma

A*

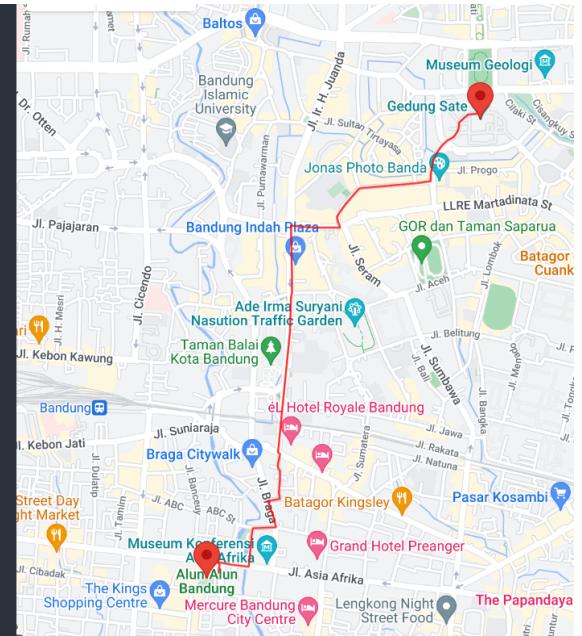
Apa tempat asal anda?

Alun-Alun Kota Bandung, Jalan Asia Afrika, Balor

Apa tempat tujuan anda?

Gedung Sate, Jalan Diponegoro, Citarum, Bandu

MULAI PENCARIAN



Waktu Eksekusi
25699.0000 ms

Jarak
3187.1440000000002 meter

Keterangan:
Kasus cukup sulit, sehingga Algoritma UCS tidak dapat menemukan rute terpendek dalam waktu yang wajar, sedangkan Algoritma A* membutuhkan waktu 25 detik.

30

IF2211 Strategi Algoritma

LAMPIRAN

Lampiran 1 Link Repository GitHub

https://github.com/arsaizdihar/Tucil3_13521093_13521101

Lampiran 2 Tabel *Check List Poin*

No.	Poin	Ya	Tidak
1.	Program dapat menerima input graf	√	
2.	Program dapat menghitung lintasan terpendek dengan UCS	√	
3.	Program dapat menghitung lintasan terpendek dengan A*	√	
4.	Program dapat menampilkan lintasan terpendek serta jaraknya	√	
5.	Bonus: program dapat menerima input peta dengan Google Map API dan menampilkan peta serta lintasan terpendek pada peta	√	

DAFTAR REFERENSI

- Munir, R. (2022). *Strategi Algoritma: Route Planning (Bagian 1)*. Dilansir dari Homepage Rinaldi Munir: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf>. Diakses pada 12 April 2023.
- Munir, R. (2022). *Strategi Algoritma: Route Planning (Bagian 2)*. Dilansir dari Homepage Rinaldi Munir: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf>. Diakses pada 12 April 2023.
- Upadhyay, Akshay. (2019). *Haversine Formula: Calculate geographic distance on earth..* Dilansir dari <https://www.igismap.com/haversine-formula-calculate-geographic-distance-earth/>. Diakses pada 12 April 2023.