LAPORAN TUGAS KECIL 2
IF2211 STRATEGI ALGORITMA

Implementasi Algoritma UCS dan A* untuk Menentukan Lintasan Terpendek

Disusun oleh:

Rinaldy Adin	13521134
Rava Maulana Azzikri	13521149

Program Studi Teknik

Informatika Sekolah Teknik

Elektro dan Informatika

Institut Teknologi Bandung

2023

I. Deskripsi Persoalan

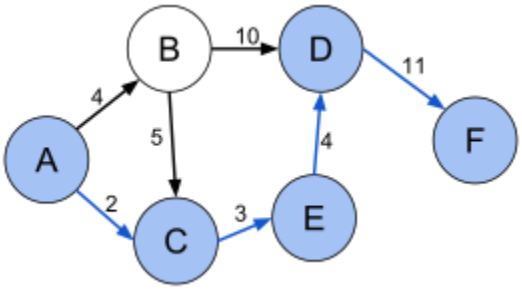
Pada tugas kecil 3 ini, anda diminta menentukan lintasan terpendek berdasarkan peta Google Map jalan-jalan di kota Bandung. Dari ruas-ruas jalan di peta dibentuk graf. Simpul menyatakan persilangan jalan (simpang 3, 4 atau 5) atau ujung jalan. Program akan membuat graf yang merepresentasikan sebuah peta. Berdasarkan graf yang dibentuk, lalu program menerima input simpul asal dan simpul tujuan, lalu menentukan lintasan terpendek antara keduanya menggunakan algoritma UCS dan A*.

Input program dapat berupa input file yang berisi informasi graf, atau berupa peta interaktif yang dapat diklik untuk membuat graf. Setelah graf dihasilkan dan ditampilkan, program akan menerima simpul yang dijadikan awal dan akhir, serta meminta metode pencarian yang ingin dipakai. Setelah itu, program akan menampilkan lintasan terpendek dari graf yang dibuat.

II. Dasar Teori

A. Pencarian Lintasan Terpendek pada Graf

Dalam teori graf, persoalan lintasan terpendek adalah persoalan yang bertujuan untuk menemukan lintasan melalui sebuah graf berbobot dengan jumlah bobot yang dilalui lintasan tersebut seminimal mungkin. Meskipun persoalan lintasan terpendek mudah diselesaikan dalam graf tidak berbobot menggunakan algoritma Breadth First Search (BFS), persoalan ini lebih sulit diselesaikan jika dilakukan terhadap graf yang memiliki bobot yang berbeda pada setiap sisinya. Meskipun lebih sulit, persoalan lintasan terpendek dalam graf berbobot seringkali perlu dilakukan dalam implementasi pathfinding di dunia nyata, seperti pada GPS atau aplikasi navigasi lainnya. Hal tersebut dikarenakan kondisi jalan di dunia asli yang tidak menentu sehingga perlu diberikan bobot-bobot yang berbeda pada setiap jalannya.



Gambar 1, visualisasi graf berbobot berarah
(Sumber: https://en.wikipedia.org/wiki/Shortest_path_problem)

Persoalan lintasan juga dapat digeneralisasikan secara lebih lanjut berdasarkan jumlah simpul asal, jumlah simpul tujuan, dan juga berdasarkan sifat-sifat grafnya, seperti apakah grafnya asiklik atau tidak, atau apakah grafnya berarah atau tidak. Pada persoalan ini, lintasan terpendek dicari antara satu simpul asal dan satu simpul tujuan

dalam suatu graf berbobot yang tidak terarah. Untuk menyelesaikan persoalan lintasan terpendek jenis ini, algoritma yang paling umum dan dikenali adalah algoritma Dijkstra.

Algoritma Dijkstra mengandalkan priority queue dalam penentuan simpul mana yang perlu dikunjungi. Dalam algoritma tersebut, priority queue akan menyimpan jarak antara setiap simpul dengan simpul asal, lalu akan memilih simpul terdekat berikutnya untuk dikunjungi. Algoritma tersebut akan berulang terus hingga semua simpul dalam graf sudah dikunjungi. Algoritma ini akan selalu menemukan solusi yang optimal, akan tetapi implementasinya masih kurang mangkus karena kompleksitasnya yang relatif tinggi, yaitu $O(V^2)$. Oleh karena itu, terdapat berbagai variasi dari algoritma Dijkstra yang lebih mangkus dengan memanfaatkan berbagai pendekatan. Program kami menggunakan Uniform Cost Search yang berupa varian dari Dijkstra yang lebih mangkus karena berhenti ketika sudah sampai simpul tujuan, serta algoritma A* yang merupakan varian Dijkstra yang memanfaatkan heuristik untuk mempercepat pencarian.

B. Algoritma Uniform Cost Search

Uniform cost search adalah algoritma uninformed yang menggunakan bobot kumulatif terkecil dalam suatu lintasan dari simpul asal ke simpul tujuan. Pencarian dan pengunjungan simpul melebar dari simpul awal berdasarkan bobot lintasan yang terkecil. Implementasi penyimpanan bobot terkecil tersebut seringkali memanfaatkan priority queue.

Priority queue pada awalnya hanya berisi simpul asal, lalu akan secara berulang menghilangkan dan mencatat elemen terdepan (simpul dengan jarak terkecil) pada priority queue tersebut. Tetangga dari simpul tersebut yang belum dikunjungi akan dimasukkan ke dalam priority queue dengan isi priority queue tetap terurut berdasarkan jarak simpul. Algoritma ini akan berulang hingga menemukan simpul tujuan, lalu program akan mengeluarkan lintasan terdekat dari simpul asal ke simpul tujuan. Karena urutan pengunjungan simpul berdasarkan jarak simpul dari simpul asal, maka algoritma ini selalu optimal karena kunjungan simpul tujuan yang terjadi pertama kali pasti menghasilkan lintasan dengan jarak terkecil.

C. Algoritma A*

Algoritma A* adalah algoritma pencarian lintasan yang merupakan variasi dari algoritma Dijkstra yang memanfaatkan heuristik untuk mengefisienkan pencarian. Algoritma ini optimal dan mangkus dalam menyelesaikan persoalan lintasan terpendek dalam suatu peta. Hal tersebut karena implementasi A* dalam pencarian lintasan terpendek dalam peta memanfaatkan heuristik berupa jarak euclidean antara simpul awal dengan simpul yang ingin dikunjungi. Heuristik tersebut akan membuat simpul-simpul yang dianggap jauh dari simpul awal dikunjungi lebih akhir karena diberikan prioritas yang lebih rendah daripada simpul-simpul yang diasumsikan berjarak lebih dekat.

Alur jalannya algoritma A* sama seperti algoritma Dijkstra atau Uniform Cost Search, tetapi ditambahkan implementasi nilai heuristik yang ditambahkan ke dalam bobot suatu simpul ketika dimasukkan ke dalam priority queue. Secara matematis, jika g(n) adalah total bobot dari simpul awal ke simpul n, serta h(n) adalah nilai heuristik yang dihitung berdasarkan simpul n, nilai yang akan digunakan sebagai pengurut node dalam priority queue adalah g(n) + h(n). Meskipun heuristik yang umum digunakan dalam algoritma A* adalah euclidean distance, heuristik lain dapat digunakan juga, seperti manhattan distance, atau heuristik lainnya. Pilihan heuristik yang digunakan tidak akan membuat algoritma A* menghasilkan solusi tidak optimal, asalkan heuristik yang dipilih masih admissible, yaitu heuristik yang dihasilkan masih lebih kecil daripada lintasan optimal.

III. Deskripsi Algoritma Yang Digunakan

A. Implementasi Algoritma Uniform Cost Search

Langkah-langkah pencarian rute menggunakan algoritma *Uniform Cost Search* adalah sebagai berikut,

1. Buat sebuah *priority queue* yang akan mengurutkan setiap *node* berdasarkan *cost* dari tersebut, lalu tambahkan *start node* pada *priority queue*.
2. Buat sebuah *array* yang menyimpan daftar *node* yang sudah dikunjungi.
3. Pop sebuah *node* dari *priority queue* dan simpan ke dalam variabel.
4. Untuk setiap *node* yang bersisian dengan *node* tersebut, tambahkan *cost* dari node yang bersisian dengan *cost* dari sisinya, lalu masukan ke dalam *priority queue*.
5. Tandai *node* yang baru saja di pop sudah dikunjungi.
6. Ulangi langkah kedua selama *end node* belum tercapai atau *priority queue* belum kosong.

B. Implementasi Algoritma A*

Dalam menyelesaikan persoalan lintasan terpendek menggunakan algoritma A*, kami menggunakan heuristik berupa euclidean distance antara suatu simpul dengan simpul awal/start. Alur algoritma kami adalah sebagai berikut,

1. Inisialisasikan array g_score untuk menyimpan jarak dari simpul awal ke suatu simpul i, array parent_node untuk menyimpan simpul asal dari setiap simpul, dan priority queue pq yang mengurutkan simpul yang akan dikunjungi berdasarkan g_score simpul tersebut ditambah dengan heuristik.
2. Isi priority queue dengan simpul awal. Karena g_score simpul awal pasti 0, maka prioritas simpul awal berdasarkan heuristik, yang juga bernilai 0.
3. Keluarkan dan catat node terdepan di priority queue, jika simpul merupakan simpul tujuan, kalkulasikan jarak dan lintasan berdasarkan array parent_node dan kembalikan nilai tersebut dan hentikan program.
4. Jika simpul bukan simpul tujuan, periksa semua simpul tetangganya, jika simpul tetangga belum dikunjungi atau jika sudah dikunjungi tetapi ditemukan lintasan

yang lebih cepat daripada lintasan yang sebelumnya, tambahkan simpul tetangga tersebut ke dalam priority queue berdasarkan g_score dan nilai heuristiknya.

5. Ulangi langkah 3 dan 4 hingga ditemukan simpul tujuan. Jika priority queue kosong dan belum ditemukan simpul tujuan, bangkitkan error.

IV. Source Code Program

index.tsx

```tsx
import React from "react";
import ReactDOM from "react-dom/client";
import "./index.css";
import App from "./components/App";

const root = ReactDOM.createRoot(
  document.getElementById("root") as HTMLElement
);
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

App.tsx

```tsx
import { useEffect, useState } from "react";
import { fileReader, graphFromList } from "../functions/Utility";
import CytoGraph from "../api/Cytoscape";
import clsx from "clsx";
import Graph from "../classes/Graph";
import Dropzone from "./Dropzone";
import Dropdown, { DropdownOptions } from "./Dropdown";
import Map from "../api/Leaflet";
import { coordinate, mapEdge } from "../classes/Graph";

import Button from "./Button";
import uniformCostSearch from "../functions/UniformCostSearch";
import aStar from "../functions/AStar";
```

```tsx
export default function App(): JSX.Element {
  const [isGmap, setGmap] = useState<boolean>(false);
  const [graph, setGraph] = useState<Graph | null>(null);
  const [nodeOptions, setNodeOptions] = useState<DropdownOptions[]>([
    { value: "none", text: "None Selected" },
  ]);
  const [sourceNode, setSourceNode] = useState<string>("none");
  const [destNode, setDestNode] = useState<string>("none");
  const [pathMethod, setPathMethod] = useState<string>("A*");
  const [nodeList, setNodeList] = useState<coordinate[]>([]);
  const [edgeList, setEdgeList] = useState<mapEdge[]>([]);
  const [graphPath, setGraphPath] = useState<number[]>([]);
  const [pathDist, setPathDist] = useState<number>(0);
  const [pathStr, setPathStr] = useState<string>("");
  const [fileErrorMsg, setFileErrorMsg] = useState<string>("");
  const [searchErrorMsg, setSearchErrorMsg] = useState<string>("");

  useEffect(() => {
    const newNodeOptions = [{ value: "none", text: "None Selected" }];
    if (graph) {
      for (let idx = 0; idx < graph.getVertexCount(); idx++) {
        newNodeOptions.push({ value: idx.toString(), text: `Node ${idx}` });
      }
    }
    setNodeOptions(newNodeOptions);
  }, [graph]);

  const handleFileChange = (file: File) => {
    if (!file.name.endsWith(".txt")) {
      setFileErrorMsg("Files should end with .txt!");
      return;
    }
    const reader = new FileReader();

    reader.onload = () => {
      try {
        setGraph(fileReader(reader.result as string));
        setFileErrorMsg("");
```

```
    } catch (error: any) {
      setFileErrorMsg(error.message);
    }
  };

  reader.readAsText(file);
};

const handleSearch = () => {
  let dist = 0;
  let path: number[] = [];

  try {
    if (sourceNode === "none" || destNode === "none")
      throw Error("Please set your source and destination nodes!");

    if (pathMethod === "A*" && graph)
      [dist, path] = aStar(parseInt(sourceNode), parseInt(destNode), graph);
    if (pathMethod === "UCS" && graph)
      [dist, path] = uniformCostSearch(
        parseInt(sourceNode),
        parseInt(destNode),
        graph
      );
  } catch (error: any) {
    setSearchErrorMsg(error.message);
    return;
  }

  setGraphPath(path);
  setPathDist(dist);

  let str = "";
  for (let idx = 1; idx < path.length; idx++) {
    str += graph?.getStreetName(path[idx - 1], path[idx]);
    if (idx !== path.length - 1) str += " - ";
  }
```

```
    setPathStr(str);
    setSearchErrorMsg("");
  };

  return (
    <main className="pb-4">
      <div className="bg-[#94C5CC] w-full py-4 px-9">
        <h1 className="text-[#000100] font-black text-4xl">PathFinder</h1>
      </div>
      <div className="flex w-[95vw] mx-auto">
        <div className="pt-4">
          <div className="flex items-center">
            <button
              onClick={() => {
                setGmap(!isGmap);
                setEdgeList([]);
                setNodeList([]);
                setGraph(null);
              }}
              className={clsx(
                "w-[74px] h-[42px] rounded-[35px] border-2 border-[#79747E]
relative",
                isGmap ? "bg-[#94C5CC]" : "bg-[#E0E9EC]"
              )}
            >
              <div
                className={clsx(
                  "w-[22px] h-[22px] rounded-full absolute top-2.5",
                  isGmap ? "right-3 bg-white" : "left-3 bg-[#A1A6B4]"
                )}
              ></div>
            </button>
            <h2 className="uppercase text-[#000100] font-black text-3xl ml-4">
              Gmaps
            </h2>
          </div>
          <div className="h-[71vh] w-[55vw] border-4 border-[#94C5CC]
rounded-md mt-4 p-4">
```

```
          {isGmap ? (
            <Map
              addNode={(coord: coordinate) =>
                setNodeList([...nodeList, coord])
              }
              addEdge={(edge: mapEdge) => setEdgeList([...edgeList, edge])}
            />
          ) : (
            graph && <CytoGraph graphPath={graphPath} graph={graph} />
          )}
        </div>
      </div>

      <div className="pt-[74px] px-5 flex-1">
        {isGmap ? (
          <>
            <h2 className="text-[#000100] mb-4 font-black text-3xl">
              Input Markers
            </h2>
            <Button
              onClick={() => setGraph(graphFromList(nodeList, edgeList))}
            >
              Set as Graph
            </Button>
          </>
        ) : (
          <>
            <h2 className="text-[#000100] mb-4 font-black text-3xl">
              File Input
            </h2>
            <Dropzone id="file-dropzone" onFileChange={handleFileChange} />
            <p
              className={clsx(
                "text-sm text-red-600",
                fileErrorMsg === "" && "invisible"
              )}
            >
              {fileErrorMsg} Please choose another file.
```

```
            </p>
          </>
        )}
        <div className="flex flex-col mt-4 gap-3">
          <Dropdown
            label="Source Node"
            value={sourceNode}
            onChange={(e) => {
              setSourceNode(e.target.value);
            }}
            options={nodeOptions}
          />
          <Dropdown
            label="Destination Node"
            value={destNode}
            onChange={(e) => {
              setDestNode(e.target.value);
            }}
            options={nodeOptions}
          />
          <Dropdown
            label="Pathfinding Method"
            value={pathMethod}
            onChange={(e) => {
              setPathMethod(e.target.value);
            }}
            options={[
              { value: "A*", text: "A*" },
              { value: "UCS", text: "UCS" },
            ]}
          />
        </div>
        <div className="flex mt-5 gap-2">
          <Button onClick={handleSearch}>Start Search</Button>
          <p
            className={clsx(
              "text-sm text-red-600 flex-1",
              searchErrorMsg === "" && "invisible"
```

```
            )}
          >
            {searchErrorMsg}
          </p>
        </div>
        {graphPath.length > 0 && (
          <div className="text-[#000100] text-xl  my-4">
            <h2 className="font-black text-3xl">Result</h2>
            <h4 className="font-bold mt-4">Shortest Path</h4>
            <p>{pathStr}</p>
            <h4 className="font-bold mt-4">Distance:</h4>
            <p>{pathDist}</p>
          </div>
        )}
      </div>
    </div>
  </main>
 );
}
```

Cytoscape.tsx

```
import { useEffect, useState } from "react";
import cytoscape from "cytoscape";
import Graph from "../classes/Graph";

let fcose = require("cytoscape-fcose");

type CytoGraphProps = {
  classname?: string;
  graph: Graph;
  graphPath: number[];
};

export default function CytoGraph({
  graph,
```

```
  graphPath,
}: CytoGraphProps): JSX.Element {
  const [cyto, setCyto] = useState<cytoscape.Core>();

  useEffect(() => {
    clearArrows();
    for (let idx = 1; idx < graphPath.length; idx++) {
      const source = graphPath[idx - 1];
      const target = graphPath[idx];

      if (
        cyto &&
        cyto
          .edges()
          .map((x) => x.id())
          .includes(`${target + "-" + source}`)
      )
        cyto.remove(`#${target + "-" + source}`);

      if (
        cyto &&
        cyto
          .edges()
          .map((x) => x.id())
          .includes(`${source + "-" + target}`)
      )
        cyto.remove(`#${source + "-" + target}`);

      if (cyto)
        cyto.add({
          group: "edges",
          data: {
            id: `${source + "-" + target}`,
            source: source.toString(),
            target: target.toString(),
            label: graph.getWeight(source, target),
          },
          classes: "directed",
```

```
        });
    }

    if (graphPath.length > 1 && cyto) cyto.fit();
  }, [graphPath]);

  useEffect(() => {
    cytoscape.use(fcose);

    const elements = graph.generateCytoElements();
    const fcoseLayout = {
      name: "fcose",
      animate: false,
    };

    const cy = cytoscape({
      container: document.getElementById("cy-container"), // container to
render in

      elements: elements,

      style: [
        // the stylesheet for the graph
        {
          selector: "node",
          style: {
            "border-width": 1,
            "border-style": "solid",
            "border-color": "#000100",
            "background-color": "#B4D2E7",
            width: 20,
            height: 20,
            label: "data(id)",
            "font-size": "0.5em",
            "text-valign": "center",
            "text-halign": "center",
            "font-weight": "bold",
          },
```

```
      },
      {
        selector: "edge",
        style: {
          width: 1,
          "line-color": "#000000",
          // "target-arrow-color": "#FF0000",
          // "target-arrow-shape": "triangle",
          "curve-style": "bezier",
        },
      },
      {
        selector: "edge[label]",
        style: {
          label: "data(label)",
          "font-size": "0.6em",
          "text-valign": "top",
          "text-halign": "center",
          "text-background-color": "#fff",
          "text-background-opacity": 1,
        },
      },
      {
        selector: ".directed",
        style: {
          "target-arrow-color": "#000",
          "target-arrow-shape": "triangle-backcurve",
        },
      },
    ],

    layout: fcoseLayout,
    boxSelectionEnabled: false,
    userPanningEnabled: false,
    userZoomingEnabled: false,
  });

  cy.fit();
```

```
    setCyto(cy);
  }, [graph]);

  const clearArrows = () => {
    if (cyto) {
      cyto.edges().removeClass("directed");
    }
  };

  return <div className="w-full h-full" id="cy-container"></div>;
}
```

Leaflet.tsx

```
import { LatLng, LatLngExpression } from "leaflet";
import { useState } from "react";
import {
  MapContainer,
  TileLayer,
  useMapEvents,
  Marker,
  Tooltip,
  Polyline,
} from "react-leaflet";
import { coordinate, mapEdge } from "../classes/Graph";

interface markerProps {
  addNode: (coord: coordinate) => void;
  addEdge: (edge: mapEdge) => void;
}

function Markers({ addNode, addEdge }: markerProps): JSX.Element {
  const [posArr, setPosArr] = useState<LatLng[]>([]);
  const [currPos, setCurrPos] = useState<LatLng>();
  const [polyArr, setPolyArr] = useState<LatLngExpression[][]>([]);
  const [currIdx, setCurrIdx] = useState<number>(0);
```

```
useMapEvents({
  click(e) {
    if (posArr.length > 0 && currPos) {
      setPolyArr([
        ...polyArr,
        [
          [currPos.lat, currPos.lng],
          [e.latlng.lat, e.latlng.lng],
        ],
      ]);
      const edge: mapEdge = {
        source: currIdx,
        dest: posArr.length,
        sourceCoord: {
          lat: posArr[currIdx].lat,
          long: posArr[currIdx].lng,
        },
        destCoord: {
          lat: e.latlng.lat,
          long: e.latlng.lng,
        },
      };
      addEdge(edge);
    } else if (posArr.length === 0) {
      setCurrPos(e.latlng);
    }
    setPosArr([...posArr, e.latlng]);
    const coord: coordinate = {
      long: e.latlng.lng,
      lat: e.latlng.lat,
    };
    addNode(coord);
  },
});

return (
  <>
```

```
      {posArr.map((pos, idx) => (
        <Marker
          key={idx}
          position={pos}
          eventHandlers={{
            click: (e) => {
              setCurrPos(e.latlng);
              setCurrIdx(idx);
            },
          }}
        >
          <Tooltip direction="right" offset={[0, 0]} opacity={1} permanent>
            {idx === 0 ? "0" : idx}
          </Tooltip>
        </Marker>
      ))}
      {polyArr.map((opt, idx) => (
        <Polyline positions={opt} key={idx} />
      ))}
    </>
  );
}

export default function Map({ addNode, addEdge }: markerProps): JSX.Element {
  return (
    <MapContainer
      center={[-6.893127, 107.610386]}
      zoom={16}
      scrollWheelZoom={false}
    >
      <TileLayer
        attribution='&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
      />
      <Markers addNode={addNode} addEdge={addEdge}></Markers>
    </MapContainer>
  );
```

```
}
```

Graph.tsx

```
import { ElementDefinition } from "cytoscape";

export type coordinate = {
  long: number;
  lat: number;
};

export type mapEdge = {
  source: number;
  dest: number;
  sourceCoord: coordinate;
  destCoord: coordinate;
};

type matrixEntry = {
  weight: number;
  street: string;
};

export default class Graph {
  private vertexCount: number;
  private adjMatrix: matrixEntry[][];
  private coordList: coordinate[];

  constructor(vertexCount: number) {
    this.vertexCount = vertexCount;
    this.adjMatrix = [];
    this.coordList = [];

    for (let i = 0; i < vertexCount; i++) {
      this.adjMatrix[i] = [];
      for (let j = 0; j < vertexCount; j++) {
```

```
        this.adjMatrix[i][j] = {
          weight: 0,
          street: "",
        };
      }
    }
  }

  public getVertexCount(): number {
    return this.vertexCount;
  }

  public getStreetName(v1: number, v2: number): string {
    return this.adjMatrix[v1][v2].street;
  }

  public getWeight(v1: number, v2: number): number {
    return this.adjMatrix[v1][v2].weight;
  }

  public getCoord(vertex: number): coordinate {
    return this.coordList[vertex];
  }

  public addVertex(idx: number, long: number, lat: number) {
    this.coordList[idx] = { long, lat };
  }

  public addEdge(v1: number, v2: number, weight: number, street: string) {
    this.adjMatrix[v1][v2] = { weight, street };
    this.adjMatrix[v2][v1] = { weight, street };
  }

  public generateCytoElements(): ElementDefinition[] {
    let elements: ElementDefinition[] = [];

    for (let idx = 0; idx < this.getVertexCount(); idx++) {
      elements.push({
```

```
        group: "nodes",
        data: {
          id: idx.toString(),
        },
      });
    }

    for (let i = 0; i < this.vertexCount; i++) {
      for (let j = i; j < this.vertexCount; j++) {
        if (this.getWeight(i, j) !== 0)
          elements.push({
            group: "edges",
            data: {
              id: i.toString() + "-" + j.toString(),
              source: i.toString(),
              target: j.toString(),
              label: this.getWeight(i, j),
            },
          });
      }
    }

    return elements;
  }
}
```

Button.tsx

```
import clsx from 'clsx';
import React from 'react';

interface ButtonProps extends React.ButtonHTMLAttributes<HTMLButtonElement> {}

export default function Button({children, className, ...props}: ButtonProps):
JSX.Element {
    return (
```

```
        <button
            className={clsx("px-6 py-2 text-xl shadow-md rounded-md text-center
bg-[#94C5CC] hover:bg-[#8DBBC2] active:bg-[#87B4BA] hover:cursor-pointer",
className !== undefined && className)}
            {...props}
        >
            {children}
        </button>
    );
}
```

Dropdown.tsx

```
import React from "react";
import clsx from "clsx";

interface DropdownOptions {
  value: string;
  text: string;
}

interface DropdownProps extends React.SelectHTMLAttributes<HTMLSelectElement> {
  label: string;
  options: DropdownOptions[];
}

export default function Dropdown({
  label,
  options,
  className,
  ...props
}: DropdownProps): JSX.Element {
  return (
    <div
      className={clsx(
```

```
        "w-full flex justify-between items-center",
        className !== undefined && className
      )}
    >
      <h4 className="text-xl font-bold">{label}</h4>
      <select
        className="rounded-[4px] border-2 w-1/2 border-[#A1A6B4] text-gray-700
px-4 outline-none py-2"
        {...props}
      >
        {options.map(({ value, text }) => (
          <option key={`option-${value}`} value={value}>{text}</option>
        ))}
      </select>
    </div>
  );
}

export type { DropdownOptions };
```

Dropzone.tsx

```
import Button from "./Button";
import React, { useRef, useState } from "react";
import clsx from "clsx";

interface DropzoneProps extends React.InputHTMLAttributes<HTMLInputElement> {
  onFileChange: (file: File) => void;
}

export default function Dropzone({
  onChange,
  onFileChange,
  ...props
}: DropzoneProps): JSX.Element {
```

```
const [filename, setFilename] = useState("");
const [dragActive, setDragActive] = React.useState(false);

const inputRef = useRef<HTMLInputElement>(null);

const handleChage: React.ChangeEventHandler<HTMLInputElement> = (event) => {
  event.preventDefault();
  event.stopPropagation();
  if (event.target.files && event.target.files[0] !== undefined) {
    onFileChange(event.target.files[0]);
    setFilename(event.target.files[0].name);
  }
  if (onChange) onChange(event);
};

const handleDrag: React.DragEventHandler<HTMLDivElement> = (event) => {
  event.preventDefault();
  event.stopPropagation();
  if (event.type === "dragenter" || event.type === "dragover") {
    setDragActive(true);
  } else if (event.type === "dragleave") {
    setDragActive(false);
  }
};

const handleDragOver: React.DragEventHandler<HTMLDivElement> = (event) => {
  event.preventDefault();
  event.stopPropagation();
};

const handleDrop: React.DragEventHandler<HTMLDivElement> = (event) => {
  event.preventDefault();
  event.stopPropagation();
  setDragActive(false);
  if (event.dataTransfer.files && event.dataTransfer.files[0]) {
    onFileChange(event.dataTransfer.files[0]);
    setFilename(event.dataTransfer.files[0].name);
  }
```

```
  };

  return (
    <div
      className={clsx(
        "flex items-center justify-center w-full border-2 border-[#A1A6B4]
border-dashed rounded-xl",
        dragActive && "bg-gray-100"
      )}
      onDragEnter={handleDrag}
      onDragLeave={handleDrag}
      onDragOver={handleDragOver}
      onDrop={handleDrop}
    >
      <div className="flex flex-col gap-2 items-center justify-center py-8">
        <h4>
          {filename === ""
            ? "Drop your files here"
            : '"' + filename + '" chosen'}
        </h4>
        <div className="text-center w-full text-[#A1A6B4] text-sm">or</div>
        <Button
          onClick={() => {
            inputRef.current?.click();
          }}
        >
          {filename === "" ? "Choose File" : "Choose Another File"}
        </Button>
      </div>
      <input
        onChange={handleChage}
        ref={inputRef}
        type="file"
        className="hidden"
        accept=".txt"
        {...props}
      />
    </div>
```

```
  );
}
```

AStar.tsx

```
import Graph from "../classes/Graph";
import { dequeuePQ, enqueuePQ, type PQElement, findNodeIdxPQ, euclideanDistance
} from "./Utility";

function heur(graph: Graph, start: number, v: number): number {
  return euclideanDistance(graph.getCoord(start), graph.getCoord(v));
}

function constructPath(
  start: number,
  end: number,
  parent_node: number[],
  graph: Graph
): [number, number[]] {
  let dist = 0;
  let graphPath = [end];

  while (end !== start) {
    dist += graph.getWeight(end, parent_node[end]);
    graphPath.unshift(parent_node[end]);
    end = parent_node[end];
  }

  return [dist, graphPath];
}

export default function aStar(
  start: number,
  end: number,
  graph: Graph
```

```
): [number, number[]] {
  const parent_node: number[] = Array(graph.getVertexCount()).fill(-1);
  let pq: PQElement[] = [{ f_score: heur(graph, start, start), node: start }];

  const g_score: number[] = Array(graph.getVertexCount()).fill(-1);
  g_score[start] = 0;

  while (pq.length > 0) {
    let current: number = dequeuePQ(pq) as number;
    if (current === end) return constructPath(start, end, parent_node, graph);

    for (let i = 0; i < graph.getVertexCount(); i++) {
      const edgeWeight: number = graph.getWeight(current, i);
      if (edgeWeight === 0) continue;

      if (g_score[i] === -1 || g_score[current] + edgeWeight < g_score[i]) {
        parent_node[i] = current;
        g_score[i] = g_score[current] + edgeWeight;
        const tentative_f_score = g_score[i] + heur(graph, start, i);

        const pqIdx = findNodeIdxPQ(pq, i);
        if (pqIdx === -1) {
          enqueuePQ(pq, { f_score: tentative_f_score, node: i });
        } else if (tentative_f_score < pq[pqIdx].f_score) {
          pq.splice(pqIdx, 1);
          enqueuePQ(pq, { f_score: tentative_f_score, node: i });
        }
      }
    }
  }

  throw Error("Path to destination not found!");
}
```

UniformCostSearch.tsx

```ts
import Graph from "../classes/Graph";

export default function uniformCostSearch(
  start: number,
  end: number,
  graph: Graph
): [number, number[]] {
  type queueEl = {
    cost: number;
    vertex: number;
    path: number[];
  };

  const prioQueue: queueEl[] = [];
  const visited: boolean[] = new Array(graph.getVertexCount()).fill(false);
  prioQueue.push({ cost: 0, vertex: start, path: [start] });
  while (prioQueue.length > 0) {
    prioQueue.sort((a, b) => b.cost - a.cost);

    const curr: queueEl = prioQueue[prioQueue.length - 1];
    prioQueue.pop();

    if (curr.vertex === end) {
      return [curr.cost, curr.path];
    }

    if (!visited[curr.vertex]) {
      for (let i = 0; i < graph.getVertexCount(); i++) {
        const edgeWeight: number = graph.getWeight(curr.vertex, i);
        if (edgeWeight !== 0) {
          prioQueue.push({
            cost: curr.cost + edgeWeight,
            vertex: i,
            path: curr.path.concat([i]),
          });
        }
      }
    }
```

```
    visited[curr.vertex] = true;
  }

  throw Error("Path to destination not found!");
}
```

Utility.tsx

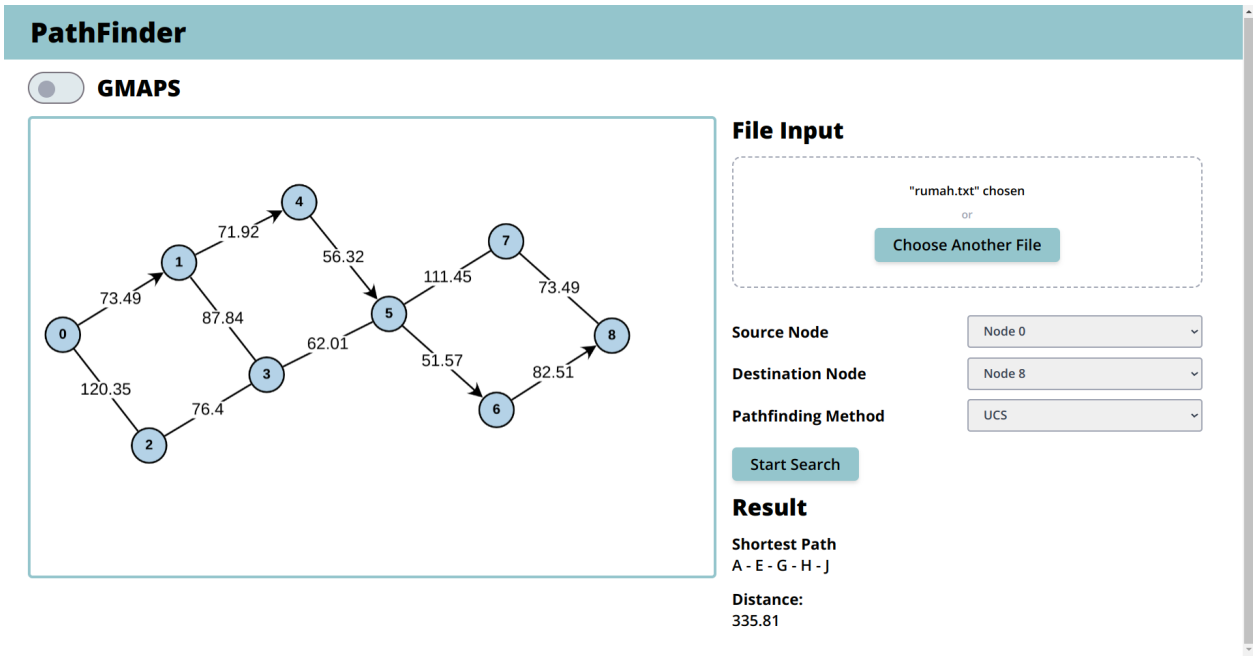
```
import Graph, { coordinate, mapEdge } from "../classes/Graph";

function fileReader(fileData: string): Graph {
  const inputFile: string[] = fileData.split("\n");

  if (isNaN(parseInt(inputFile[0]))) throw Error("Invalid first line!");

  const graph: Graph = new Graph(parseInt(inputFile[0]));

  for (let i = 1; i <= graph.getVertexCount(); i++) {
    const temp: string[] = inputFile[i]
      .slice(1, inputFile[i].length - 1)
      .split(",");
    if (
      temp.length !== 2 ||
      isNaN(parseFloat(temp[0])) ||
      isNaN(parseFloat(temp[1]))
    )
      throw Error("Invalid coordinates!");

    graph.addVertex(i - 1, parseFloat(temp[0]), parseFloat(temp[1]));
  }

  for (let i = graph.getVertexCount() + 2; i < inputFile.length; i++) {
    const temp: string[] = inputFile[i].split(" ");

    if (
```

```
      temp.length !== 4 ||
      isNaN(parseFloat(temp[0])) ||
      isNaN(parseFloat(temp[1])) ||
      isNaN(parseFloat(temp[2]))
    )
      throw Error("Invalid street!");

    graph.addEdge(parseFloat(temp[0]), parseFloat(temp[1]),
parseFloat(temp[2]), temp[3]);
  }

  return graph;
}

function graphFromList(nodeList: coordinate[], edgeList: mapEdge[]): Graph {
  const graph: Graph = new Graph(nodeList.length);
  for (let i = 0; i < nodeList.length; i++) {
    graph.addVertex(i, nodeList[i].long, nodeList[i].lat);
  }
  for (let i = 0; i < edgeList.length; i++) {
    const curr: mapEdge = edgeList[i];
    graph.addEdge(
      curr.source,
      curr.dest,
      euclideanDistance(curr.sourceCoord, curr.destCoord),
      String(curr.source) + String(curr.dest)
    );
  }
  return graph;
}

function euclideanDistance(c1: coordinate, c2: coordinate) {
  return Math.sqrt((c2.lat - c1.lat) ** 2 + (c2.long - c1.long) ** 2);
}

type PQElement = {
  f_score: number;
  node: number;
```

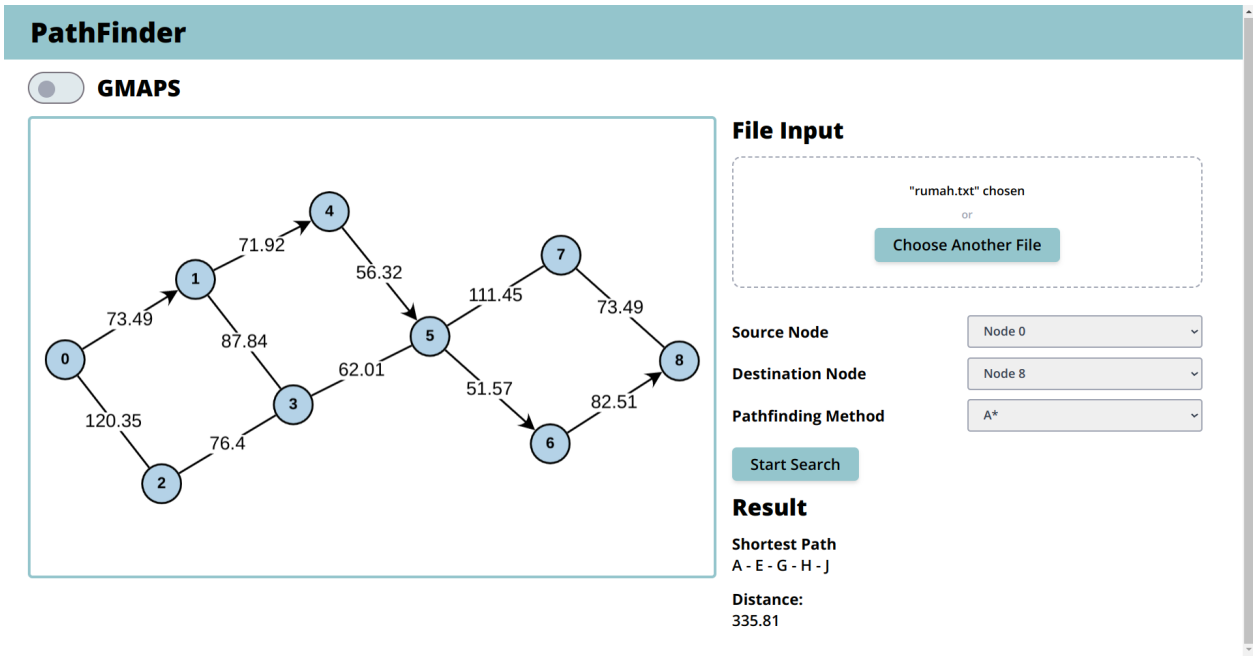
```
};

function enqueuePQ(pq: PQElement[], x: PQElement): void {
  let idx = 0;
  while (idx < pq.length && pq[idx].f_score < x.f_score) idx++;
  pq.splice(idx, 0, x);
}

function dequeuePQ(pq: PQElement[]): Object {
  const x = pq[0].node;
  pq.splice(0, 1);
  return x;
}

function findNodeIdxPQ(pq: PQElement[], node: number): number {
  for (let idx = 0; idx < pq.length; idx++) {
    if (pq[idx].node === node) return idx;
  }
  return -1;
}

export {
  fileReader,
  euclideanDistance,
  enqueuePQ,
  dequeuePQ,
  findNodeIdxPQ,
  type PQElement,
  graphFromList,
};
```

V. Masukan dan Keluaran Program

A. Lintasan terpendek dengan jumlah simpul sebanyak n = 9

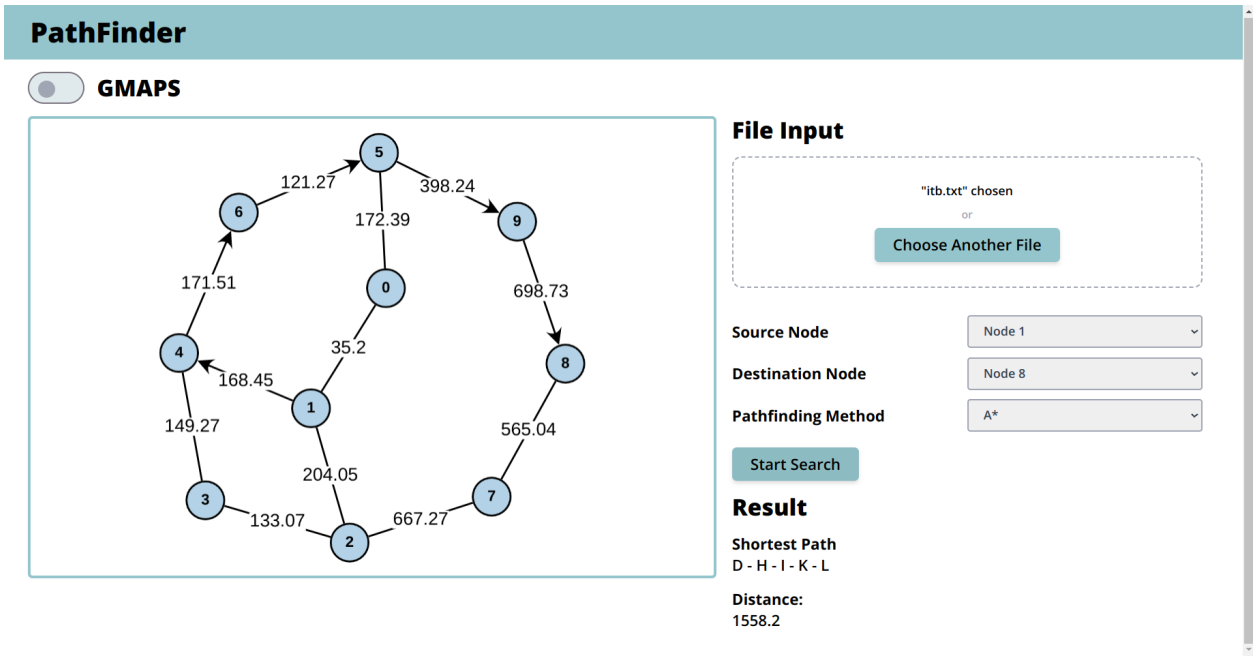


Algoritma UCS dengan 9 simpul

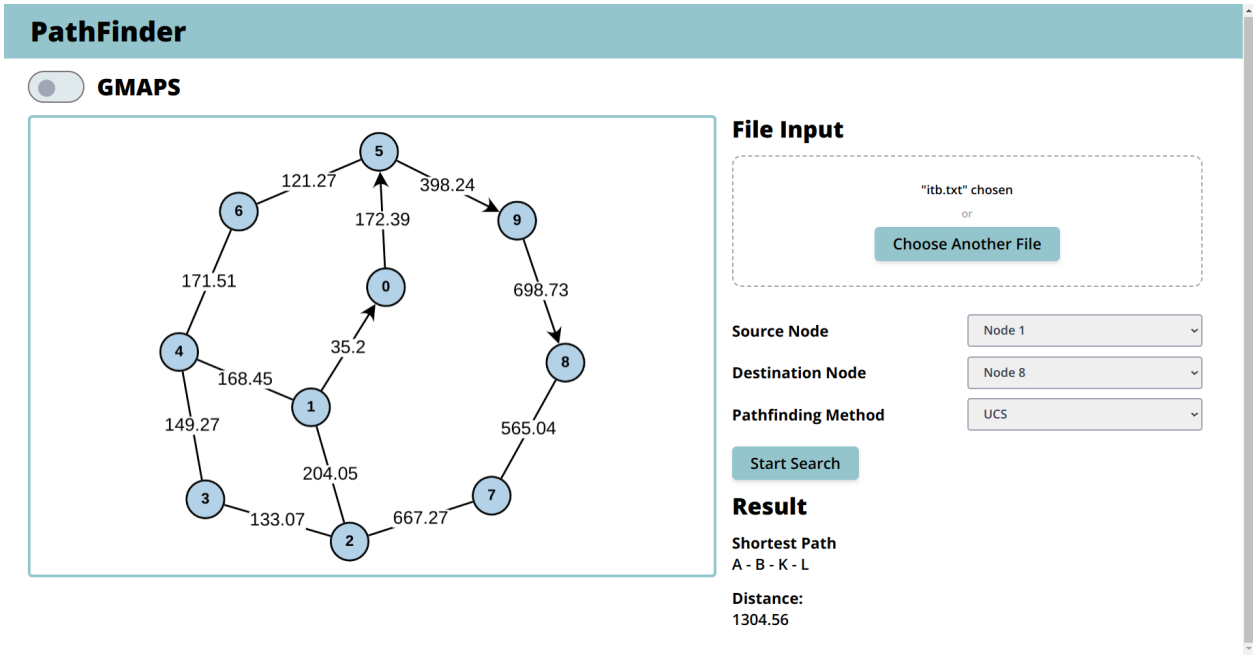


Algoritma A* dengan 9 simpul

B. Lintasan terpendek dengan jumlah simpul sebanyak n = 10

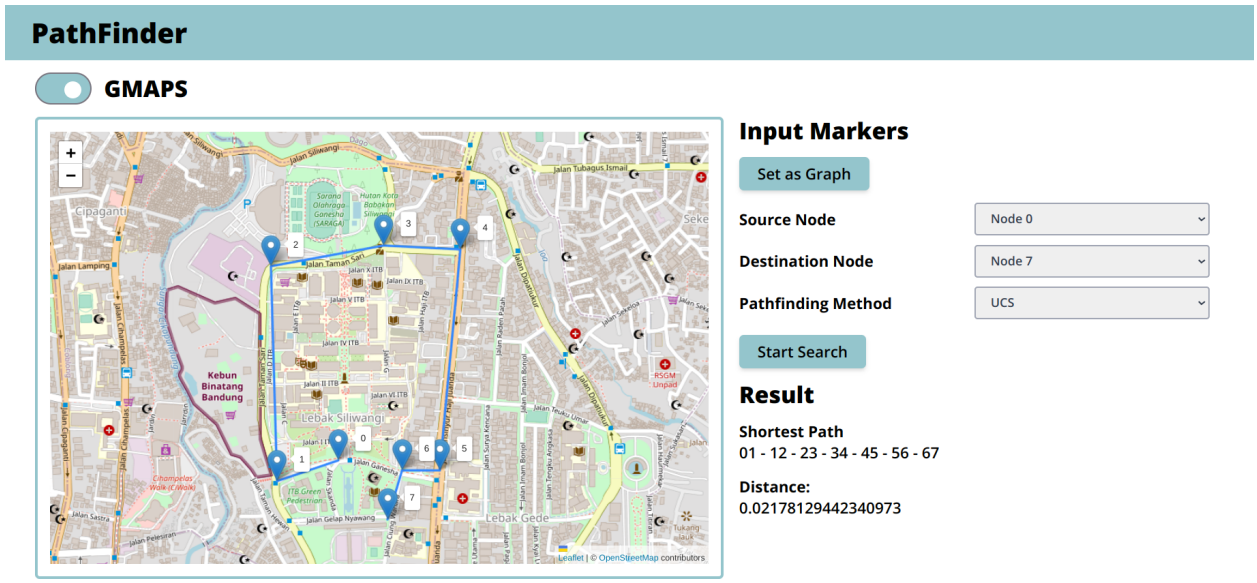


Algoritma A* dengan 10 simpul

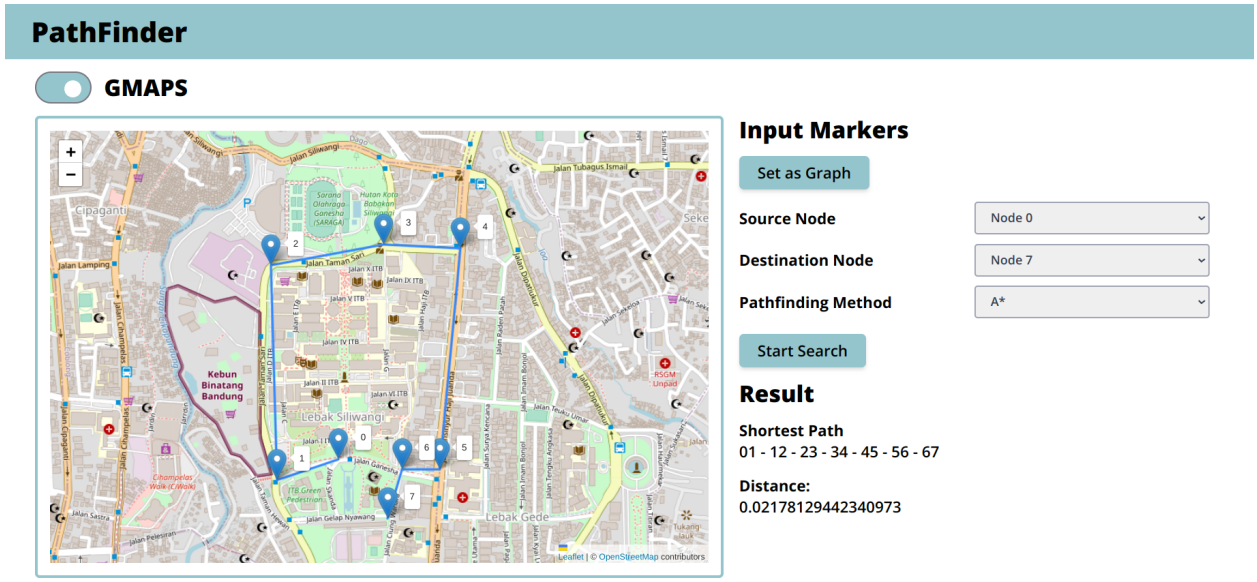


Algoritma UCS dengan 10 simpul

C. Lintasan terpendek dengan maps



Algoritma UCS untuk maps



Algoritma A* untuk maps

VI. Kesimpulan dan Saran

A. Kesimpulan

Dalam tugas kecil IF2211 Strategi Algoritma ini, kami berhasil membuat aplikasi web yang mengaplikasikan algoritma A* dan UCS dalam menyelesaikan persoalan pencarian lintasan terpendek, serta berhasil mengimplementasikan fitur map menggunakan leaflet. Program web yang kami buat memiliki tampilan yang interaktif dan mudah digunakan, serta berhasil menampilkan rute yang dihasilkan, serta jarak yang dihasilkan lintasan tersebut.

Dari pengerjaan tugas kecil ini, dapat disimpulkan bahwa algoritma A* dan UCS mangkus dalam menyelesaikan persoalan lintasan terpendek dalam graf berbobot. Hal tersebut dapat terjadi karena heuristik A* yang digunakan, yaitu euclidean distance simpul dengan simpul awal, bersifat admissible.

B. Saran

Beberapa saran yang kami dapatkan dari pengerjaan pengembangan aplikasi ini adalah sebagai berikut,

1. Perlu dilakukan pendalaman dan pemahaman terlebih dahulu terhadap penggunaan bahasa pemrograman serta framework yang digunakan untuk menyelesaikan tugas ini.
2. Perlunya perencanaan terlebih dahulu mengenai struktur dari program yang akan digunakan untuk menyelesaikan persoalan.

VII. Referensi

GeeksforGeeks. (2023). Uniform Cost Search Dijkstra for large Graphs. *GeeksforGeeks*. https://www.geeksforgeeks.org/uniform-cost-search-dijkstra-for-large-graphs/

Maulidevi, N. U. M., S. T, M. Sc. (n.d.). Penentuan Rute (Route/Path Planning) Bagian 1: BFS, DFS, UCS, Greedy Best First Search. *Bahan Kuliah IF2211 Strategi Algoritma*.

Maulidevi, N. U. M., S. T, M. Sc, & Munir, R. M. (n.d.). Penentuan Rute (Route/Path Planning) Bagian 2: Algoritma A*. *Bahan Kuliah IF2211 Strategi Algoritma*.

Searching: Uniform Cost Search. (2017, August 24). School of Computer Science. https://socs.binus.ac.id/2017/08/24/searching-uniform-cost-search/

Wikipedia contributors. (2023a). Shortest path problem. *Wikipedia*. https://en.wikipedia.org/wiki/Shortest_path_problem

Wikipedia contributors. (2023b). A* search algorithm. *Wikipedia*. https://en.wikipedia.org/wiki/A*_search_algorithm

VIII. Lampiran

Pranala Repository Github

https://github.com/RMA1403/tucil3-13521134-13521149

Matriks Ketercapaian

Poin	Ya	Tidak
1. Program dapat menerima input graf	v	
2. Program dapat menghitung lintasan terpendek dengan UCS	v	
3. Program dapat menghitung lintasan terpendek dengan A*	v	
4. Program dapat menampilkan lintasan terpendek serta jaraknya	v	
5. Bonus: Program dapat menerima input peta dengan Google Map API dan menampilkan peta serta lintasan terpendek pada peta	v	