

ALGORITME *UNIFORM COST SEARCH* DAN A* DALAM BAHASA PEMROGRAMAN PYTHON UNTUK MENENTUKAN LINTASAN TERPENDEK

Disusun untuk memenuhi laporan tugas mata kuliah IF2211
Strategi Algoritma semester 4 di Institut Teknologi Bandung.



Disusun oleh:

Jimly Firdaus (13521102)

Ryan Samuel Chandra (13521140)

TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

Jl. Ganesa No. 10, Lb. Siliwangi, Kecamatan Coblong,
Kota Bandung, Jawa Barat, 40132

2023

PRAKATA

Manusia tidak pernah bisa melakukan segala sesuatu, tanpa adanya bimbingan serta rahmat dari Tuhan Yang Mahakuasa. Maka dari itu, pertama-tama, marilah kita senantiasa panjatkan ucapan syukur dengan sepenuh hati ke hadirat-Nya. Setelah seminggu penuh melakukan eksplorasi dan kerja sama pemrograman, kami akhirnya dapat menuntaskan dan mempersembahkan hasil kerja kami dengan bangga.

Kami mengucapkan terima kasih kepada Ibu Dr. Nur Ulfa Maulidevi, S.T, M.Sc., serta para asisten mata kuliah IF2211 Strategi Algoritma, yang telah berbagi ilmu dan memberikan kesempatan kepada kami untuk menggali dunia informatika lebih dalam lagi melalui tugas ini.

Berikut tersaji dokumen berisi laporan lengkap dengan judul “Algoritme *Uniform Cost Search* dan A* dalam Bahasa Pemrograman Python untuk Menentukan Lintasan Terpendek”. Laporan ini selain dibuat agar membawa manfaat bagi masyarakat, juga secara khusus disusun untuk memenuhi salah satu tugas besar mata kuliah IF2211 Strategi Algoritma di semester 4 Teknik Informatika Institut Teknologi Bandung.

Dengan semangat pantang menyerah dalam menyusun potongan-potongan informasi yang didapat, pun beberapa kali pertemuan rutin untuk membahas koherensi program, kami berusaha sebaik mungkin untuk memenuhi tujuan pembuatan yang telah ditetapkan meskipun mendapatkan tantangan dari waktu dan lingkungan.

Kami berharap percobaan kami ini dapat menjadi sesuatu yang memuaskan bagi semua pembaca. Tetapi, kami pun menyadari bahwa masih banyak kekurangan yang bisa diperbaiki. Maka dari itu, kami mohon kritik dan saran yang membangun untuk perkembangan percobaan kami. Terima kasih, selamat membaca.

Bandung, 12 April 2023

Penyusun Laporan

Daftar Isi

| | |
|---|-----|
| Prakata | ii |
| Daftar Isi | iii |
| BAB 1 : DESKRIPSI MASALAH | 1 |
| 1.1 Latar Belakang | 1 |
| 1.2 Spesifikasi | 1 |
| BAB 2 : TINJAUAN PUSTAKA | 3 |
| 2.1 Google Maps | 3 |
| 2.2 <i>Shortest-Path Problem</i> (Permasalahan Rute-Terpendek)..... | 3 |
| 2.3 Graf Berbobot (<i>Weighted Graph</i>) | 4 |
| 2.4 Algoritme UCS | 5 |
| 2.5 Algoritme A* | 6 |
| 2.6 Formula <i>Haversine</i> | 8 |
| 2.7 Bahasa Pemrograman Python | 8 |
| BAB 3 : IMPLEMENTASI | 10 |
| 3.1 Format Fail Masukan | 10 |
| 3.2 Pendekatan untuk Algoritme UCS | 11 |
| 3.3 Pendekatan untuk Algoritme A* | 12 |
| BAB 4 : DOKUMENTASI EKSPERIMEN | 15 |
| BAB 5 : PENUTUP | 19 |
| 5.1 Kesimpulan | 19 |
| 5.2 Saran | 19 |
| 5.3 Refleksi | 20 |
| 5.4 Tanggapan | 20 |
| LAMPIRAN | 21 |
| DAFTAR REFERENSI | 22 |

BAB 1

DESKRIPSI MASALAH

1.1 Latar Belakang

Algoritme UCS (*Uniform Cost Search*) dan A* (atau *A star*) dapat digunakan untuk menentukan lintasan terpendek dari suatu titik ke titik lain. Pada tugas kecil ketiga ini, mahasiswa 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. Jalan diasumsikan dapat dilalui dari dua arah. Bobot graf menyatakan jarak (dalam meter atau kilometer) antarsimpul. Jarak antara dua simpul dapat dihitung dari koordinat kedua simpul menggunakan rumus jarak Euclidean (berdasarkan koordinat) atau dapat menggunakan ruler di Google Map, atau cara lainnya yang disediakan oleh Google Map.



Gambar 1.1.1 Contoh Peta Daerah Bandung dari Google Maps

Langkah pertama dalam membangun program ini adalah membuat graf yang merepresentasikan peta (di area tertentu, misalnya di sekitar Bandung Utara / Dago). Berdasarkan graf yang dibentuk, program menerima input simpul asal dan simpul tujuan, lalu menentukan lintasan terpendek antara keduanya menggunakan algoritme UCS dan A*. Lintasan terpendek dapat ditampilkan pada peta/graf (misalnya jalan-jalan yang menyatakan lintasan terpendek diberi warna merah). Nilai heuristik yang dipakai adalah jarak garis lurus dari suatu titik ke tujuan.

1.2 Spesifikasi

Berikut adalah spesifikasi program yang dipenuhi:

1. Program menerima input *file* graf (direpresentasikan sebagai matriks ketetanggaan berbobot), 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* saja.

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

Berkas yang dikumpulkan adalah laporan format PDF yang berisi:

1. Deskripsi persoalan.
2. Kode program.
3. Peta/graf input, output *screenshot* peta yang memperlihatkan lintasan terpendek untuk sepasang simpul, tampilkan hasil untuk beberapa lintasan terpendek di kota Bandung atau kota lainnya yang kalian suka.
4. Alamat github/google drive tempat kode sumber program diletakkan jika perlu dieksekusi oleh asisten.
5. Kesimpulan, komentar, dll.

Peta jalan yang digunakan sebagai kasus uji adalah:

1. Peta jalan sekitar kampus ITB/Dago/Bandung Utara
2. Peta jalan sekitar Alun-alun Bandung
3. Peta jalan sekitar Buahbatu atau Bandung Selatan
4. Peta jalan sebuah kawasan di kota asalmu

(Dikutip dari spesifikasi tugas kecil 3 IF2211 Strategi Algoritma tahun ajaran 2023, pranala: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Tucil3-Stima-2023.pdf>)

BAB 2

TINJAUAN PUSTAKA

2.1 Google Maps

Google Maps adalah layanan web yang memberikan informasi terperinci tentang wilayah geografis dan situs di seluruh dunia. Selain peta jalan konvensional, Google Maps menawarkan tampilan udara dan satelit dari banyak lokasi. Di beberapa kota, Google Maps menawarkan tampilan jalan yang terdiri dari foto yang diambil dari sebuah kendaraan ^[9]. Google Maps juga menawarkan beberapa layanan yang lebih kompleks. Misalnya, fitur perencanaan rute menawarkan petunjuk arah bagi pengemudi, pengendara sepeda motor, pejalan kaki, dan pengguna transportasi umum yang bepergian dari satu lokasi tertentu ke lokasi lainnya.

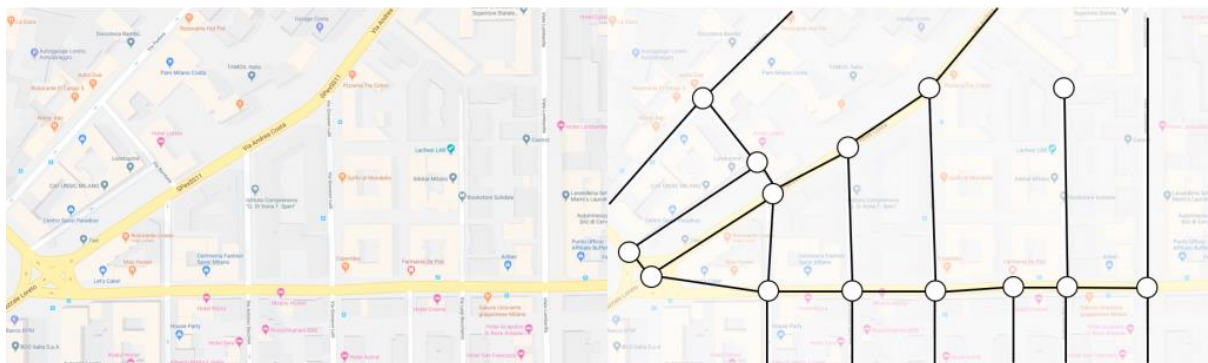
Google Maps *Application Program Interface* (API) memungkinkan perangkat lunak yang berbeda untuk berkomunikasi satu sama lain. Misalnya, administrator sebuah situs web dapat menyematkan Google Maps ke dalam situs yang sudah berwujud, seperti panduan *real estate*, halaman layanan komunitas, dan sebagainya.

Untuk mengukur jarak antara dua titik, mulailah dengan mengeklik kanan di titik awal. Selanjutnya, klik “*measure distance*” pada menu tersebut, lalu klik pada titik tujuan untuk menggambar garis dan mengukur jarak antara kedua lokasi tersebut.

2.2 Shortest-Path Problem (Permasalahan Rute-Terpendek)

Permasalahan seperti menentukan rute terpendek dari suatu tempat ke tempat lain, memilih arah evakuasi sehingga sebanyak mungkin orang bisa melarikan diri dari bencana, dan lainnya, dapat dikategorikan sebagai masalah optimisasi jaringan ^[4]. Selain bidang manajemen lalu lintas, ada pula aplikasi di bidang telekomunikasi, misalnya bagaimana mentransfer data secara efisien (tidak banyak penundaan).

Interkoneksi antarentitas umumnya direpresentasikan sebagai *network diagram* yang dikenal sebagai graf. Graf dibangun oleh *node / vertice* (simpul) yang mewakili entitas, dan *arcs / edges* (sisi), yang mewakili koneksi. Misalnya, pada peta Google Maps, setiap percabangan adalah simpul dan rute penghubung antara dua kota adalah sisi. Dalam beberapa kasus, dapat juga ditambahkan bobot (misalnya untuk merepresentasikan jarak antara dua kota atau waktu yang digunakan untuk berpindah di antara dua tempat).



Gambar 2.1.1 Graf yang Terbentuk dari Peta Google Maps

(Sumber: <https://sistem-komputer-s1.stekom.ac.id/index.php/informasi/baca/TEKNOLOGI-GOOGLE-MAPS-DAN-TEORI-GRAFIK/0efd889600670870277d0d0cd9460e8498f96464>)

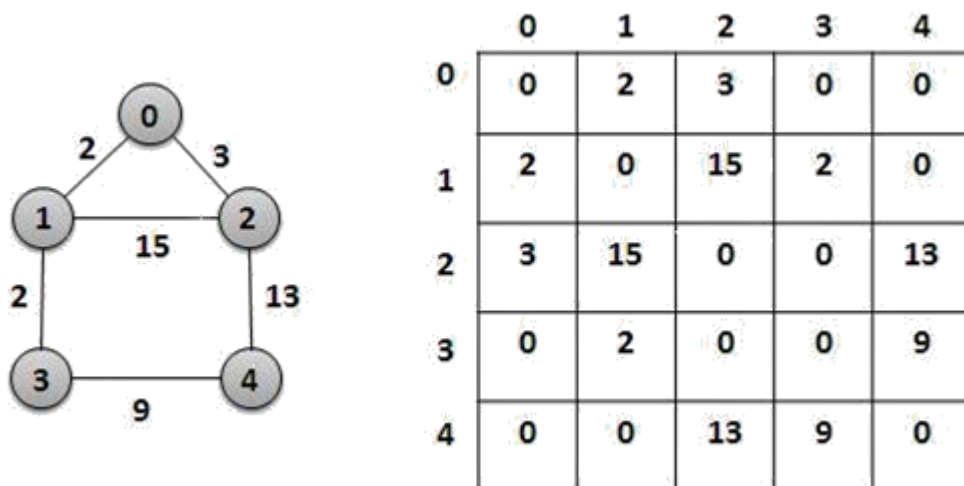
Jadi, *shortest-path problem* merupakan salah satu masalah optimasi jaringan yang bertujuan untuk menentukan jalur terpendek dari satu *node* ke *node* lainnya. Untuk menangani masalah penggunaan graf pada komputer, digunakan *adjacency matrix* (matriks ketetanggaan). Setiap baris dan kolom berhubungan dengan sebuah *node*. “1” ditempatkan di baris ke-*i*; kolom ke-*j* jika ada sisi yang menghubungkan simpul *i* dan simpul *j*. Jika tidak, diisi dengan “0”.

2.3 Graf Berbobot (*Weighted Graph*)

Jika dua buah simpul memiliki sisi di antara mereka (terhubung), maka mereka disebut bertetangga. Ketika hanya dipedulikan apakah dua buah *node* bertetangga atau tidak, graf yang terbentuk disebut tidak-berbobot. Graf seperti itu hanya cocok untuk menjawab pertanyaan seperti: “adakah jalur antara simpul *u* ke simpul *v*?”, “simpul apa saja yang bisa dicapai dari *u*?”, atau “ada berapa simpul yang membentuk jalur terpendek antara *u* dan *v*?” [8].

Bagaimanapun juga, dalam banyak aplikasi, *edge* sering dilengkapi dengan properti numerik yang turut diperhitungkan dalam algoritme untuk menyelesaikan masalah yang sedang dihadapi. Misalnya, seseorang harus mempertimbangkan panjang jalan dan kepadatan lalu lintas saat mencari jalur terpendek antara dua kota. Untuk itu, setiap sisi *e* diasosiasikan dengan nilai *real* $w(e)$ yang disebut bobot. Jadi, graf berbobot adalah graf yang setiap sisinya diberi sebuah harga (bobot) [5].

Cara merepresentasikan graf berbobot adalah perluasan dari representasi graf tak berbobot. Bobot sisi biasanya bernilai positif, jadi nilai nol pada matriks ketetanggaan graf berbobot menunjukkan bahwa tidak ada sisi di antara dua simpul. Namun, pada kasus tertentu, bobot juga bisa diatur menjadi negatif (satu nilai khusus) untuk mengimplisitkan ketidakterhubungan.



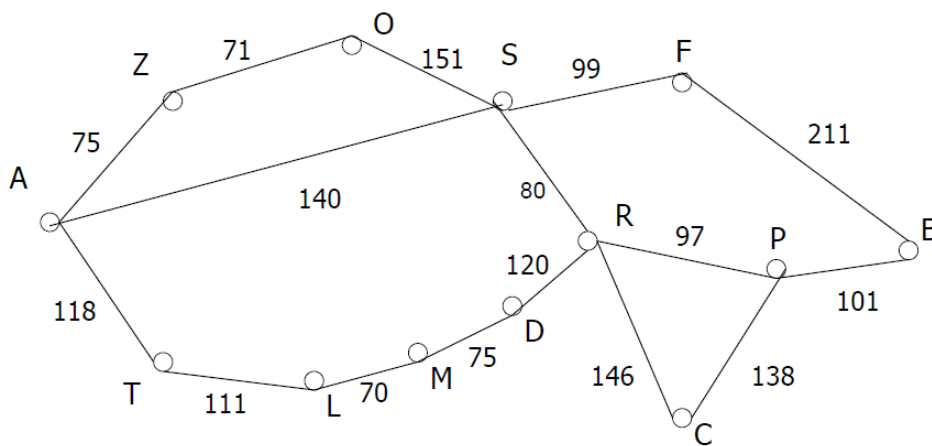
Gambar 2.2.1 Contoh Graf Berbobot dan Matriks Ketetanggaannya

(Sumber: <https://www.thecrazyprogrammer.com/2014/03/representation-of-graphs-adjacency-matrix-and-adjacency-list.html>)

2.4 Algoritme UCS (*Uniform Cost Search*)

Traversal graf adalah cara mengunjungi satu per satu simpul dalam graf. Sama seperti BFS (*Breadth-First Search*), DFS (*Depth-First Search*), DLS (*Depth-Limited Search*), dan IDS (*Iterative-Deepening Search*) yang merupakan algoritme *traversal* graf, UCS dikategorikan sebagai *uninformed search*. *Uninformed search* adalah teknik pencarian yang tidak memiliki informasi tambahan dan karenanya disebut juga dengan teknik pencarian buta (*blind search*)^[6].

BFS dan IDS dapat digunakan untuk mencari jalur dengan langkah (jumlah simpul yang dilalui) paling sedikit, sedangkan UCS meminimumkan total *cost*. *Cost* adalah biaya atau waktu yang diperlukan untuk melangkah dari sebuah simpul ke simpul lainnya, dan dapat disetarakan dengan bobot pada graf, seperti yang sudah disebutkan sebelumnya. Maka dari itu, akan dikenal sebuah pemetaan baru, $g(n)$, yang berarti total biaya untuk sampai ke simpul tertentu dari simpul asal. Diberikan contoh sebagai berikut:



Gambar 2.4.1 Contoh Soal *Path Finding*

(Sumber: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf>)

Gambar di atas merupakan peta daerah Romania yang dimodelkan dalam bentuk graf berbobot. Apabila kita ingin menemukan jalur dengan biaya kumulatif terendah dari Arad (A) ke Bucharest (B), berikut adalah penyelesaian secara BFS dan UCS:

| BFS (<i>Queue</i>) | | UCS (<i>Priority Queue</i>) | |
|----------------------|-------------------------|-------------------------------|--|
| Simpul Ekspan | Simpul Hidup | Simpul Ekspan | Simpul Hidup |
| A | ZA,SA,TA | A | ZA-75,TA-118,SA-140 |
| ZA | SA,TA,OAZ | ZA-75 | TA-118, SA-140,OAZ-146 |
| SA | TA,OAZ,OAS,FAS,RAS | TA-118 | SA-140,OAZ-146,LAT-229 |
| TA | OAZ,OAS,FAS,RAS,LAT | SA-140 | OAZ-146,RAS-220,LAT-229,FAS-239,OAS-291 |
| OAZ | OAS,FAS,RAS,LAT | OAZ-146 | RAS-220, LAT-229,FAS-239,OAS-291 |
| OAS | FAS,RAS,LAT | RAS-220 | LAT-229,FAS-239,OAS-291,PASR-317,DASR-340,CASR-366 |
| FAS | RAS,LAT,BASF | LAT-229 | FAS-239,OAS-291,MATL-299,PASR-317,DASR-340,CASR-366 |
| RAS | LAT,BASF,DASR,CASR,PASR | FAS-239 | OAS-291,MATL-299,PASR-317,DASR-340,CASR-366,BASF-450 |

| | |
|------|--------------------------------------|
| LAT | BASF,DASR,CASR, PASR, MATL |
| BASF | Solusi ditemukan |

| Hasil Akhir: | | |
|--------------|---------|---------------|
| | BFS | UCS |
| Path | A→S→F→B | A→S→ R→P→B |
| Cost | 450 | 418 |

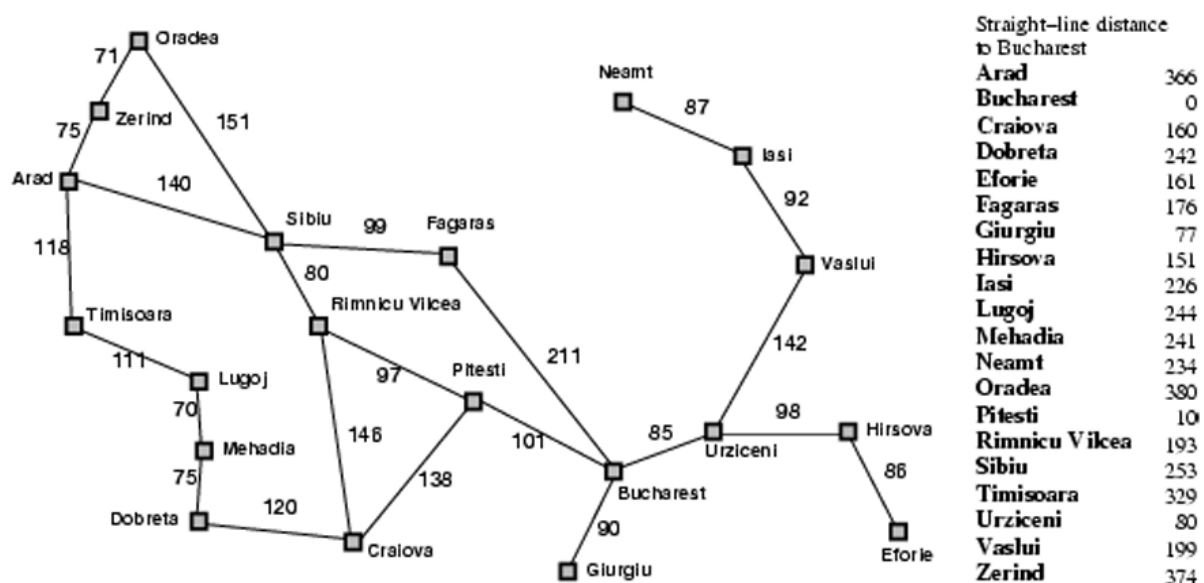
| | |
|-----------|--|
| OAS-291 | MATL-299,PASR-317,DASR-340,CASR-366,BASF-450 |
| MATL-299 | PASR-317,DASR-340, DATLM-364 ,CASR-366,BASF-450 |
| PASR-317 | DASR-340,DATLM-364,CASR-366, BASRP-418 , CASRP-455 , BASF-450 |
| DASR-340 | DATLM-364,CASR-366,BASRP-418,CASRP-455, BASF-450 |
| DATLM-364 | CASR-366,BASRP-418,CASRP-455, BASF-450 |
| CASR-366 | BASRP-418,CASRP-455, BASF-450 |
| BASRP-418 | Solusi ditemukan |

Pada kedua pencarian di atas, terdapat kesamaan, bahwa sebuah simpul yang sudah pernah menjadi simpul ekspansi tidak akan dimasukkan lagi ke dalam antrian. Hal ini bertujuan untuk mencegah pencarian rute menjadi berputar-putar karena harus kembali lagi ke jalur yang sebelumnya sudah dikunjungi.

2.5 Algoritme A*

Algoritme A* bekerja dengan cara menggabungkan pendekatan *greedy best-first search* dan Dijkstra untuk mencari jalur terpendek dari simpul awal ke simpul tujuan. Algoritme ini menggunakan fungsi heuristik untuk memperkirakan jarak dari simpul saat ini ke simpul tujuan. Menurut KBBI, heuristik berarti berkaitan dengan formulasi yang biasanya spekulatif, berfungsi sebagai panduan dalam penyelidikan atau pemecahan masalah. Fungsi heuristik ini digunakan untuk menentukan simpul mana yang harus dikunjungi selanjutnya.

Algoritme A* memulai pencarian dari simpul awal dan mengevaluasi simpul tetangganya dengan menggunakan fungsi $f(n) = g(n) + h(n)$, di mana $g(n)$ adalah biaya dari simpul awal ke simpul n dan $h(n)$ adalah perkiraan (heuristik) biaya dari simpul n ke simpul tujuan. Simpul dengan nilai $f(n)$ terkecil akan dipilih untuk dikunjungi selanjutnya. Proses ini diulangi hingga simpul tujuan ditemukan atau tidak ada lagi simpul yang bisa dikunjungi (tidak ada solusi).



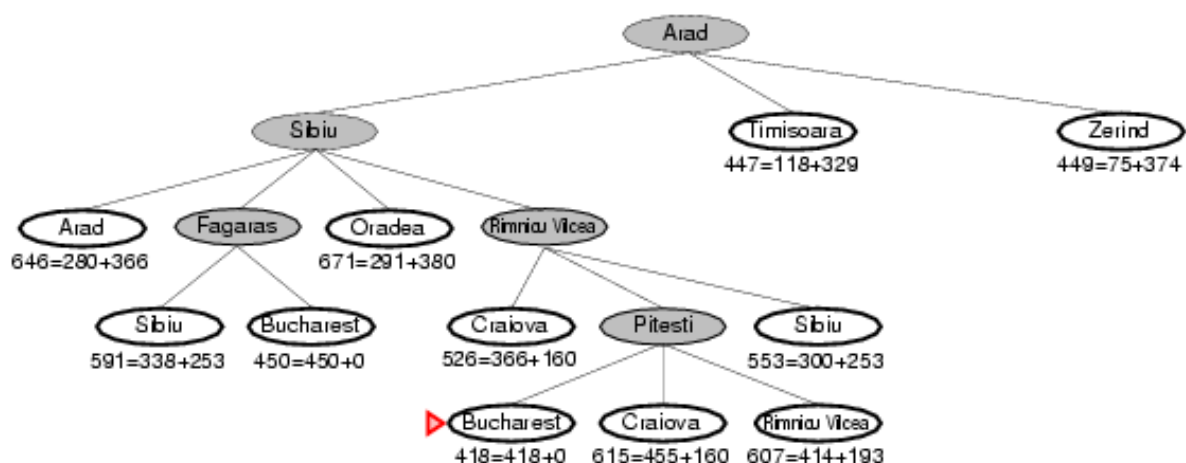
Gambar 2.5.1 Graf Permasalahan Arad-Bucharest dengan Data Heuristik

(Sumber: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf>)

| Penyelesaian Rute Terpendek Arad-Bucharest dengan Algoritme A* | | | |
|--|--------------------|----|--------------------|
| No | Pohon Ruang Status | No | Pohon Ruang Status |
| 1 | | 2 | |
| 3 | | 4 | |
| 5 | | 6 | |

Untuk setiap simpul pada pencarian di atas, diberikan nilai $f(n)$ seperti yang sudah dijelaskan sebelumnya. Simpul berikutnya yang diperiksa adalah simpul dengan $f(n)$ terkecil, sehingga pohon ruang status terbentuk seiring pencarian berlangsung. Maka dari itu, algoritme A* melibatkan graf (pohon) yang bersifat dinamis.

Perlu diperhatikan juga bahwa pada praktiknya, A* sama dengan UCS, yaitu menandai simpul mana saja yang sudah dikunjungi, sehingga tidak akan dikunjungi untuk kedua kalinya. Namun, jika tidak dilakukan (seperti pada gambar), tidak akan menjadi masalah, sebab $f(n)$ dari rute yang mengunjungi sebuah simpul lebih dari satu kali, sudah tentu kalah kecil dengan milik rute lainnya.



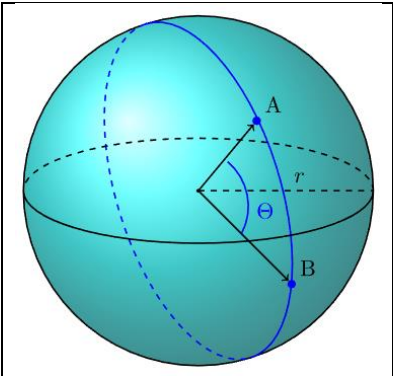
Gambar 2.5.2 Kondisi Akhir Pohon Ruang Status Pencarian Rute Arad-Bucharest dengan A*
(Sumber: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf>)

2.6 Formula *Haversine*

Haversine formula adalah rumus untuk menentukan secara akurat jarak lingkaran besar antara dua titik pada bola. Informasi garis lintang dan garis bujur kedua titik harus diberikan. Ini penting dalam melakukan navigasi; kasus khusus dari rumus yang lebih umum dalam trigonometri bola, sebab hukum *haversine* menghubungkan sisi dan sudut segitiga sebuah bola.

Rumus *haversine* memungkinkan *haversine* dari θ (yaitu, $\text{hav}(\theta)$) untuk dihitung langsung dari garis lintang (diwakili oleh ϕ) dan garis bujur (diwakili oleh λ) dari dua titik: ϕ_1, ϕ_2 adalah garis lintang titik 1 dan garis lintang titik 2, sedangkan λ_1 dan λ_2 adalah garis bujur titik 1 dan garis bujur titik 2.

Fungsi *haversine* menghitung setengah *versine* dari sudut θ , atau rumus yang lebih mudah dipahami: $\text{haversine}(\theta) = \sin^2(\theta/2)$. Untuk mendapatkan jarak d antara dua titik, bisa diterapkan *archaversine* (*haversine* terbalik) ke $h = \text{hav}(\theta)$ atau gunakan fungsi *arcsine* (*sine* terbalik).

| | |
|--|--|
|  | $d_{AB} = r \times \Theta$ |
| | $\text{hav}(\Theta) = \text{hav}(\phi_B - \phi_A) + \cos(\phi_B) \cos(\phi_A) \text{hav}(\lambda_B - \lambda_A)$ |
| | $d_{AB} = r \times \text{archav}(\text{hav}(\Theta))$ |

Gambar 2.6.1 Ilustrasi Penggunaan Rumus *Haversine*
(Sumber: <https://www.baeldung.com/cs/haversine-formula>)

2.7 Bahasa Pemrograman Python

Python adalah bahasa pemrograman komputer, sama layak seperti bahasa pemrograman lain, misalnya C, C++, Pascal, Java, PHP, Perl, dan lain-lain. Sebagai bahasa pemrograman, Python tentu memiliki dialek, kosakata atau kata kunci (*keyword*), dan aturan tersendiri yang jelas berbeda dengan bahasa lainnya ^[10].

Bahasa pemrograman Python disusun pada Desember 1989 oleh Guido Van Rossum di Centrum Wiskunde & Informatica (CWI), sebuah pusat riset di bidang matematika dan sains, Amsterdam-Belanda; sebagai suksesor atau pengganti dari bahasa pemrograman pendahulunya, ABC, yang juga dikembangkan di tempat yang sama ^[10].

Saat ini, Python sudah banyak digunakan oleh perusahaan-perusahaan atau organisasi-organisasi besar untuk mengembangkan sistem yang mereka butuhkan. Mesin pencari Google, YouTube, dan produk-produk Google lainnya seperti Google App Engine adalah contoh sistem yang dikembangkan menggunakan Python, meskipun pada beberapa bagian masih ditulis menggunakan C dan C++. Python juga banyak digunakan untuk mengembangkan aplikasi-aplikasi di berbagai bidang, seperti: pengembangan web, keuangan, permainan komputer (*games*), pemerintahan, sains, edukasi, dan lain sebagainya ^[10].

Secara umum, para *programmer* banyak yang menjatuhkan pilihannya ke bahasa Python karena beberapa alasan keunggulan berikut:

1. Python memiliki konsep desain yang bagus dan sederhana, yang berfokus pada kemudahan dalam penggunaan. Kode Python dirancang untuk mudah dibaca, dipelajari, digunakan ulang, dan dirawat. Selain itu, Python juga mendukung pemrograman berorientasi objek dan pemrograman fungsional ^[10].
2. Python dapat meningkatkan produktivitas dan menghemat waktu bagi para *programmer*. Untuk memperoleh hasil program yang sama, kode Python jauh lebih sedikit dibandingkan dengan kode yang ditulis menggunakan bahasa pemrograman lain seperti C, C++, Java, maupun C# ^[10].
3. Program yang ditulis menggunakan Python dapat dijalankan di hampir semua sistem operasi (Unix, Linux, Windows, MacOS, dll.), termasuk perangkat *mobile* ^[10].
4. Python memiliki banyak dukungan pustaka yang dikembangkan oleh pihak ketiga, misalnya pustaka untuk pengembangan web, pengembangan aplikasi visual (berbasis GUI), pengembangan *game*, dan masih banyak lagi ^[10].
5. Melalui mekanisme tertentu, kode Python dapat diintegrasikan dengan aplikasi yang ditulis dalam bahasa pemrograman lain ^[10].
6. Python bersifat gratis atau bebas (*free*) dan *open-source*, meskipun digunakan untuk kepentingan komersil ^[10].

BAB 3

IMPLEMENTASI

3.1 Format Fail Masukan



3.2 Pendekatan untuk Algoritme UCS

Secara umum, algoritme UCS mungkin perlu didukung oleh adanya struktur data *priority queue* yang diisi dengan tipe data bentukan juga. Isi yang dimaksud misalnya berupa *tuple* yang mengandung larik dari *nodes* yang sudah dilalui, berpasangan dengan sebuah nilai $g(n)$, untuk kemudian diprioritaskan berdasarkan `tuple[1]` terkecil.

Ide lainnya—yang sudah pasti tidak akan berhasil—adalah menggunakan dua buah larik secara bersamaan, yang pertama menampung *path*, dan satunya menampung $g(n)$. Penambahan elemen pada kedua larik selalu dilakukan berbarengan agar *path* pada indeks 0, misalnya, memiliki $g(n)$ pada indeks yang sama. Kemudian, $g(n)$ diurutkan dengan fungsi `sort`. Kegagalan bisa dijamin karena pengurutan pada larik $g(n)$ membuat sinkronisasi kedua larik hancur lebur: pasangan-pasangan tidak berada pada indeks yang sama.

Terlepas dari semua itu, algoritme UCS pada program kali ini dibuat menggunakan pendekatan yang cukup unik. Alih-alih mengimplementasikan struktur data yang rumit, Python membuat pekerjaan menjadi mudah, yaitu cukup dengan sebuah *dictionary* untuk dipakai di keseluruhan program UCS. Selain itu, dibuat juga sebuah *list boolean* yang diinisialisasi *false* sejumlah simpul, dan `list[idx]` akan bernilai *true* apabila simpul ke-*idx* sudah dikunjungi.

Key dari *dictionary* adalah *path* dalam bentuk angka (bersesuaian dengan indeks pada matriks ketetanggaan) yang dikonversi menjadi *string* dan dipisah-pisah dengan menggunakan tanda “-”. Sebagai contoh, “1-3-5-2”, yang kemudian memiliki *value* berupa jarak dari *starting node* (dalam kasus ini simpul 1), sampai *node* saat ini (simpul 2).

<repeat> Sebelum pemeriksaan simpul ekspansi dimulai, *dictionary* akan disortir secara menaik berdasarkan *value*-nya menggunakan sintaksis khusus bawaan Python. Maka dari itu, simpul ekspansi akan selalu terletak di angka terakhir pada pasangan *key-value* pertama. Untuk mendapatkannya, digunakan fungsi `split` dengan *delimiter* “-”. Jika “1-3-5-2” adalah *path* yang sejauh ini memiliki $\sum g(n)$ terendah, maka ia diambil dan diubah menjadi larik `['1', '3', '5', '2']`. Simpul ‘2’ yang menjadi simpul ekspansi dikonversi dari tipe *string* menjadi *integer* (bukan hal sulit dalam Python).

`list[2]` diubah menjadi *true*, kemudian matriks ketetanggaan dicek untuk mengetahui simpul mana saja yang bertetangga dengan simpul 2. Apabila `matrix[2][i]`, dengan *i* sembarang angka; $0 \leq i < \text{jumlah simpul}$, bernilai 1, dan `list[i]` bernilai *false*, dilakukan penambahan pasangan ke *dictionary*. Katakanlah $i = 4$, maka *dictionary* ditambahkan *key* “1-3-5-2-4” dengan *value* berupa: *value* dari “1-3-5-2” ditambah perhitungan *haversine* simpul 2 ke simpul 4.

Jika simpul 2 adalah *goal node*, maka pencarian dihentikan; keluar dari fungsi dengan *me-return* “1-3-5-2” beserta *value*-nya. Tetapi kalau bukan, “1-3-5-2” dihapus dari *dictionary*, kemudian proses kembali ke bagian bertanda *<repeat>* dengan isi *dictionary* dan *list* yang sudah berbeda dari sebelumnya.

Penjelasan lebih lanjut dapat diamati melalui contoh sederhana. Misalkan seseorang ingin mencari lintasan terpendek dari simpul 0 ke simpul 4, dengan simpul 0 bertetangga dengan 1 dan 3, simpul 3 bertetangga dengan simpul 4, simpul 4 bertetangga dengan simpul 5, maka *dictionary* kurang lebih secara bertahap akan berisi hal-hal berikut: $\{ '0' : 0 \} \rightarrow \{ '0-1' : 70.26, '0-3' : 85.34 \} \rightarrow \{ '0-3' : 85.34 \} \rightarrow \{ '0-3-4' : 102.09 \} \rightarrow \{ '0-3-4-5' : 143.5 \}$

Berikut adalah implementasi kode dalam bahasa Python:

```
#DEKLARASI LIBRARY
from src.algorithm.util.Utility import *

#DEKLARASI DAN IMPLEMENTASI FUNGSI
def UCS(adjacency_matrix, total_nodes, nodes_data, start_node, _goal_node):
    #Fungsi untuk ...
    #KAMUS LOKAL
    # ...
    #ALGORITME
    visited = []
    for i in range(total_nodes):
        visited.append(False)

    # search start node data
    starting_node = 0
    for name, pos in nodes_data:
        if (name == str(start_node)):
            break
        starting_node += 1

    # search goal node data
    finish_node = 0
    for name, pos in nodes_data:
        if (name == str(_goal_node)):
            break
        finish_node += 1

    simpul_hidup = {str(starting_node) : 0}
    while (bool(simpul_hidup) == True):
        sorted_simpul_hidup = sorted(simpul_hidup.items(), key=lambda x:x[1])
        sorted_simpul_hidup = dict(sorted_simpul_hidup)
        check = next(iter(sorted_simpul_hidup))
        list_check = check.split("-")
        simpul_ekspan = int(list_check[len(list_check)-1])
        visited[simpul_ekspan] = True

        for i in range (len(adjacency_matrix[simpul_ekspan])):
            if (adjacency_matrix[simpul_ekspan][i] == 1):
                if (visited[i] == False):
                    distance_now = d_haversine(nodes_data[simpul_ekspan][1][0], nodes_data[simpul_ekspan][1][1], nodes_data[i][1][0], nodes_data[i][1][1])
                    simpul_hidup[check+"-"+str(i)] = simpul_hidup[check] + distance_now

        if (visited[finish_node] == True):
            return (check, sorted_simpul_hidup[check])
        else:
            del simpul_hidup[check]
```

Gambar 3.2.1 Implementasi Algoritme UCS dalam Python

(Sumber: dokumentasi penulis)

3.3 Pendekatan untuk Algoritme A*

Algoritme A* adalah algoritme pencarian jalur yang efisien untuk menemukan jalur terpendek antara *starting node* dan *goal node*. Algoritme ini menggunakan pendekatan heuristik untuk memperkirakan jarak dari *node* saat ini ke *goal node*, yang memungkinkan algoritme untuk mengeksplorasi jalur yang paling mungkin menuju *goal node* terlebih dahulu.

Dalam algoritme A*, *starting node* dan *goal node* ditentukan terlebih dahulu. *Starting node* kemudian di-enqueue ke dalam *priority queue* A*. Dalam sebuah loop, *node-node* dalam *priority queue* di-dequeue satu per satu hingga *node* yang dicek saat ini merupakan *goal node*. Setiap *node* baru yang ditemukan akan disimpan data rute dari *root* hingga ke *node* tersebut dan juga disimpan biaya yang dibutuhkan dari *root node* untuk mencapai *node* tersebut. Prioritas pada *priority queue* didasarkan pada total biaya dari *root node* hingga ke *node* saat ini ditambah dengan biaya minimum dari *node* saat ini menuju *goal node* (heuristik). Proses ini diulang hingga *goal node* tercapai. Karena setiap loop akan memproses *node* dengan biaya minimum, maka saat *goal node* tercapai, solusi sudah optimal.

Berikut adalah implementasi kode dalam bahasa Python:

```
class AStarNode:
    def __init__(self, path_to_current_node, total_cost_to_this_node, node_pos, current_node_number, reliased_cost):
        self.path_to_node = path_to_current_node
        self.node_total_cost = total_cost_to_this_node
        self.latitude = node_pos[0]
        self.longitude = node_pos[1]
        self.current_node_number = current_node_number
        self.reliased_cost = reliased_cost

    # heuristic here
    def __lt__(self, other):
        return self.node_total_cost < other.node_total_cost

    def get_history_route(self):
        return self.path_to_node

    def get_cost_to_this_node(self):
        return self.reliased_cost

    def get_node_position(self):
        return (self.latitude, self.longitude)

    def get_node_number(self):
        return self.current_node_number
```

```
from src.algorithm.util.AStarUtil import AStarNode
from src.algorithm.util.Utility import d_haversine
import heapq

class AStar(IAStar):
    def __init__(self, adjacency_matrix, total_nodes, nodes_data):
        self.adjacency_matrix = adjacency_matrix
        self.total_nodes = total_nodes
        self.nodes = nodes_data

    def solve(self, start_node, _goal_node):
        # bound index
        max_index = len(self.adjacency_matrix)

        root_position = {
            "latitude": 0,
            "longitude": 0
        }
        goal_position = {
            "latitude": 0,
            "longitude": 0
        }

        # search starting node data
        starting_node = 0
        for name, pos in self.nodes:
            if (name == str(start_node)):
                root_position["latitude"] = pos[0]
                root_position["longitude"] = pos[1]
                break
            starting_node += 1

        # search goal node data
        goal_node = 0
        for name, pos in self.nodes:
            if (name == str(_goal_node)):
                goal_position["latitude"] = pos[0]
                goal_position["longitude"] = pos[1]
                break
            goal_node += 1

        # found the starting, goal data, and starting_node for adjacency matrix
        # start searching
        goal_not_found = True
```

```

# initiate starting node
cost_to_goal = d_haversine(root_position["latitude"], root_position["longitude"], goal_position["latitude"], goal_position["longitude"])
start_node_data = AStarNode("", cost_to_goal, (root_position["latitude"], root_position["longitude"]), starting_node, 0)
AStarQueue = []
heapq.heappush(AStarQueue, (cost_to_goal, start_node_data))
current_node = starting_node
while (goal_not_found != False):
    # find all the available path from current node
    # get the cost then push it to Pqueue
    # repeat until found goal
    current_node_data = heapq.heappop(AStarQueue)
    current_node = current_node_data[1].get_node_number()

    # if found goal
    if (current_node == goal_node):
        goal_not_found = False
        self.result_node = current_node_data[1].get_cost_to_this_node()
        self.final_route = current_node_data[1].get_history_route() + "-" + str(current_node)
        break

    for i in range(max_index):
        if (self.adjacency_matrix[current_node][i] == 1):
            path_to_this_node = current_node_data[1].get_history_route() + "-" + str(current_node)
            reliased_cost = d_haversine(self.nodes[i][1][0], self.nodes[i][1][1], self.nodes[current_node][1][0], self.nodes[current_node][1][1]) + current_node_data[1].get_cost_to_this_node()

            # Heuristic for A* algorithm.
            cost_to_goal = d_haversine(self.nodes[i][1][0], self.nodes[i][1][1], goal_position["latitude"], goal_position["longitude"]) + reliased_cost

            # Enqueue new node
            new_node_data = AStarNode(path_to_this_node, cost_to_goal, self.nodes[i][1], i, reliased_cost)
            heapq.heappush(AStarQueue, (cost_to_goal, new_node_data))

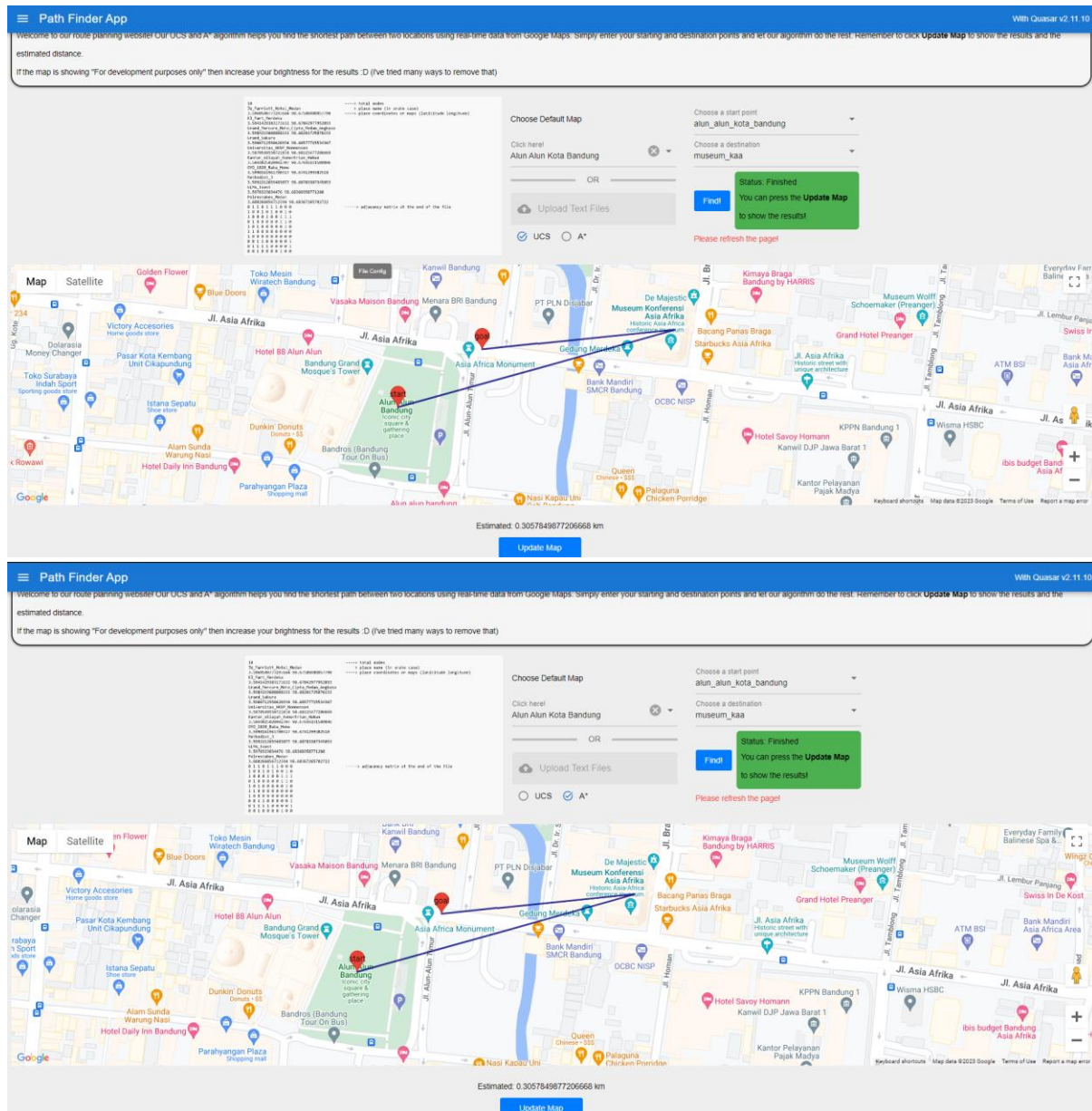
def get_result_route(self):
    result_route = self.final_route[1:]
    return (str(self.result_node), result_route)

```

Gambar 3.3.1 Implementasi Algoritme A* dalam Python
(Sumber: dokumentasi penulis)

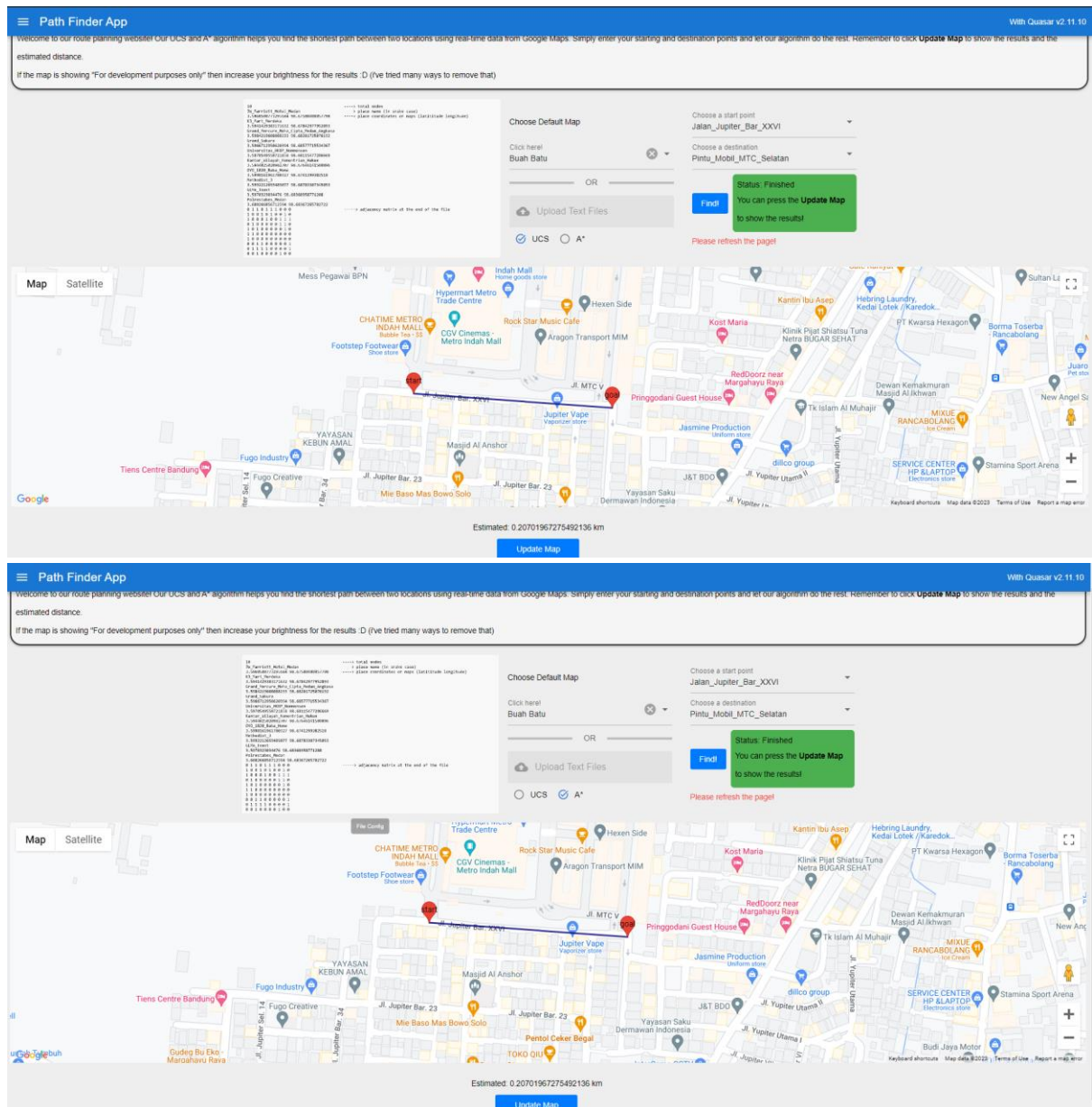
BAB 4

DOKUMENTASI EKSPERIMEN



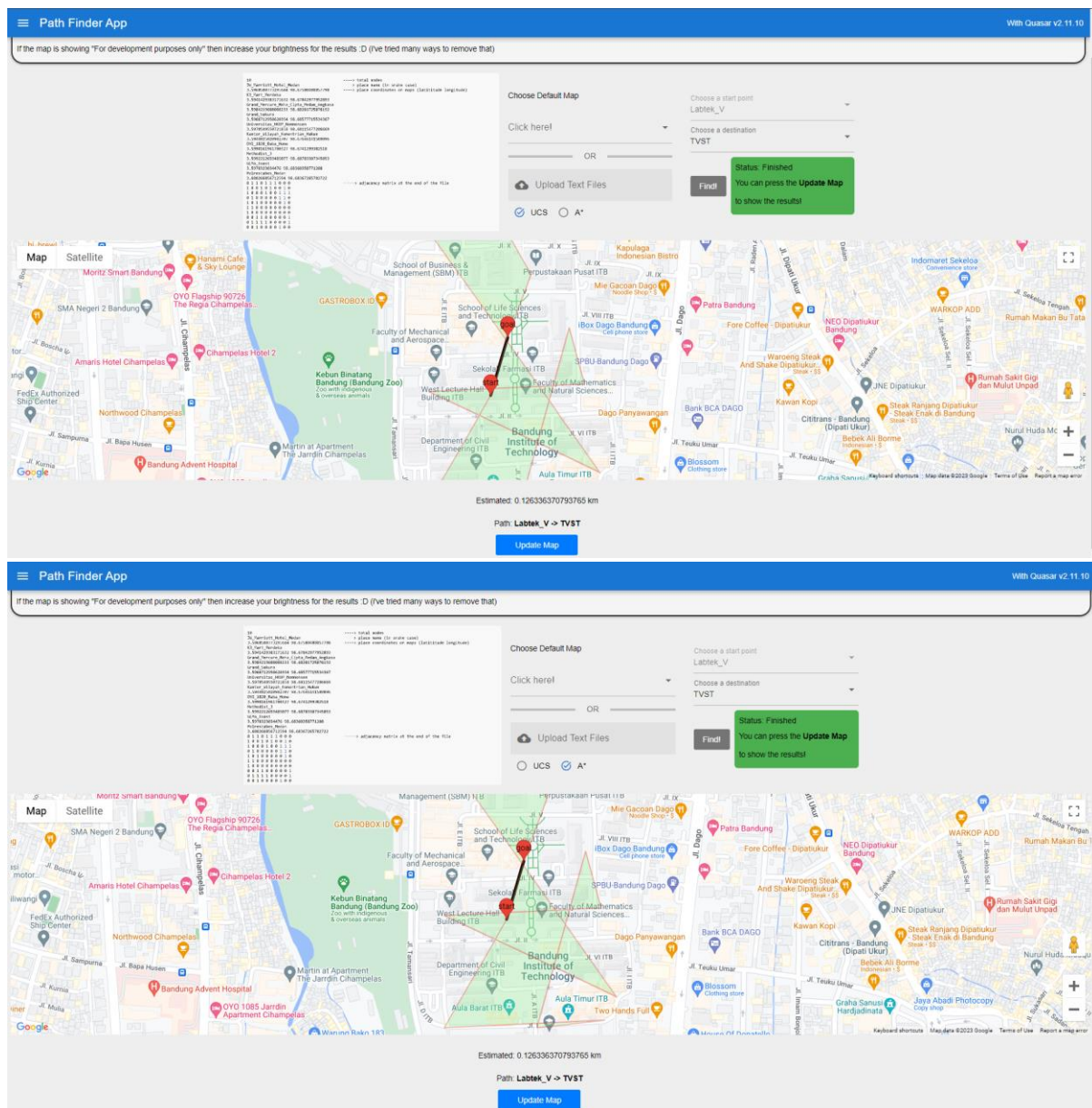
Gambar 4.1 Kasus Uji: dari Alun-Alun Kota Bandung Menuju Museum KAA Menggunakan Algoritme UCS dan A*

(Sumber: dokumentasi penulis)

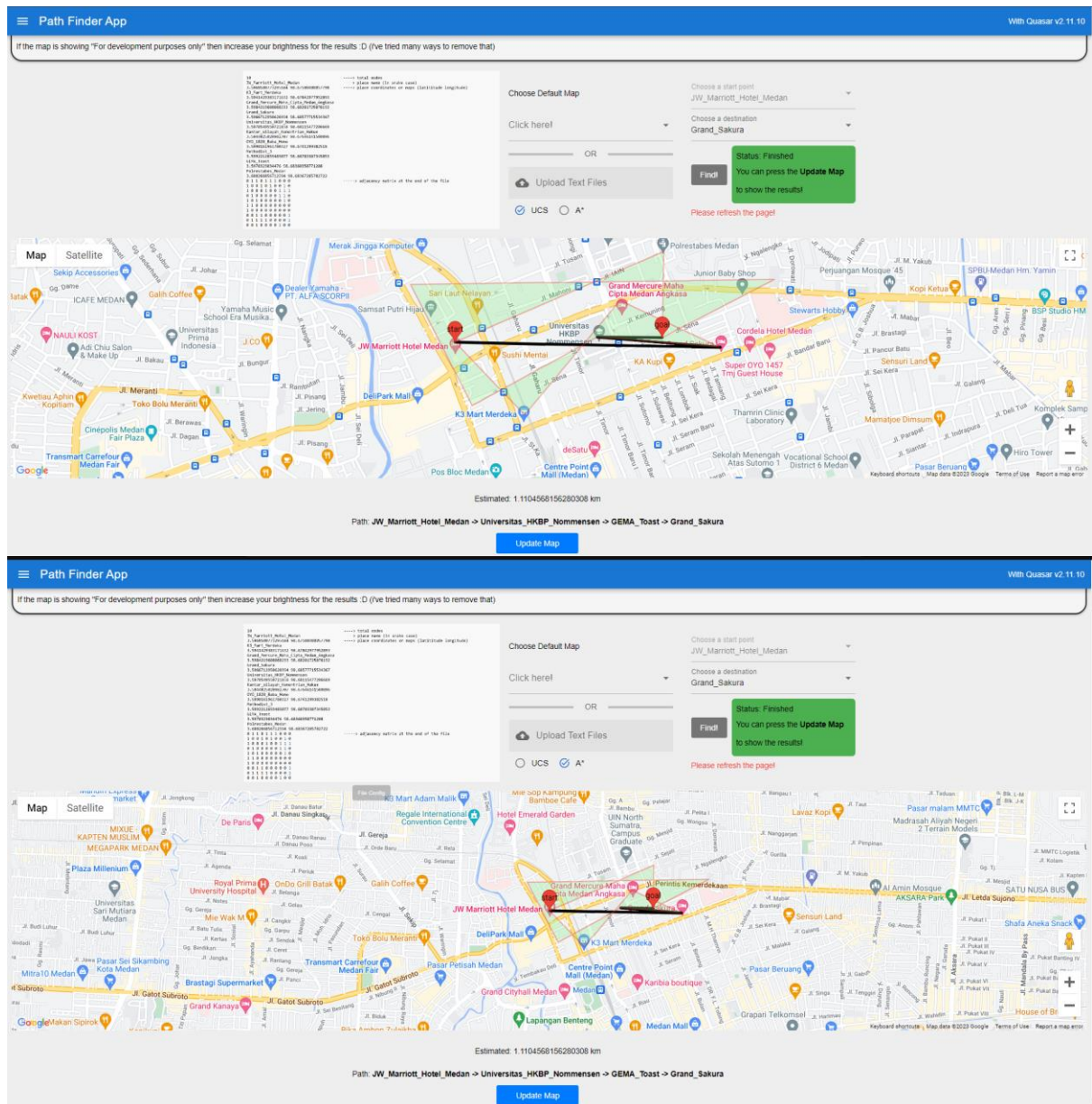


Gambar 4.2 Kasus Uji: dari Jalan Jupiter Bar XXVI Menuju Pintu Mobil MTC Selatan Menggunakan Algoritme UCS dan A*

(Sumber: dokumentasi penulis)



Gambar 4.3 Kasus Uji: dari Labtek V Menuju TVST Menggunakan Algoritme UCS dan A*
(Sumber: dokumentasi penulis)



Gambar 4.4 Kasus Uji: dari JW_Marriott_Hotel_Medan Menuju Grand_Sakura Menggunakan Algoritme UCS dan A*

(Sumber: dokumentasi penulis)

BAB 5

PENUTUP

4.1 Kesimpulan

Dalam kehidupan sehari-hari, manusia sering dihadapkan dengan masalah pencarian rute. Misalnya saat akan berkuliah, seseorang ingin berangkat dari rumah dan sampai di kampus dalam durasi waktu yang sesingkat-singkatnya. Dalam dunia komputasi modern, dikenal Google Maps. Selain memberikan informasi terperinci tentang wilayah geografis dan situs di seluruh dunia, layanan web ini juga memiliki fitur perencanaan rute yang mampu memberikan petunjuk arah bagi pengemudi maupun pejalan kaki.

Di antara sekian banyak algoritme *path / route planning*, ada setidaknya dua algoritme yang selalu memberikan solusi optimal menggunakan representasi graf, yaitu *Uniform Cost Search* (UCS) dan *A**. Dengan kata lain, solusi yang dihasilkan selalu berupa lintasan dengan *cost* (biaya) minimum, entah dari segi jarak yang ditempuh atau waktu untuk mencapai tujuan. Meskipun UCS dikategorikan sebagai *blind / uninformed search*, sedangkan *A** adalah *informed search* yang melibatkan teknik heuristik, keduanya memanfaatkan kemampuan logis dari sebuah struktur data universal, yaitu *priority queue*.

Nyatanya, terdapat variasi pada implementasi kali ini, sehingga tidak menggunakan *priority queue* secara gamblang. Selain itu, matriks ketetanggaan dibuat tidak berbobot, dan bobot (jarak) dapat dihitung menerapkan rumus *haversine* pada koordinat simpul. Bagaimanapun juga, algoritme UCS dan *A** dalam bahasa Python untuk menyelesaikan *shortest-path problem*, dinilai sama-sama efektif. Dengan *file* masukan yang sama, kedua algoritme berhasil menyelesaikan pekerjaan dalam hitungan detik, serta menampilkan solusi yang 100% akurat. Program lengkap dapat dilihat pada pranala di bagian **LAMPIRAN**, sedangkan contoh eksekusi ada di **BAB 4**.

4.2 Saran

1. Sebaiknya, penyelesaian *shortest-path problem* dengan graf, dilakukan dalam bahasa pemrograman yang kaya akan fungsi-fungsi bawaan. Misalnya dalam Python, akses elemen pada suatu senarai, *typecasting*, operasi *string*, serta *sorting* sangatlah mudah.
2. Sebelum fase implementasi, tim pemrogram wajib berdiskusi dan menetapkan format masukan terlebih dahulu. Hal ini bertujuan agar algoritme-algoritme yang dibuat menjadi sinkron (bisa menggunakan data masukan yang sama), serta mempermudah modifikasi program.
3. Membaca spesifikasi dengan teliti sampai tuntas sebaiknya dilakukan sebelum memulai pemrograman. Dengan demikian, tidak perlu ada perubahan massal di tengah pekerjaan karena ketidaksesuaian dengan spesifikasi.
4. Seharusnya, durasi pengerjaan tugas diberikan lebih dari satu minggu, mengingat tugas “bermain” algoritme seperti ini membutuhkan eksplorasi yang sangat banyak, serta mungkin dapat dikembangkan lebih lanjut lagi apabila waktu tersedia.

4.3 Refleksi

Tugas kecil ketiga memberikan banyak sekali ilmu baru dalam pemrograman dengan bahasa Python. Tidak hanya *Command-Line Interface* (CLI), kami jadi berpengalaman dan mengasah keterampilan dalam menggunakan GUI yang dipadukan dengan Google Maps API, dan lain sebagainya. Perlu diingat bahwa, apa pun yang terjadi, solidaritas dan koordinasi antaranggota kelompok merupakan kunci utama terselesaikannya semua pekerjaan dengan baik. Apabila satu orang saja membuat bagian program yang tidak sejalan secara logis, maka keseluruhan program pun menjadi salah.

Dengan terbatasnya pengetahuan serta keterampilan, kami harus berusaha keras untuk mengeksplor jauh lebih dalam daripada materi yang diajarkan di kelas. Internet, bahkan teman dan saudara, telah menjadi sumber untuk meningkatkan pemahaman. Akhirnya, usaha kami tidak sia-sia. Kami dapat menyelesaikan tugas kali ini dengan baik.

4.4 Tanggapan

Katanya, jurusan Teknik Informatika memang kental dengan tugasnya yang melimpah. Perasaan itu memang terus kami warisi dari kakak-kakak tingkat yang sudah lebih dahulu berjuang di posisi kami. Mungkin “keren” atau “seru” adalah sebuah kata yang dapat menggambarkan hasil-hasil dari tugas yang sudah diselesaikan oleh mahasiswa Teknik Informatika. Bagaimana tidak? Melihat aplikasi menarik yang dibangun dari nol hanya dengan perangkat laptop, pasti sangat menyenangkan.

Di balik semua itu, terdapat perjuangan, tawa dan tangis yang tidak terbatas jumlahnya. Hati kami terus menjerit, raga terus berusaha, merindukan masa depan yang cerah-cemerlang, sebagai bayaran atas upaya terbaik kami. Akhir kata, kami merasa haru bercampur sedih dalam menjalani seluruh proses tugas ini. Meskipun perjalanan tidak mudah, kami dapat selangkah lebih maju menuju tercapainya mimpi dan cita-cita. Amin.

LAMPIRAN

Pranala *Repository*:

https://github.com/Jimly-Firdaus/Tucil3_13521102_13521140

Tabel A Runut Pengerjaan Tugas

| POIN | YA | TIDAK |
|--|-------------------------------------|-------|
| 1. Program dapat menerima input graf. | <input checked="" type="checkbox"/> | |
| 2. Program dapat menghitung lintasan terpendek dengan UCS. | <input checked="" type="checkbox"/> | |
| 3. Program dapat menghitung lintasan terpendek dengan A*. | <input checked="" type="checkbox"/> | |
| 4. Program dapat menampilkan lintasan terpendek serta jaraknya. | <input checked="" type="checkbox"/> | |
| 5. Bonus: Program dapat menerima input peta dengan Google Map API dan menampilkan peta serta lintasan terpendek pada peta. | <input checked="" type="checkbox"/> | |

DAFTAR REFERENSI

1. Maulidevi, Nur Ulfa. 2021. *Penentuan Rute (Route/Path Planning) Bagian 1: BFS, DFS, UCS, Greedy Best First Search*. (Dikases pada tanggal 6 April 2023 dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf>)
2. Maulidevi, N.U., dan Rinaldi Munir. 2021. *Penentuan Rute (Route/Path Planning) Bagian 2: Algoritme A**. (Dikases dari pada tanggal 6 April 2023 dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf>)
3. Chris, Kolade. 13 September 2022. *Sort Dictionary by Value in Python – How to Sort a Dict*. (Diakses dari <https://www.freecodecamp.org/news/sort-dictionary-by-value-in-python/> pada tanggal 6 April 2023)
4. Loem, Mengsay. 27 Mei 2020. *Network Optimization(1): Shortest Path Problem*. (Diakses dari pada tanggal 8 April 2023 <https://medium.com/swlh/network-optimization-1-shortest-path-problem-3757a67a129c>)
5. Munir, Rinaldi. Tanpa tahun. *Graf (Bag. 1); Bahan Kuliah IF2120 Matematika Diskrit*. (Diakses dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf> pada tanggal 8 April 2023)
6. Trivusi. 17 Oktober 2022. *Apa itu Uniform-Cost Search? Pengertian dan Cara Kerjanya*. (Diakses pada tanggal 9 April 2023 dari <https://www.trivusi.web.id/2022/10/apa-itu-algoritme-uniform-cost-search.html>)
7. manjeet_04, dkk. 15 Maret 2023. *Python – Convert Delimiter separated list to Number*. (Diakses dari <https://www.geeksforgeeks.org/python-convert-delimiter-separated-list-to-number/> pada tanggal 9 April 2023)
8. Simic, Milos. 6 November 2022. *Weighted vs. Unweighted Graphs*. (Diakses pada tanggal 11 April 2023 dari <https://www.baeldung.com/cs/weighted-vs-unweighted-graphs>)
9. Zola, Andrew. September 2022. *Google Maps*. (Diakses pada tanggal 11 April 2023 dari <https://www.techtarget.com/whatis/definition/Google-Maps>)
10. Raharjo, Budi. 2019. *Mudah Belajar Python untuk Aplikasi Desktop dan Web (Edisi Revisi)*. Bandung: Penerbit Informatika.
11. Kettle, Simon. 10-5-2017. *Distance on a sphere: The Haversine Formula*. (Diakses dari <https://community.esri.com/t5/coordinate-reference-systems-blog/distance-on-a-sphere-the-haversine-formula/ba-p/902128> pada tanggal 11 April 2023)