Implementasi Algoritma *Uniform Cost Search* dalam Penentuan Rute Mudik di Sumatera Barat

Kayla Namira Mariadi - 13522050[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522050@std.stei.itb.ac.id

***Abstract*—Berbagai kendala sering terjadi ketika mudik yang dapat menghambat mobilisasi, misalnya kemacetan karena tingginya arus mobilisasi. Rekomendasi rute terpendek sangatlah dibutuhkan pemudik untuk mengurangi waktu dan biaya yang dibutuhkan. Makalah ini akan membahas cara menentukan rute terpendek yang efisien bagi pemudik dengan memanfaatkan algoritma Uniform Cost Search yang diimplementasikan dengan bahasa python.**

***Keywords*—Rute terpendek, Uniform Cost Search, Graf, Sumatera Barat.**

I. PENDAHULUAN

Mudik adalah kegiatan tahunan yang sudah umum terjadi di Indonesia. Saat mudik, biasanya orang atau perantau yang tinggal di berbagai kota akan kembali ke kampung atau kota asalnya. Mudik biasanya dilakukan menjelang libur hari besar keagamaa seperti Idul Fitri, Natal dan Tahun Baru, atau Idul Adha.

Saat musim mudik, kerap terjadi permasalahan yang dihadapi pemudik, salah satunya kemacetan. Kemacetan itu bisa disebabkan tingginya jumlah pengguna kendaraan, kerusakan jalan, kecelakaan, keramaian, atau faktor lainnya. Hal ini tentunya dapat menghambat alur mobilisasi pemudik dan memperlambat waktu perjalanan. Oleh karena itu, pemilihan rute jalan yang tepat dapat meminimalisir kendala yang dihadapi.

Sumatera Barat sebagai salah satu provinsi dengan jumlah perantau terbanyak menjadi salah satu destinasi mudik terpopuler. Berdasarkan laporan Statistik Migrasi Indonesia Hasil Long Form Sensus Penduduk 2020, posisi ketujuh provinsi dengan perantau atau penduduk yang melakukan migrasi keluar terbanyak ditempati oleh Sumatera Barat, yakni dengan jumlah 980.911 jiwa. Pemudik ini tidak hanya dari luar provinsi saja, tapi juga dari dalam provinsi.

Pada makalah ini, akan diimplementasikan cara untuk menentukan rute mudik terpendek antarwilayah kota/kabupaten di Sumatera Barat. Untuk menentukan rute mudik yang paling baik, dapat digunakan algoritma *shortest path* atau penentuan rute terpendek. Salah satu algoritma penentuan rute yang dapat digunakan adalah algoritma *Uniform Cost Search* (UCS). Dengan algoritma ini, dapat ditentukan menemukan rute yang tepat dan efisien bagi pemudik di Sumatera Barat agar dapat menempuh perjalanan menuju daerah tujuan.

II. LANDASAN TEORI

A. Graf

Graf G didefinisikan sebagai pasangan himpunan (V,E), yaitu:

V = Himpunan tidak kosong dari simpul

$$V = \{v1\ ,\ v,\ v3\ ,\ \dots\ ,\ vn\}$$

E = Himpunan sisi (edges) yang menghubungkan sepasang simpul

$$E = \{e1\ ,\ e,\ e3\ ,\ \dots\ ,\ en\}$$

Berdasarkan keberadaan gelang, graf dibedakan menjadi 2 jenis, yaitu:

1. Graf sederhana (*simple graph*), yaitu graf yang tidak mengandung gelang maupun sisi ganda.
2. Graf tak sederhana (*unsimple graph*), yaitu graf yang memiliki gelang atau sisi ganda

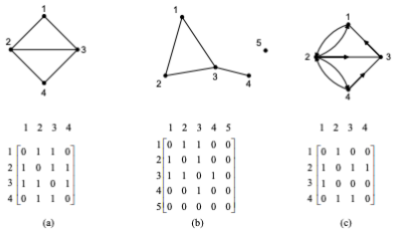
Berdasarkan orientasi arah pada sisi, graf dibedakan menjadi 2 jenis, yaitu:

1. Graf berarah (*directed graph*), yaitu graf yang setiap sisinya diberikan orientasi arah.

2. Graf tak-berarah (*undirected graph*), yaitu graf yang setiap sisinya tidak mempunyai orientasi arah

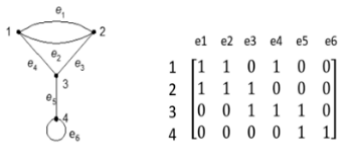
Graf dapat direpresentasikan dengan beberapa cara, yaitu:

1. *Adjacency Matrix* (Matriks Bertetanggaan)
   Martiks bertetanggaan merupakan matriks yang merepresentasikan nilai keterhubungan antar simpul pada sebuah graf. Pada gambar graf dibawah merupakan graf yang tidak memiliki bobot nilai, tetapi akan bernilai 1 ketika dua simpul terhubung dan 0 ketika tidak terhubung. Sedangkan pada graf berbobot, nilai yang ditunjukkan adalah nilai/bobot dari sebuah sisi yang menghubungkan dua simpul.



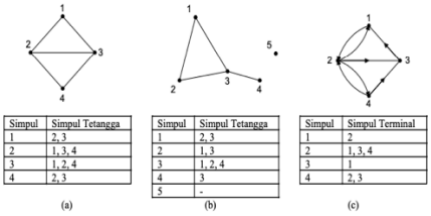
Gambar 1.1. *Adjacency Matrix (Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian2.pdf)*

2. *Incidence Matrix* (Matriks Bersisian)
   Matriks bersisian merupakan matriks yang merepresentasikan hubungan antara simpul dan sisi dari sebuah graf. Baris pada matriks akan di misalkan sebagai simpul dan kolom sebagai sisi. Jika simpul A dan sisi E bersisian, maka akan ditandai dengan 1, sedangkan jika tidak, akan ditandai dengan 0.



Gambar 1.2. *Incidence Matrix (Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian2.pdf)*

3. *Adjacency* List (Senarai Ketetanggan)
   Senarai ketetanggan merupakan senarai yang mencatat setiap simpul yang bertetangga dengan suatu simpul.



Simpul	Simpul Tetangga
1	2, 3
2	1, 3, 4
3	1, 2, 4
4	2, 3

(a)

Simpul	Simpul Tetangga
1	2, 3
2	1, 3
3	1, 2, 4
4	3
5	-

(b)

Simpul	Simpul Terminal
1	2
2	1, 3, 4
3	1
4	2, 3

(c)

Gambar 1.3. *Adjacency List (Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian2.pdf)*

B. Algoritma Uniform Cost Search

Algoritma *Uniform Cost Search* termasuk salah satu algoritma yang digunakan dalam pencarian rute dan termasuk ke dalam *uninformed search*. Algoritma yang termasuk ke dalam uninformed search adalah *Breadth First Search, Depth First Search, Uniform Cost Search, Iterative Deepening Search,* dan *Depth Limited Search*. Algoritma ini biasa dipakai dalam pencarian sebuah jalur dengan total bobot yang paling minimum.

Dalam implementasinya, algoritma ini memanfaatkan *priority queue* untuk menyimpan simpul dengan bobotnya masing–masing, bobot ini menentukan urutan/prioritas. Simpul selanjutnya yang diperiksa diambil dari awal *priority queue* dan dihapus dari *priority queue* setelah diperiksa. Dalam pembangkitan simpul ekspan, nilai bobot dari simpul tetangganya bernilai bobot jarak dari simpul awal hingga simpul tetangga. Simpul-simpul dalam data yang diberikan akan terus ditelusuri berdasarkan *priority queue* yang dibuat sampai diperiksa simpul yang dicari.

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    frontier ← a priority queue ordered by PATH-COST, with node as the only element
    explored ← an empty set
    loop do
        if EMPTY?(frontier) then return failure
        node ← POP(frontier)   /* chooses the lowest-cost node in frontier */
        if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
        add node.STATE to explored
        for each action in problem.ACTIONS(node.STATE) do
            child ← CHILD-NODE(problem, node, action)
            if child.STATE is not in explored or frontier then
                frontier ← INSERT(child, frontier)
            else if child.STATE is in frontier with higher PATH-COST then
                replace that frontier node with child
```

Gambar 1.4. Cuplikan *Pseudocode* Algoritma *Uniform Cost Search*

Uniform Cost Search memiliki kemiripan dengan Algoritma Dijkstra yang sama-sama merupakan algoritma pencarian rute terpendek. Perbedaan algoritma *Uniform Cost Search* dengan algoritma

Dijkstra yaitu pada Algorima Dijkstra, semua simpul dimasukkan ke dalam *priority queue* dan diinisialisasi dengan bobot tak hingga sehingga secara penggunaan memori lebih boros.

III. IMPLEMENTASI

A. Batasan Masalah

Pada makalah ini, terdapat beberapa asumsi dan batasan dalam penyederhanaan penentuan rute, berikut beberapa asumsi pada makalah ini:

1. Rute hanya ditentukan berdasarkan jarak tempuh minimum antardaerah. Terdapat faktor eksternal seperti kemacetan, tingginya arus mobilisasi, kualitas jalan, atau faktor lainnya yang tidak menjadi pertimbangan dalam makalah ini.
2. Tiap jalur diasumsikan merupakan jalur 2 arah sehingga implementasi masalah direpresentasikan dengan graf tidak berarah (*undirected graph*).
3. Dari 19 kabupaten/kota di Sumatera Barat menurut data dari https://sumbar.bps.go.id/statictable/2023/05/31/472/nama-ibukota-kabupaten-kota-di-provinsi-sumatera-barat.html, hanya diambil nama 16 kabupaten/kota. 3 kabupaten/kota yang tidak termasuk batasan makalah ini adalah Kabupaten Solok (diasumsikan sama dengan Kota Solok), Kabupaten Padang Pariaman (diasumsikan sama dengan Kota Pariaman), dan Kabupaten Kepulauan Mentawai (tidak terhubung dengan wilayah lainnya melalui jalur darat karena merupakan kepulauan)

B. Pemodelan Masalah

Pertama, diambil data 16 kabupaten dan kota berdasarkan peta wilayah Sumatera Barat. Jarak terpendek antara 2 wilayah yang dapat dilalui kendaraan roda-2 maupun roda-4 juga ditentukan dengan menggunakan bantuan Google Maps.

Data yang didapat kemudian dimodelkan pada suatu graf yang menunjukkan rute atau hubungan antarwilayah pada Sumatera Barat. Graf berupa graf berbobot dengan tiap simpul merepresentasikan suatu wilayah kabupaten/kota dan tiap sisi merepresentasikan adanya jalur dari suatu wilayah ke wilayah lain, sedangkan bobot merepresentasikan jarak minimum antar 2 wilayah yang bertetangga pada graf. Simpul dipilih berdasarkan pusat dari kota/kabupaten, sedangkan sisi dipilih berdasarkan adanya jalur antar 2 wilayah atau simpul. Perhatikan bahwa dua wilayah yang berbatasan langsung pada peta belum tentu terhubung karena faktor kontur wilayah dan keterbatasan fasilitas sehingga pada graf tidak ada sisi yang menghubungkan kedua wilayah tersebut. Misalnya, Kabupaten Pesisir Selatan dan Kota Solok terlihat berbatasan pada peta, namun pada graf tidak bertetangga karena belum ada jalur darat yang langsung menghubungkan kedua wilayah tersebut.

Berikut gambaran peta dan rute antarwilayah kabupaten/kota di Sumatera Barat dengan graf:

Gambar 1. Pemodelan peta dan rute antarwilayah kabupaten/kota di Sumatera Barat pada graf
(Sumber: https://commons.wikimedia.org/wiki/File:Prov._Sumatera_Barat.jpg dengan modifikasi)

Untuk mempermudah pengolahan data, nama tiap kabupaten/kota direpresentasikan oleh simpul yang dilabeli angka 1-16.

No.	Nama Kabupaten/Kota	Simpul
1.	Pasaman Barat	*v1*
2.	Pasaman	*v2*
3.	Bukittingi	*v3*
4.	Agam	*v4*
5.	Payakumbuh	*v5*
6.	Padang Panjang	*v6*
7.	Tanah Datar	*v7*
8.	Sijunjung	*v8*
9.	Sawahlunto	*v9*
10.	Solok Selatan	*v10*

11.	Pesisir Selatan	*v11*
12.	Solok	*v12*
13.	Dharmasraya	*v13*
14.	Padang	*v14*
15.	Pariaman	*v15*
16.	Lima Puluh Kota	*v16*

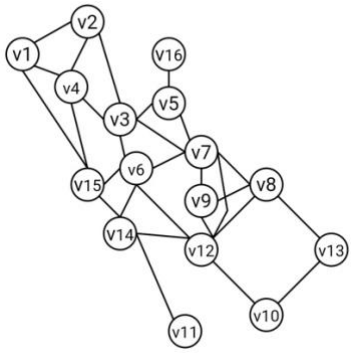
Tabel 1. Representasi Nama Kabupaten/Kota di Sumatera Barat oleh Simpul pada Graf

Bobot untuk tiap sisi adalah jarak tiap wilayah pada peta (dalam km) dengan pembulatan dan dihitung dengan bantuan Google Maps. Berikut informasi mengenai sisi yang ada pada graf beserta bobotnya:

No.	Sisi (Simpul, Simpul)	Jarak (dalam km)
1.	(v1, v2)	130
2.	(v2, v3)	86
3.	(v3, v4)	47
4.	(v3, v5)	32
5.	(v3, v6)	21
6.	(v3, v7)	44
7.	(v4, v1)	130
8.	(v4, v2)	113
9.	(v5, v7)	39
10.	(v6, v12)	54
11.	(v7, v6)	29
12.	(v7, v8)	69
13.	(v8, v9)	39
14.	(v8, v12)	51
15.	(v9, v12)	45
16.	(v9, v7)	30
17.	(v11, v14)	78
18.	(v12, v7)	46
19.	(v12, v10)	96
20.	(v12, v14)	53
21.	(v13, v8)	112
22.	(v13, v10)	52
23.	(v14, v6)	76
24.	(v14, v15)	55
25.	(v15, v1)	157
26.	(v15, v6)	46
27.	(v15, v4)	76
28.	(v16, v5)	13

Tabel 2. Representasi Hubungan Antar Kabupaten/Kota oleh Sisi pada Graf beserta Jaraknya

Berikut graf sederhana, berbobot, terhubung, dan tak-berarah yang dihasilkan:



Gambar 3.2. Pemodelan rute antarwilayah kabupaten/kota di Sumatera Barat pada graf berbobot (Sumber : Dokumen pribadi)

Setelah graf hubungan antarwilayah kabupaten/kota di Sumatera Barat terbentuk, dapat ditentukan rute optimum antar 2 wilayah dengan algoritma UCS.

C. Penerapan Algoritma UCS

Berikut langkah-langkah penerapan Algoritma UCS:

1. Inisialisasi array *visited* yang akan berisi simpul yang telah dikunjungi, *array path* untuk menyimpan rute, dan *priority queue.* Simpul awal dimasukkan ke *queue*.
2. Lakukan iterasi selama *priority queue* tidak kosong atau simpul tujuan belum ditemukan.
3. Pada tiap iterasi, diambil simpul dengan bobot minimum yang merupakan *head* dari *priority queue*
4. Jika simpul merupakan tujuan dari rute, hentikan pencarian dan keluarkan hasil *path*.
5. Iterasi untuk setiap tetangga dari simpul. Jika simpul tetangga belum ada dalam priority queue, simpul ditambahkan dengan bobotnya ditambah bobot dari simpul-simpul pada *path* sebelumnya.

Penerapan algoritma UCS ditampilkan pada tabel yang berisi jumlah iterasi, simpul ekspan (simpul yang sedang diperiksa), dan simpul hidup (simpul

yang ada pada *Priority Queue*). Simpul direpresentasikan dengan angka dan index merepresentasikan *path* atau simpul simpul yang dilewati sebelumnya untuk menuju simpul tersebut, serta akumulasi bobotnya.

Sebagai contoh implementasi proses algoritma UCS, akan dipilih Kota Padang sebagai simpul awal dan Kota Payakumbuh sebagai simpul tujuan. Algoritma juga bisa diterapkan untuk simpul awal dan tujuan yang berbeda. Berikut proses pencarian rute terpendek dari Padang menuju Payakumbuh dengan UCS.

Iterasi	Simpul Ekspan	Simpul Hidup
1	v14	v12v14-53, v15v14-55, v6v14-76, v11v14-78
2	v12v14-53	v151-55, v6v14-76, v11v14-78, v9 v14v12-98, v7v14v12-99, v8v14v12-104, v6v14v12-107, v10 v14v12-149
3	v15v14-55	v6v14-76, v11v14-78, v9 v14v12-98, v7v14v12-99, v6v14v15-101, v8v14v12-104, v6v14v12-107, v4v14v15-131, v10 v14v12-149, v1v14v15-212
4	v6v14-76	v11v14-78, v3v14v6-97, v9 v14v12-98, v7v14v12-99, v6v14v15-101, v8v14v12-104, v6v14v12-107, v7v14v6-120, v4v14v15-131, v10 v14v12-149, v1v14v15-212
5	v11v14-78	v3v14v6-97, v9 v14v12-98, v7v14v12-99, v6v14v15-101, v8v14v12-104, v6v14v12-107, v7v14v6-120, v4v14v15-131, v10 v14v12-149, v1v14v15-212
6	v3v14v6-97	v9 v14v12-98, v7v14v12-99, v6v14v15-101, v8v14v12-104, v6v14v12-107, v7v14v6-120, v5v14v6v3-129, v4v14v15-131, v7v14v6v3-141, v4v14v6v3-144, v10 v14v12-149, v2v14v6v3-183, v1v14v15-212
7	v9 v14v12-98	v7v14v12-99, v6v14v15-101, v8v14v12-104, v6v14v12-107, v7v14v6-120, v7v14v12v9-128, v5v14v6v3-129, v4v14v15-131, v8v14v12v9-137, v7v14v6v3-141, v4v14v6v3-144, v10 v14v12-149, v2v14v6v3-183, v1v14v15-212
		v8v14v12v9-137, v7v14v6v3-141, v4v14v6v3-144, v10 v14v12-149, v2v14v6v3-183, v1v14v15-212
8	v7v14v12-99	v6v14v15-101, v8v14v12-104, v6v14v12-107, v7v14v6-120, v7v14v12v9-128, v5v14v6v3-129, v4v14v15-131, v8v14v12v9-137, v5v14v12v7-138, v7v14v6v3-141, v4v14v6v3-144, v10 v14v12-149, v8v14v12v7-168, v2v14v6v3-183, v1v14v15-212
9	v6v14v15-101	v8v14v12-104, v6v14v12-107, v7v14v6-120, v7v14v12v9-128, v5v14v6v3-129, v4v14v15-131, v8v14v12v9-137, v5v14v12v7-138, v7v14v6v3-141, v4v14v6v3-144, v10 v14v12-149, v8v14v12v7-168, v2v14v6v3-183, v1v14v15-212
10	v8v14v12-104	v6v14v12-107, v7v14v6-120, v7v14v12v9-128, v5v14v6v3-129, v4v14v15-131, v8v14v12v9-137, v5v14v12v7-138, v7v14v6v3-141, v4v14v6v3-144, v10 v14v12-149, v8v14v12v7-168, v2v14v6v3-183, v1v14v15-212, v13v14v12v8-216
10	v6v14v12-107	v7v14v6-120, v7v14v12v9-128, v5v14v6v3-129, v4v14v15-131, v8v14v12v9-137, v5v14v12v7-138, v7v14v6v3-141, v4v14v6v3-144, v10 v14v12-149, v8v14v12v7-168, v2v14v6v3-183, v1v14v15-212, v13v14v12v8-216
11	v7v14v6-120	v7v14v12v9-128, v5v14v6v3-129, v4v14v15-131, v8v14v12v9-137, v5v14v12v7-138, v7v14v6v3-141, v4v14v6v3-144, v10 v14v12-149, v8v14v12v7-168, v2v14v6v3-183, v1v14v15-212, v13v14v12v8-216
12	v7v14v12v9-128	v5v14v6v3-129, v4v14v15-131, v8v14v12v9-137, v5v14v12v7-138, v7v14v6v3-141, v4v14v6v3-144, v10 v14v12-149, v8v14v12v7-168, v2v14v6v3-183, v1v14v15-212, v13v14v12v8-216
10	v5v14v6v3-129	Solusi ditemukan

Tabel 3. Proses Iterasi Algoritma UCS

Berdasarkan tabel di atas, rute terpendek yang

dapat ditempuh dari Kota Padang (v14) ke Kota Payakumbuh (v5) yaitu dari Kota Padang melalui Kota Padang Panjang, Kota Bukittinggi, dan sampai di Kota Payakumbuh dengan total jarak 129 km.

D. Implementasi Algoritma UCS dalam Bahasa Python

Implementasi program dilakukan dengan bahasa pemrograman python. Berikut tahapan implementasi program :

1. Inisialisasi *Adjacency List* berdasarkan Data Kabupaten/Kota beserta Jaraknya

   Program pertama menerima masukan berupa file *data.txt* berisi sisi-sisi yang terdapat pada graf beserta bobotnya (nama kabupaten/kota asal, tujuan, beserta jaraknya). File kemudian dibaca oleh program dengan fungsi *make_adjacency_list* yang menerima parameter nama file.

```python
# membuat adjacency_list dari masukan file txt
def make_adjacency_list(filename):
    graph = {}
    file = open(filename, 'r')
    for line in file:
        if 'END OF INPUT' in line:
            return graph

        start, end, d = line.split()
        graph.setdefault(start, []).append((end, d))
        graph.setdefault(end, []).append((start, d))
```

Gambar 3.3. Cuplikan fungsi *make_adjacency_list*

```
ucs.py    data.txt
data.txt
1  Pasaman_Barat Pasaman 130
2  Pasaman Bukittinggi 86
3  Bukittinggi Agam 47
4  Bukittinggi Payakumbuh 32
5  Bukittinggi Padang_Panjang 21
6  Bukittinggi Tanah_Datar 44
7  Agam Pasaman_Barat 130
8  Agam Pasaman 113
9  Payakumbuh Tanah_Datar 39
```

Gambar 3.4. Cuplikan file *data.txt*

Fungsi mengembalikan *adjacency list* yang direpresentasikan oleh *dictionary* dengan *key* berupa nama simpul (nama kabupaten/kota) dan *item* berupa list simpul tetangganya beserta bobot/jaraknya.

```
Adjacency list :
{'Pasaman_Barat': [('Pasaman', 130), ('Agam', 130), ('Pariaman', 157)], 'Pasaman': [('Pasaman_Barat', 130), ('Bukittinggi', 86), ('Agam',
ttinggi': [('Pasaman', 86), ('Agam', 47), ('Payakumbuh', 32), ('Padang_Panjang', 21), ('Tanah_Datar', 44)], 'Agam': [('Bukittinggi', 47),
rat', 130), ('Pasaman', 113), ('Pariaman', 76)], 'Payakumbuh': [('Bukittinggi', 32), ('Tanah_Datar', 39), ('Lima_Puluh_Kota', 13)], 'Padan
[('Bukittinggi', 21), ('Solok', 54), ('Tanah_Datar', 29), ('Padang', 76), ('Pariaman', 46)], 'Tanah_Datar': [('Bukittinggi', 44), ('Payaku
('Padang_Panjang', 29), ('Sijunjung', 69), ('Sawahlunto', 38), ('Solok', 46)], 'Solok': [('Padang_Panjang', 54), ('Sijunjung', 51), ('Sawa
, ('Tanah_Datar', 46), ('Solok_Selatan', 96), ('Padang', 53)], 'Sijunjung': [('Tanah_Datar', 69), ('Sawahlunto', 39), ('Solok', 51), ('Dha
12)], 'Sawahlunto': [('Sijunjung', 39), ('Solok', 45), ('Tanah_Datar', 38)], 'Pesisir_Selatan': [('Padang', 78)], 'Padang': [('Pesisir_Sel
('Solok', 53), ('Padang_Panjang', 76), ('Pariaman', 55)], 'Solok_Selatan': [('Solok', 96), ('Dharmasraya', 52)], 'Dharmasraya': [('Sijunjun
'Solok_Selatan', 52)], 'Pariaman': [('Padang', 55), ('Pasaman_Barat', 157), ('Padang_Panjang', 46), ('Agam', 76)], 'Lima_Puluh_Kota': [('P
13)]}
```

Gambar 3.5. *Adjacency List* yang terbentuk

2. Implementasi Algoritma UCS

   Berikut implementasi kode program pencarian rute terpendek antarwilayah di Sumatera Barat dengan menerapkan algoritma UCS:

```
data.txt    ucs.py
ucs.py > ...
1  import sys
2  from queue import PriorityQueue
3
4  # Implementasi algoritma UCS
5  def ucs(graph, start, end):
6      visited_vertex = set() #Dict berisi simpul yang telah dikunjungi
7      path = [] #List berisi path menuju simpul
8
9      queue = PriorityQueue() #Inisialisasi prioqueue
10     queue.put((0, [start])) #Inisialisasi dengan data simpul start dan bobot 0
```

Gambar 3.6. Inisialisasi variabel

Fungsi menerima parameter *graph* yang direpresentasikan *adjacency* list, simpul awal atau *start*, dan simpul tujuan atau *end*. Pertama, diinisiasi *dictionary* visited_vertex yang akan berisi simpul-simpul yang telah diekspan. Lalu, diinisialisasi list *path* kosong yang digunakan untuk menyimpan rute menuju suatu simpul. Diinisiasi pula variabel *queue* sebagai *Priority Queue* yang akan menyimpan simpul-simpul hidup beserta bobotnya dari simpul asal. Pada kondisi awal, *queue* hanya berisi simpul asal dan bobotnya diinisialisasi sebagai 0. Implementasi *queue* memanfaatkan *library* PriorityQueue python.

```
12     while queue:
13         #Handle prioque kosong
14         if queue.empty():
15             print('Tak ada rute!')
16             return
17
18         weight, path = queue.get() #Mengambil simpul ekspan dari prioque
19         vertex = path[-1]
20         if vertex not in visited_vertex:
21             visited_vertex.add(vertex) #Menandai simpul yang sudah dikunjungi
22             if vertex == end: #Mengakhiri loop jika simpul tujuan ditemukan
23                 path.append(weight)
24                 return path
25
26             # Menentukan jarak menuju simpul tetangga
27             for neighbor,distance in graph[vertex]:
28                 if neighbor not in visited_vertex:
29                     new_weight = weight + int(distance)
30                     temp = path[:]
31                     temp.append(neighbor)
32                     queue.put((new_weight, temp))
```

Gambar 3.7. *Looping* pada algoritma UCS

Kemudian, akan dilakukan *looping* selama queue tidak kosong, jika queue kosong akan keluar dari *loop*. Lalu, dilakukan *dequeue* elemen pertama pada *queue* untuk mendapatkan simpul beserta bobot dan *path*-nya. Simpul ini merupakan simpul ekspan dan memiliki bobot yang paling minimum pada *queue*. Dilakukan pengecekan apakah simpul sudah pernah dikunjungi sebelumnya terlebih dahulu. Jika tidak, simpul tersebut ditambahkan ke *visited_vertex*.

Dilakukan pengecekan apakah simpul merupakan simpul akhir atau *end*, jika ya maka fungsi akan mengembalikan *path* dari simpul akhir tersebut. Jika bukan simpul akhir, dilakukan penelusuran untuk tiap tetangga simpul ekspan pada *graph*. Jika simpul tetangga belum ada pada *visited_vertex*, simpul tetangga tersebut beserta bobotnya ditambah bobot

dari simpul-simpul pada *path* sebelumnya akan dimasukkan ke *queue.*

Proses pembangkitan simpul ekspan dan penelusuran tetangganya dilakukan hingga ditemukan simpul *end* atau tujuan. Jika *queue* kosong, maka tidak ditemukan rute yang diinginkan. Fungsi ucs lalu akan mengembalikan *path* dari simpul tujuan.

3. Implementasi Fungsi *main*

   Berikut implementasi fungsi *display_path* yang mencetak rute serta jarak ke layar.

```python
# Output graph
def display_path(path):
    distance = path[-1]
    print('Jarak : '+ str(distance) + ' km')
    print('Rute: ')
    for i in range(len(path)-1):
        print(f"{path[i]}", end=" ")
        if (i != len(path)-2):
            print("->", end=" ")
        else:
            print()
```

Gambar 3.8. Implementasi Fungsi *display_path*

Pada fungsi *main*, pertama akan diminta input wilayah asal dan tujuan yang akan dicari rutenya, dilakukan pula validasi input. Lalu, diinisialisasi *graph* dengan memanggil fungsi *make_adjacency_list.* Lalu, rute antara wilayah asal dan tujuan akan dicari dengan memanggil fungsi ucs dan hasilnya akan dicetak dengan fungsi *display_path*.

```python
def main():
    source = input("Asal : ")
    destination = input("Tujuan : ")
    print()

    graph = {}
    graph = make_adjacency_list('data.txt')
    if source not in graph.keys():
        print('Wilayah asal tidak ditemukan')
        sys.exit()
    if destination not in graph.keys():
        print('Wilayah tujuan tidak ditemukan')
        sys.exit()

    path = []
    path = ucs(graph, source, destination)

    if path:
        display_path(path)

if __name__ == '__main__':
    main()
```

Gambar 3.9. Implementasi Fungsi *main*

IV. Analisis dan Pembahasan

A. Hasil Eksekusi Program

Berikut hasil eksekusi program pencarian rute terpendek yang dapat ditempuh dari Kota Padang ke Kota Payakumbuh.

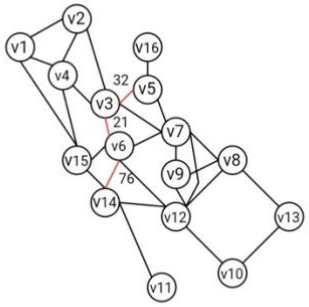
```
Kaylas-MacBook-Air:matdis usagitsukino$ python3 ucs.py
Asal : Padang
Tujuan : Payakumbuh

Jarak : 129 km
Rute:
Padang -> Padang_Panjang -> Bukittinggi -> Payakumbuh
Kaylas-MacBook-Air:matdis usagitsukino$
```

Gambar 4.1. Hasil eksekusi program pencarian rute Padang-Payakumbuh

Hasil eksekusi sesuai dengan tabel 3, yaitu rute melalui Kota Padang, Kota Padang Panjang, Kota Bukittinggi, dan Kota Payakumbuh dengan total jarak 129 km.

Berikut visualisasi graf yang telah disimplifikasi dalam pencarian rute terpendek yang dapat ditempuh dari Kota Padang ke Kota Payakumbuh dengan algoritma UCS.



Gambar 4.2. Visualisasi graf pencarian rute terpendek yang dapat ditempuh dari Kota Padang ke Kota Payakumbuh dengan algoritma UCS (Sumber : Dokumen pribadi)

B. Analisis Kompleksitas Waktu

Misalkan b adalah *branching factor*, c adalah *cost* atau bobot dari rute terbaik, dan ε adalah *cost* atau bobot minimum antara 2 simpul. Dalam algoritma yang diimplementasikan, setiap simpul diekspan sebelum ditemukan simpul tujuannya sehingga kompleksitas waktunya adalah $O(b^{(C/\varepsilon)})$. Karena tiap simpul juga ditambahkan ke dalam *priority queue* pada kasus terburuk maka kompleksitas ruangnya juga $O(b^{(C/\varepsilon)})$.

V. Kesimpulan

Dengan mengaplikasikan algoritma *Uniform Cost Search*, dapat dibuat suatu program penentuan rute terpendek antar wilayah di Sumatra Barat. Hasil dari algoritma ini adalah rute dengan bobot minimum dari

kota asal menuju kota tujuan sehingga didapatkan rute mudik terbaik. Dengan rekomendasi rute ini, pemudik harapannya dapat menempuh perjalanan dengan lebih singkat dan efisien dari daerah asal menuju daerah tujuan.

V. Ucapan Terima Kasih

Penulis menyampaikan ucapan terima kasih kepada:

1. Tuhan Yang Maha Esa atas berkat-Nya sehingga penulis dapat menyelesaikan makalah ini dengan baik,
2. Kedua orang tua penulis yang telah memberikan dukungan kepada penulis,
3. Bapak Dr. Ir. Rinaldi Munir, M.T., Ibu Fariska Zakhralativa Ruskanda S.T.,M.T, dan Ibu Dr. Nur Ulfa Maulidevi S.T., M.Sc selaku dosen mata kuliah IF2120 Matematika Diskrit yang telah membimbing penulis dan memberikan banyak ilmu yang bermanfaat selama perkuliahan.

Akhir kata, penulis mengharapkan makalah ini dapat bermanfaat bagi pihak yang membacanya.

References

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf2020-Bagian2.pdf diakses pada tanggal 10 Desember 2023
[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf2020-Bagian1.pdf diakses pada tanggal 10 Desember 2023
[3] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/RoutePlanning-Bagian1-2021.pdf diakses pada tanggal 10 Desember 2023
[4] https://inst.eecs.berkeley.edu/~cs188/su22/assets/slides/Lecture3.pdf diakses pada tanggal 10 Desember 2023
[5] Stuart Russell, Peter Norvig,. 2010. *Artificial Intelligence : a modern approach*. PE. New Jersey.

Pernyataan

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 11 Desember 2023

Kayla Namira Mariadi 13522050