Penerapan Algoritma *Depth-First Searching* dan *Greedy* dalam Menentukan Banyak Kota yang Bisa Dikunjungi dengan Waktu yang Ditentukan

Pandyaka Aptanagi - 13517003
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl.Ganesha 10 Bandung 40132, Indonesia
13517003@std.stei.itb.ac.id

***Abstrak*—Pada saat ini, perkembangan teknologi sudah berkembang sangat pesat dan ada di semua lini kehidupan masyarakat. Banyak hal yang sebelumnya harus dilakukan secara manual, sekarang sudah mulai terotomatisasi semuanya. Seperti saat menentukan dalam sekian hari, bisa pergi berlibur kemana saja dengan tujuan sebanyak mungkin. Pada makalah ini, penulis melakukan implementasi dua algoritma yang cukup baik untuk membantu menentukan permasalahan tersebut, yaitu algoritma *Depth-First-Searching* dan algoritma *Greedy*. Memang, dalam menentukan penyelesaian dari permasalahan ini tidak hanya bisa menggunakan dua algoritma tersebut, tetapi menurut penulis dua algoritma ini sudah cukup baik untuk menyelesaikan permasalahan ini. Selanjutnya penuulis juga menuliskan algoritma dalam bahasa Python untuk mempermudah pembaca memahami bagaimana algoritma *Depth-First-Searching* dan *Greedy* terimplementasikan.**

Kata Kunci—Kota; Terbanyak; Optimasi; Jarak; Maksimum;

I. PENDAHULUAN

Manusia memiliki waktu yang terbatas dalam hidupnya. Dalam sehari, tidak mungkin untuk melakukan sebuah kegiatan yang benar-benar fokus pada kegiatan tersebut selama 24 jam lamanya dikarenakan pekerjaan yang lain sudah menunggu untuk dikerjakan. Salah satu waktu yang sangat berharga adalah waktu untuk berlibur. Liburan, selain digunakan untuk bersenang-senang, adalah waktu dimana kita akan melakukan *refreshing* dari kegiatan sehari-hari yang snagat padat dan melelahkan.

Karena waktu yang kita miliki terbatas, terkadang kita bingung ingin berlibur kemana, dengan waktu yang sangat terbatas. Jika ingin melakukan pencarian tempat secara manual, maka waktu yang seharusnya kita gunakan untuk berlibur, justru akan habis hanya untuk sekadar mencari tempat liburan, dan malah membuat kemungkinan untuk gagal berlibur semakin besar.

Oleh karena itu, dengan memanfaatkan algoritma *Depth-First-Searching* dan algoritma *Greedy*, akan dibuat sebuah penyelesaian dari permasalahan tersebut. Kombinasi dari algoritma *Depth-First-Searching* dan algoritma *Greedy* akan menghasilkan solusi yang cukup baik, meskipun tidak selalu optimal, sehingga waktu yang dibutuhkan untuk mencari tempat berlibur bisa dipangkas, dan digunakan untuk hal-hal yang lain.

Dalam makalah ini, penulis akan membahas tentang penerapan algoritma *Depth-First-Searching* dan *Greedy* dalam menyelesaikan permasalahan berapa banyak dan apa saja kota yang bisa dikunjungi dengan waktu yang telah ditentukan

Gambar 1.1 Ilustrasi keluarga berlibur

Sumber :
https://www.liputan6.com/lifestyle/read/2133620/tips-tetap-enerjik-dan-sehat-saat-liburan

II. DASAR TEORI

A. *Algoritma Traversal Graf*

Algoritma travelsal graf adalah algoritma yang mengunjungi simpul dengan cara yang sistematik, yaitu
1. Pencarian melebar (breadth-first search/BFS)
2. Pencarian Mendalam (Depth-first search/DFS)

Algoritma traversal graf juga dapat dibedakan sesuai dengan informasinya, yaitu:
1. Tanpa informasi (uninformed/blind search)
- Tidak ada informasi tambahan
- Contoh : DFS, BFS, Depth Limited Search, Iterative Deepening Search, Uniform Cost Search
2. Dengan informasi (informed Search)
- Pencarian berbasis heuristic
- Mengetahui non-goal state "lebih menjanjikan" daripada yang lain
- Contoh: Best First Search, A*

B. *Representasi Graf dalam Pencarian Solusi*

Dalam proses pencarian solusi, terdapat dua pendekatan yang dilakukan, yaitu :

1. Graf statis : Graf yang sudah terbentuk sebelum proses pencarian dilakukan
2. Graf dinamis : Graf yang terbentuk saat proses pencarian dilakukan.

C. *Algoritma Depth-First Searching*

Algoritma DFS ditemukan pada abad ke-19 oleh Matematikawan Perancis, Charles Pierre Tremaux sebagai strategi untuk menyelesaikan labirin. Algoritma tersebut disebut dengan *Tremaux's Algorithm*.

Pada awalnya, Algoritma Tremaux digunakan untuk menyelesaikan labirin dengan efisien dengan menggambarkan garis pada lantai menjadi sebuah jalur, dan dijamin bekerja untuk semua labirin yang memiliki jalan keluar. Algoritma ini digunakan selama ratusan tahun sebelum adanya algoritma DFS.

Algoritma ini bisa digunakan baik pada struktur data dengan bentuk *tree* maupun dalam bentuk *graph.* Hanya saja perbedaannya adalah jika pada *tree* tidak mungkin memiliki *cycle*, tetapi pada *graph* sangat mungkin untuk memiliki *cycle*, sehingga adanya perbedaan dalam pengimplementasian algoritma dengan struktur yang berbeda.

Algoritma DFS memiliki kompleksitas waktu O(|V| + |E|), sesuai dengan banyaknya simpul dan busur pada sebuah graf. DFS juga menggunakan ruang $O(|V|)$ pada kasus terburuknya untuk menyimpan stack dari setiap simpul pada pencarian jalur saat ini serta set simpul yang sudah dikunjungi.

Algoritma DFS bekerja dengan cara-cara sebagai berikut :

Pada sebuah graf, akan ada satu simpul *v*, dimana simpul tersebut menjadi simpul tempat dimulainya DFS.
Setelah itu :

1. Kunjungi simpul v
2. Kunjungi simpul w yang bertetangga dengan simpul v
3. Ulangi DFS mulai dari simpul w
4. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik (backtrack) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.
5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

Algoritma DFS bisa dilakukan dalam dua cara, yaitu dengan metode *rekursif* dan metode *iteratif*. Metode iteratif lebih baik untuk digunakan, karena lebih hemat konsumsi memori daripada metode *rekursif* .

Metode *rekursif :*

```
procedure dfs (input pos:integer)
{ Melakukan pencarian solusi pada sebuah graf }
If (kondisi sudah memenuhi) {
   Kembalikan solusi dalam bentuk yang diinginkan }
For (tiap indeks dalam ruang solusi) {
    If (dapat menghasilkan solusi) {
        Masukkan dalam solusi
        dfs(posisi++)
    }
}
```

Metode *iteratif :*

```
procedure dfs (input pos:integer) {
stack<int> st;
st.push(pos)
while (!st.empty()) {
     Int temp = st.top()
     st.pop()
     For(tiap indeks dalam ruang solusi) {
            If (dapat menghasilkan solusi) {
                 st.push(indeks) }
      }
   }
}
```

D. Algoritma Greedy

Algoritma greedy adalah salah satu algoritma terpopuler untuk menyelesaikan permasalahan optimasi. Pemasalahan optimasi adalah masalah untuk mencari solusi yang paling optimum, baik itu paling maksimum atau paling minimum.

Algoritma greedy memiliki prinsip yaitu "*Take What you Can Get Now!"*, yang berarti pada saat itu, langkah yang paling optimum adalah langkah yang akan diambil dari sekian banyak kemungkinan langkah yang ada, dan tidak bisa kembali lagi ketika sudah mengambil langkah.

Algoritma greedy memiliki langkah umum seperti berikut :

Pada setiap langkahnya,

1. mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan (prinsip "take what you can get now!")
2. berharap bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global.

Persoalan optimasi pada algoritma greedy akan memiliki beberapa elemen utama, yaitu :

1. Himpunan Kandidat

   Himpunan ini berisi semua pilihan keputusan yang dapat membangun solusi global. Jika memenuhi apa yang diinginkan program, maka pada setiap langkahnya, himpunan kandidat akan dipindahkan ke himpunan solusi yang akan membangun solusi global di akhir langkah nanti

2. Himpunan Solusi

   Himpunan ini berisi penerapan algoritma Greedy ke himpunan-himpunan kandidat pada setiap langkahnya. Keputusan yang paling baik pada setiap langkah, akan menjadi bagian dari himpunan solusi untuk menjadi solusi global pada akhir langkah.

3. Fungsi Seleksi

   Fungsi seleksi merupakan fungsi yang digunakan pada setiap langkah pencarian solusi untuk memilih elemen dari himpunan kandidat yang paling berpotensi untuk membentuk solusi global yang terbaik.

4. Fungsi Kelayakan

   Fungsi kelayakan adalah fungsi yang berguna untuk memeriksa kelayakan dari elemen himpunan kandidat yang telah dipilih oleh fungsi seleksi. Hal ini diperlukan karena dalam algoritma greedy, selalu terdapat syarat yang tidak boleh dilanggar dalam pengambilan langkah solusi. Fungsi kelayakan akan memeriksa kelayakan kandidat berdasarkan syarat yang telah ditentukan tersebut.

5. Fungsi Obyektif

   Fungsi obyektif merupakan fungsi yang berguna untuk memaksimumkan atau meminimumkan nilai solusi yang dicari.

Algoritma greedy adalah algoritma yang memiliki kelebihan berupa mudah untuk dipahami dan diimplementasikan. Selain itu, algoritma greedy juga dapat menghasilkan solusi yang cukup baik, meskipun tidak selalu optimal dikarenakan prinsipnya yang cukup unik.

Namun, algoritma ini juga memiliki kekurangan yang cukup fatal, yaitu sulit untuk didesain dikarenakan untuk setiap permasalahan, akan berbeda pula solusinya tergantung dengan faktor apa yang akan dinilai oleh greedy-nya. Selain itu, jika terdapat banyak faktor yang bisa dinilai, maka tidak cukup sulit pula untuk menentukan mana saja yang harus di greedy, dan menjadikan kompleksitasnya semakin rumit dan tidak mangkus.

III. Penyelesaian Permasalahan dengan Algoritma Greedy dan DFS

Dalam menyelesaikan permasalahan ini, penulis mengelompokkan langkah-langkahnya dalam dua langkah, yaitu :

A. Langkah Penyelesaian

Pencarian solusi diawali dengan merepresentasikan kota-kota yang ada pada basis data, dengan jarak antar kota disertakan pula, sehingga akan terbentuk sebuah matriks.

JAKARTA	SERANG	90
JAKARTA	PANDEGLA	111
JAKARTA	LEBAK	131
JAKARTA	TANGERAN	25
JAKARTA	BEKASI	29
JAKARTA	KARAWAN	71
JAKARTA	PURWAKA	113
JAKARTA	SUBANG	161
JAKARTA	BOGOR	58

Pada basis data saat makalah ini ditulis, penulis menyertakan 21 kota yang terhubung satu sama lain beserta jaraknya masing-masing.

Setelah membuat matrix yang berisi kota asal, kota tujuan, dan jarak, langkah selanjutnya adalah mengimplementasikan algoritma DFS dan greedy dengan menghubungkannya dengan matrix yang telah dibuat sebelumnya. Kemudian, juga akan

dimasukkan berapa lama pengguna memiliki waktu (dalam hari), dan diubah dengan rumus seperti berikut :

*JarakYangDimiliki = HariYangDimiliki * 4 * 40*

Dimana 4 berasal dari asumsi bahwa rata-rata jam untuk mengemudi adalah 4 jam saja, dan 40 adalah asumsi kecepatan rata-rata yang dimiliki oleh pengemudi adalah 40 km / jam.

Asumsikan bahwa HariYangDimiliki memiliki nilai 1, maka akan didapatkan JarakYangDimiliki dengan nilai 160 dalam satuan kilometer, yang akan digunakan dalam proses Greedy nantinya.

Sesuai dengan definisi algoritma DFS, maka diperlukan adanya simpul v sebagai simpul mulai dari algoritma DFS. Karena itu, akan dipilih salah satu kota sebagai titik awal dari algoritma DFS. Asumsikan kota pertama adalah Jakarta, sehingga simpul v pertama adalah titik Jakarta.

JAKARTA	SERANG	90
JAKARTA	PANDEGLA	111
JAKARTA	LEBAK	131
JAKARTA	TANGERAN	25
JAKARTA	BEKASI	29
JAKARTA	KARAWAN	71
JAKARTA	PURWAKA	113
JAKARTA	SUBANG	161
JAKARTA	BOGOR	58
JAKARTA	SUKABUM	119
JAKARTA	CIANJUR	122
JAKARTA	BANDUNG	187
JAKARTA	SUMEDAN	232
JAKARTA	GARUT	250
JAKARTA	TASIKMAL	293
JAKARTA	CIAMIS	308
JAKARTA	KUNINGAN	293
JAKARTA	MAJALENG	278
JAKARTA	CIREBON	258
JAKARTA	INDRAMAY	205

Setelah itu, algoritma greedy akan dijalankan dengan himpunan kandidat adalah jarak semua kota dari kota Jakarta. Dengan prinsip “Take What You Can Get Now!”, maka algoritma greedy akan memilih kota dengan jarak yang paling pendek dari Jakarta, yaitu Tangerang dengan jarak 25 kilometer.

Kemudian, simpul v akan pindah ke Tangerang, dan mengurangi JarakYangDimiliki menjadi :

160 – 25 = 135

Karena JarakYangDimiliki masih leih dari 0, maka DFS akan berlanjut untuk mencari kota selanjutnya, dengan titik awal dari Tangerang.

TANGERAN	JAKARTA	25
TANGERAN	SERANG	65
TANGERAN	PANDEGLA	86
TANGERAN	LEBAK	106
TANGERAN	BEKASI	54
TANGERAN	KARAWAN	96
TANGERAN	PURWAKA	138
TANGERAN	SUBANG	186
TANGERAN	BOGOR	83
TANGERAN	SUKABUM	144
TANGERAN	CIANJUR	147
TANGERAN	BANDUNG	212
TANGERAN	SUMEDAN	257
TANGERAN	GARUT	275
TANGERAN	TASIKMAL	318
TANGERAN	CIAMIS	333
TANGERAN	KUNINGAN	318
TANGERAN	MAJALENG	303
TANGERAN	CIREBON	283
TANGERAN	INDRAMAY	230

Sama seperti DFS yang sebelumnya, akan dicari lagi oleh greedy yaitu jarak mana yang paling pendek. Hanya saja, perbedaannya adalah, jika hanya greedy saja, maka akan diambil kota Jakarta dikarenakan jaraknya yang paling pendek diantara yang lain, namun, karena menggunakan DFS, maka Jakarta dianggap sudah dikunjungi(*visited)* sehingga tidak akan dikunjungi lagi. Oleh karena itu, kota yang akan dikunjungi selanjutnya adalah Bekasi dengan jarak 54 kilometer. Sehingga JarakYangDimiliki sisa adalah :

135-54 = 81

Karena JarakYangDimiliki masih lebih dari 0, maka akan dicari lagi kota mana yang bisa dikunjungi, dari titik awal Bekasi dan sisa JarakYangDimilliki = 81.

BEKASI	JAKARTA	29
BEKASI	SERANG	119
BEKASI	PANDEGLA	140
BEKASI	LEBAK	160
BEKASI	TANGERAN	54
BEKASI	KARAWAN	42
BEKASI	PURWAKA	84
BEKASI	SUBANG	132
BEKASI	BOGOR	87
BEKASI	SUKABUM	148
BEKASI	CIANJUR	151
BEKASI	BANDUNG	154
BEKASI	SUMEDAN	199
BEKASI	GARUT	217
BEKASI	TASIKMAL	260
BEKASI	CIAMIS	279
BEKASI	KUNINGAN	261
BEKASI	MAJALENG	239
BEKASI	CIREBON	229
BEKASI	INDRAMAY	176

Masih sama seperti sebelumnya, akan dicari lagi kota mana dengan jarak yang paling dekat dan belum dikunjungi, dan menghasilkan Karawang dengan jarak 42 kilometer. Sehingga, sisa JarakYangDimiliki adalah :

81-42 = 39

Karena JarakYangDimiliki masih lebih dari 0, maka akan dicari lagi kota mana yang bisa dikunjungi, dari titik awal Karawang dan sisa JarakYangDimilliki = 39.

KARAWAN	JAKARTA	71
KARAWAN	SERANG	161
KARAWAN	PANDEGLA	182
KARAWAN	LEBAK	202
KARAWAN	TANGERAN	96
KARAWAN	BEKASI	42
KARAWAN	PURWAKA	42
KARAWAN	SUBANG	90
KARAWAN	BOGOR	96
KARAWAN	SUKABUM	172
KARAWAN	CIANJUR	147
KARAWAN	BANDUNG	112
KARAWAN	SUMEDAN	157
KARAWAN	GARUT	175
KARAWAN	TASIKMAL	218
KARAWAN	CIAMIS	233
KARAWAN	KUNINGAN	219
KARAWAN	MAJALENG	203
KARAWAN	CIREBON	184
KARAWAN	INDRAMAY	134

Ketika dilakukan Greedy, maka sudah tidak ditemukan lagi jarak yang lebih kecil atau sama dengan JarakYangDimiliki, maka program akan melakukan backtracking ke kota-kota sebelumnya dan akan melakukan DFS lagi di kota-kota tersebut.

Salah satu contohnya adalah backtracking kembali ke Jakarta, kemudian akan pergi ke Bogor dengan jarak 58 kilometer

JAKARTA	SERANG	90
JAKARTA	PANDEGLA	111
JAKARTA	LEBAK	131
JAKARTA	TANGERAN	25
JAKARTA	BEKASI	29
JAKARTA	KARAWAN	71
JAKARTA	PURWAKA	113
JAKARTA	SUBANG	161
JAKARTA	BOGOR	58
JAKARTA	SUKABUM	119
JAKARTA	CIANJUR	122
JAKARTA	BANDUNG	187
JAKARTA	SUMEDAN	232
JAKARTA	GARUT	250
JAKARTA	TASIKMAL	293
JAKARTA	CIAMIS	308
JAKARTA	KUNINGAN	293
JAKARTA	MAJALENG	278
JAKARTA	CIREBON	258
JAKARTA	INDRAMAY	205

Sehingga JarakYangDimiliki adalah :

160-58 = 102

Karena JarakYangDimiliki masih lebih dari 0, maka akan dicari lagi kota mana yang bisa dikunjungi, dari titik awal Bogor dan sisa JarakYangDimiliki = 102

BOGOR	JAKARTA	58
BOGOR	SERANG	148
BOGOR	PANDEGLA	118
BOGOR	LEBAK	98
BOGOR	TANGERAN	83
BOGOR	BEKASI	87
BOGOR	KARAWAN	96
BOGOR	PURWAKA	163
BOGOR	SUBANG	186
BOGOR	SUKABUM	61
BOGOR	CIANJUR	74
BOGOR	BANDUNG	129
BOGOR	SUMEDAN	174
BOGOR	GARUT	192
BOGOR	TASIKMAL	235
BOGOR	CIAMIS	250
BOGOR	KUNINGAN	194
BOGOR	MAJALENG	220
BOGOR	CIREBON	259
BOGOR	INDRAMAY	313

Dan begitu seterusnya hingga semua kemungkinan yang bisa dicapai baik oleh algoritma Greedy, yaitu jarak antar kota kurang atau sama dengan JarakYangDimiliki, dan juga oleh Algoritma DFS, yaitu simpul yang dituju bukan simpul yang sudah pernah dituju.

B. Uji Coba dengan Program

Berdasarkan langkah-langkah yang telah dibuat sebelumnya, penulis membuat sebuah program dengan menggunakan bahasa Python dan basis data yang telah dibuat.

Untuk basis data, dibuat dengan menggunakan data antar kota yang telah tersedia banyak di internet, dan kemudian dimasukkan ke dalam file dengan format .csv dengan tiga kolom, yaitu kolom kota asal, kota tujuan, dan jarak antar kota. Kemudian program akan membaca file eksternal tersebut, dan merepresentasikannya dalam bentuk matriks.

Sedangkan untuk programnya itu sendiri, menggunakan struktur data *dictionary* supaya mudah untuk membuat *graf* yang tersusun dari *string* , bukan dari *integer*.

Fungsi utama yang akan digunakan, adalah gabungan dari Algoritma DFS dan Algoritma Greedy, dengan *pseudo-code* sebagai berikut :

```
FUNCTION
  solve(start,visited,mat,length,listjalan) :
    visited[start] <- True
    listjalan.append(start)
    for i in mat :
        IF (i[0] = start) :

  IF (length >= i[2] AND not(visited[i[1]])) :
                length <- length - i[2]

  solve(i[1],visited,mat,length,listjalan)

  listjalan.remove(listjalan[len(listjalan)-1])
        ENDIF
            ENDIF
    ENDFOR
    OUTPUT listjalan,": ",len(listjalan)
ENDFUNCTION
```

Fungsi tersebut akan memberikan keluaran berupa kota apa saja yang bisa dituju, serta jumlah dari kota yang bisa dituju.

Contoh penggunaannya adalah sebagai berikut :

```
C:\Users\Pandyaka\Desktop\Stima\Makalah>python main.py
JAKARTA
1
['JAKARTA', 'TANGERANG', 'BEKASI', 'KARAWANG'] :  4
['JAKARTA', 'TANGERANG', 'BEKASI'] :  3
['JAKARTA', 'TANGERANG', 'SERANG'] :  3
['JAKARTA', 'TANGERANG'] :  2
['JAKARTA', 'BOGOR', 'SUKABUMI'] :  3
['JAKARTA', 'BOGOR'] :  2
['JAKARTA'] :  1
```

Tangkapan layar diatas adalah contoh penggunaan program dengan masukan kota awal adalah JAKARTA dan pengguna hanya memiliki 1 hari saja untuk berlibur.

```
C:\Users\Pandyaka\Desktop\Stima\Makalah>python main.py
BANDUNG
2
['BANDUNG', 'SUMEDANG', 'MAJALENGKA', 'KUNINGAN', 'CIREBON', 'INDRAMAYU'] :  6
['BANDUNG', 'SUMEDANG', 'MAJALENGKA', 'KUNINGAN', 'CIREBON'] :  5
['BANDUNG', 'SUMEDANG', 'MAJALENGKA', 'KUNINGAN', 'CIAMIS', 'TASIKMALAYA', 'GARUT'] :  7
['BANDUNG', 'SUMEDANG', 'MAJALENGKA', 'KUNINGAN', 'CIAMIS', 'TASIKMALAYA'] :  6
['BANDUNG', 'SUMEDANG', 'MAJALENGKA', 'KUNINGAN', 'CIAMIS'] :  5
['BANDUNG', 'SUMEDANG', 'MAJALENGKA', 'KUNINGAN'] :  4
['BANDUNG', 'SUMEDANG', 'MAJALENGKA', 'SUBANG'] :  4
['BANDUNG', 'SUMEDANG', 'MAJALENGKA'] :  3
['BANDUNG', 'SUMEDANG', 'CIANJUR', 'SUKABUMI', 'BOGOR'] :  5
['BANDUNG', 'SUMEDANG', 'CIANJUR', 'SUKABUMI'] :  4
['BANDUNG', 'SUMEDANG', 'CIANJUR'] :  3
['BANDUNG', 'SUMEDANG', 'PURWAKARTA'] :  3
['BANDUNG', 'SUMEDANG'] :  2
```

Sedangkan untuk tangkapan layar yang ini, adalah contoh penggunaakn program dengan masukan kota awal BANDUNG dan pengguna memiliki 2 hari untuk berlibur.

Pada akhir penyelesaian, akan muncul solusi berupa kota apa saja yang bisa dikunjungi beserta jumlah dari kota-kota tersebut namun tidak secara urut dari terbesar ke kecil atau sebaliknya. Hasil akhir yang dikeluarkan bergantung pada bagaimana algoritma DFS melakukan searching, dan bagaimana bentuk *graf* yang terbentuk sesuai dengan masukan pengguna.

IV. KESIMPULAN

Dari bagian-bagian yang telah dijelaskan di atas, dapat disimpulkan bahwa algoritma *Depth-First Searching* dan algoritma *Greedy* dapat secara efektif digunakan untuk mencari berapa banyak kota yang bisa dikunjungi dalam waktu yang telah ditentukan. Namun, program ini juga sangat bergantung dengan data yang baik supaya bisa menghasilkan hasil yang baik pula.

UCAPAN TERIMA KASIH

Ucapan terimakasih penulis nyatakan kepada Tuhan Yang Maha Esa, karena karunia-Nya penulis bisa diberikan kesempatan untuk menyelesaikan dan bisa memberikan kontribusi nyata dalam memberikan ide yang dituliskan pada makalah ini.

Penulis juga mengucapkan banyak terimakasih kepada Agung Budi Cahyono S.T, M.Sc, DEA yang telah memberikan bantuan kepada penulis berupa data-data yang dibutuhkan dalam pengerjaan makalah ini.

Dan terakhir, penulis mengucapkan terimakasih kepada Dr. Rinaldi Munir, atas dedikasinya dalam memberikan ilmu pengetahuan yang sangat berharga kepada penulis.

REFERENSI

[1] Munir. Rinaldi, Diktat Kuliah IF2211 Strategi Algoritma, 2009, pp 111-114

[2] GeeksforGeeks, "Depth First Searching or DFS for a Graph", diakses dari https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/ dari pada tanggal 24 April 2019.

[3] GeeksforGeeks, "Greedy Algorithm", diakses dari https://www.geeksforgeeks.org/greedy-algorithms/ pada tanggal 24 April 2019.

[4] http://qa.geeksforgeeks.org/6653/what-is-the-history-of-depth-first-search-algorithm.html diakses pada tanggal 26 April 2019.

[5] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Makalah2018/Makalah-IF2211-2018-006.pdf diakses pada tanggal 26 April 2019

[6] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Makalah2018/Makalah-IF2211-2018-005.pdf diakses pada tanggal 26 April 2019

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, 26 April 2019

Pandyaka Aptanagi
13517003