TUGAS AKHIR - KI141502

IMPLEMENTASI APLIKASI PENCARIAN RUTE BERBASIS TOP-K SHORTEST PATH DENGAN KERAGAMAN UNTUK PERANGKAT BERGERAK

Bramastya Dewa Indraswara
NRP 05111540000130

Dosen Pembimbing
Bagus Jati Santoso, S.Kom, Ph.D.
Dr.Eng. Radityo Anggoro, S.Kom, M.Sc.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

[Halaman ini sengaja dikosongkan]

TUGAS AKHIR – KI141502

IMPLEMENTASI APLIKASI PENCARIAN RUTE BERBASIS TOP-K SHORTEST PATH DENGAN KERAGAMAN UNTUK PERANGKAT BERGERAK

Bramastya Dewa Indraswara
NRP 05111540000130

Dosen Pembimbing
Bagus Jati Santoso, S.Kom, Ph.D.
Dr.Eng. Radityo Anggoro, S.Kom, M.Sc.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2019

[Halaman ini sengaja dikosongkan]

FINAL PROJECT – KI141502

IMPLEMENTATION OF PATH FINDING APPLICATION BASED ON TOP-K SHORTEST PATH WITH DIVERSITY FOR MOBILE DEVICES

Bramastya Dewa Indraswara
NRP 05111540000130

Advisor
Bagus Jati Santoso, S.Kom, Ph.D.
Dr.Eng. Radityo Anggoro, S.Kom, M.Sc.

DEPARTMENT OF INFORMATICS
Faculty of Information Technology and Communication
Institut Teknologi Sepuluh Nopember
Surabaya 2019

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

Implementasi Aplikasi Pencarian Rute Berbasis Top-K Shortest Path Dengan Keragaman Untuk Perangkat Bergerak

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Berbasis Jaringan
Program Studi S-1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh :
Bramastya Dewa Indraswara
NRP : 05111540000130

Disetujui oleh Dosen Pembimbing Tugas Akhir :

BAGUS JATI SANTOSO, S.KOM, PH.D.
NIP: 198611252018031001 ..... (pembimbing 1)

DR.ENG RADITYO ANGGORO, S.KOM, M.SC.
NIP: 198410162008121002 ..... (pembimbing 2)

SURABAYA
JANUARI 2019

[Halaman ini sengaja dikosongkan]

IMPLEMENTASI APLIKASI PENCARIAN RUTE BERBASIS TOP-K SHORTEST PATH DENGAN KERAGAMAN UNTUK PERANGKAT BERGERAK

Nama Mahasiswa : Bramastya Dewa Indraswara
NRP : 05111540000130
Jurusan : Informatika FTIK-ITS
Dosen Pembimbing 1 : Bagus Jati Santoso, S.Kom, Ph.D
Dosen Pembimbing 2 : Dr.Eng. Radityo Anggoro, S.Kom, M.Sc.

ABSTRAK

Shortest path merupakan rute dengan jarak atau total bobot terkecil dari suatu *vertex* ke *vertex* lain pada suatu graf. Masalah *shortest path* sendiri merupakan masalah umum yang dapat ditemukan pada kehidupan sehari-hari. Baik dalam navigasi sampai kepada *routing* pada jaringan. Banyak sekali algoritma yang dapat digunakan untuk mencari *shortest path* pada sebuah graf. Setiap jenis graf memiliki algoritma yang optimal dalam menyelesaikan masalah ini.

Tujuan dari pembuatan tugas akhir ini adalah untuk mendesain algoritma yang dapat mencari *k shortest path* yang cukup berbeda antara satu dengan lainnya berdasarkan preferensi pengguna. Selain itu, algoritma ini dapat memberikan *shortest path* yang memenuhi kebutuhan dan keinginan pengguna karena adanya *tag-tag* yang dapat dimasukkan pengguna dalam pencarian *shortest path*. *Tag-tag* tersebut digunakan sebagai salah satu patokan dalam mengurutkan tingkat kesesuaian rute. Jadi, rute dengan jarak yang kecil, jika pengguna lebih menginginkan *tag*nya terpenuhi, bisa saja berada pada urutan yang rendah jika rute tersebut tidak memenuhi *tag* yang diberikan pengguna.

Kata kunci : ***Shortest Path***, ***Path Finding, Road Network***

[Halaman ini sengaja dikosongkan]

IMPLEMENTATION OF PATH FINDING APPLICATION BASED ON TOP-K SHORTEST PATH WITH DIVERSITY FOR MOBILE DEVICES

Student Name	: Bramastya Dewa Indraswara
Student ID	: 05111540000130
Major	: Department of Informatics Faculty of ICT-ITS
Advisor 1	: Bagus Jati Santoso, S.Kom, Ph.D
Advisor 2	: Dr.Eng. Radityo Anggoro, S.Kom, M.Sc.

ABSTRACT

Shortest path is a path with the shortest distance or total weight between two vertices within a graph. Shortest path problem itself is a general problem that can be found in daily life problem, Be it for navigation to network routing. There are many algorithms that have been implemented to find the shortest path within a graph, Each type of graph has its own algorithm that can be used to solved this problem optimally.

The goals of this research is to design algorithm that can find k shortest path which diverse from each other based on user preferences. In addition, this algorithm can provide a shortest path that meets the needs and desires of users because of the tags that users can enter in searching for the shortest path. These tags are used as one of the criterions in order to sort the level of suitability of the path. So, a path with a small distance, if the user wants more tags to be fulfilled, could be in a low order if the path does not meet the tag given by the user.

Keywords : ***Shortest Path, Path Finding, Road Network***

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Segala Puji syukur kepada Tuhan Yang Maha Esa yang telah melimahkan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan tugas akhir yang berjudul **"IMPLEMENTASI APLIKASI PENCARIAN RUTE BERBASIS TOP-K SHORTEST PATH DENGAN KERAGAMAN UNTUK PERANGKAT BERGERAK"**.

Tugas akhir ini dilakukan dengan tujuan agar penulis dapat menghasilkan sesuatu atau menerapkan apa yang telah dipelajari selama pada masa perkuliahan dan untuk memenuhi salah satu syarat memperoleh gelar Sarjana di Jurusan Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember Surabaya.

Terimakasih yang sebesar – besarnya saya ucapkan kepada berbagai pihak yang telah mendukung dan membantu saya dalam pengerjaan tugas akhir ini, antara lain :

1. Orang Tua dan keluarga saya yang selalu mendo'akan dan senantiasa memotivasi dalam menyelesaikan pengerjaan tugas akhir ini.
2. Bapak Bagus Jati Santoso, S.Kom, Ph.D. selaku dosen pembimbing I.
3. Bapak Dr.Eng Radityo Anggoro, S.Kom, M.Sc. selaku dosen pembimbing II.
4. Anggota GLBK yang selalu mendukung dan menyemangati penulis dalam pengerjaan tugas akhir ini.
5. Seluruh keluarga Mahasiswa Teknik Informatika angkatan 2015.
6. Dan seluruh pihak yang tidak bisa penulis sebutkan satu persatu yang telah membantu dan mendukung penulis dalam menyelesaikan tugas akhir ini.

Penulis menyadari bahwa masih terdapat banyak kekurangan dalam pengerjaan tugas akhir ini. Penulis sangat

mengharapkan kritik dan saran dari pembaca untuk perbaikan dan pembelajaran kedepannya. Semoga tugas akhir ini dapat dimanfaatkan dengan sebagai mestinya, dan bermanfaat bahkan setelah tugas akhir ini selesai untuk Almamater, masyarakat dan bangsa.

Surabaya, Januari 2019
Penulis

Bramastya Dewa Indraswara
05111540000130

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

DAFTAR KODE SUMBER

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan Tugas Akhir, dan sistematika penulisan.

1.1. Latar Belakang

Path Finding merupakan suatu masalah yang sangat penting dalam kehidupan kita sehari-hari. Meski banyak algoritma mengenai *path finding* telah diimplementasikan, kita masih mencari metode *path finding* lain yang lebih efisien dan berbeda. Contohnya adalah ketika kita berada di jalan, jika rute dengan jarak terpendek yang ditemukan melewati banyak lampu lalulintas, maka ada beberapa orang yang akan lebih memilih rute yang lebih jauh dengan lampu lalulintas yang lebih sedikit. Pengaplikasian hal tersebut menggunakan pencarian *top-k shortest path* yang digabungkan dengan fleksibilitas pengguna untuk memilih keputusan tersendiri.

Metode pencarian *top-k shortest path* biasa (KSP) [1] memberikan hasil berupa *shortest path* dari suatu lokasi asal menuju ke tujuan sebanyak *k* rute. Akan tetapi, dari *shortest path* yang dihasilkan biasanya memiliki tingkat keragaman yang cukup tinggi karena *edge* yang digunakan biasanya sama [2]. Keragaman ini kurang diinginkan karena mengurangi fleksibilitas pengguna untuk memilih rute. Maka dari itu, terbentuklah ide untuk mencari *top-k shortest path* dengan keragaman, *top-k shortest path with diversity* (KSPD), dimana tingkat keragaman antar rute harus di bawah batas tertentu (*threshold*) yang ditentukan oleh pengguna. KSPD sendiri memiliki tingkat kesulitan yang cukup tinggi dikarenakan adanya banyak rute dari suatu lokasi asal menuju ke tujuan tertentu.

Dalam menyelesaikan persoalan KSPD sendiri, dibuatlah sebuah kerangka kerja yang untuk mencari rute yang paling

menjanjikan untuk diselidiki. Rute yang menjanjikan ditentukan dari *lower-bound*-nya saat menggunakan fungsi similaritas tertentu dan *threshold* yang telah ditentukan pengguna. Kerangka kerja ini menggunakan *reverse shortest path tree* dari tujuan untuk menentukan *lower-bound* dari sebuah rute yang telah diselidiki sebagian. Setiap rute yang telah diselidiki sebagian nantinya juga akan diklasifikasikan ke dalam kelas-kelas tertentu untuk membantu memangkas kemungkinan rute yang sudah tidak memungkinkan untuk dijadikan calon *shortest path*. Maka dari itu, saat tidak ada *threshold* yang ditentukan, kerangka kerja ini juga mampu menyelesaikan masalah KSP.

Dalam proses pencarian rute, algoritma ini juga akan mengambil preferensi pengguna dalam bentuk *tag-tag* yang terdapat pada tiap lokasi. *Tag-tag* ini membantu menentukan rute mana yang paling memenuhi keinginan pengguna. Pengguna dapat menentukan preferensinya pada saat mencari rute, baik itu semakin pendek rute semakin baik, atau semakin banyak *tag* yang terpenuhi semakin baik. Dari data ini kemudian diurutkan dari rute yang paling sesuai dengan preferensi pengguna.

1.2. Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini ada sebagai berikut:

1. Bagaimana menganalisis dan menentukan parameter yang paling berpengaruh terhadap kinerja masalah *top-k shortest path* dengan keragaman untuk perangkat bergerak?
2. Bagaimana mengimplementasi desain algoritma untuk mencari *top-k shortest path* dengan keragaman untuk perangkat bergerak?

1.3. Batasan Masalah

Permasalahan yang dibahas pada Tugas Akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Implementasi algoritma menggunakan bahasa pemrograman Python dan Java.
2. Fokus dari implementasi algoritma ini adalah untuk perangkat bergerak (*mobile device*).
3. Jenis graf yang digunakan adalah *Bidirectional Weighted Graph.*
4. Algoritma yang digunakan dalam tugas akhir ini adalah algoritma Dijkstra untuk menentukan *shortest path* pertama.

1.4. Tujuan Penelitian

1. Menganalisis dan menentukan parameter yang paling mempengaruhi kinerja masalah *top-k shortest path* dengan keragaman.
2. Mengimplementasikan desain algoritma untuk mencari *top-k shortest path* dengan keragaman untuk perangkat bergerak.

1.5. Manfaat Penelitian

Manfaat yang diharapkan dari penulisan tugas akhir ini adalah untuk dapat membantu penemuan *top-k shortest path* dari suatu lokasi asal ke tujuan dengan keragaman yang diinginkan pada perangkat bergerak.

1.6. Metodologi

Langkah-langkah yang ditempuh dalam pengerjaan Tugas Akhir ini yaitu:

1.6.1. Penyusunan Proposal Tugas Akhir

Tahap pertama dalam penyusunan tugas akhir ini adalah penyusunan proposal tugas akhir. Proposal tugas akhir ini memberikan gambaran mengenai masalah yang ingin diselesaikan dan metode-metode yang akan digunakan dan metode baru yang diajukan.Studi Literatur

1.6.2. Studi Literatur

Pada tahap studi literatur akan dipelajari referensi-referensi yang berkaitan dengan tugas akhir. Referensi dapat diambil dari tugas akhir, internet, paper, jurnal, maupun konferensi.

1.6.3. Analisis dan Desain Perangkat Lunak

Sebelum memulai implementasi akan dianalisa mengenai permasalahan yang akan diselesaikan lalu dilanjutkan dengan pembuatan solusi yang meliputi penentuan struktur data dan algoritma baru dan dilanjutkan dengan implementasi.

1.6.4. Implementasi Perangkat Lunak

Tahap implementasi meliputi implementasi algoritma dan struktur data pada perangkat lunak yang telah didukung oleh hasil analisis dan desain pada tahap sebelumnya. Implementasi ini dilakukan dengan menggunakan python dan java.

1.6.5. Pengujian dan Evaluasi

Pengujian dalam algoritma ini berfokus pada keberhasilan dalam algoritma pengolahan *skyline query* pada dynamic uncertain data oleh titik tidak bergerak dan objek bergerak pada jaringan jalan raya.

1.7. Sistematika Penulisan

Buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini.

Bab I Pendahuluan

Bab ini berisi latar belakang masalah, tujuan dan manfaat pembuatan Tugas Akhir, permasalahan, batasan masalah,

metodologi yang digunakan, dan sistematika penyusunan Tugas Akhir.

Bab II Dasar Teori

Bab ini membahas beberapa teori penunjang yang berhubungan dengan pokok pembahasan dan mendasari pembuatan Tugas Akhir ini.

Bab III Analisis dan Perancangan Sistem

Bab ini membahas mengenai perancangan perangkat lunak. Perancangan perangkat lunak meliputi perancangan algoritma dan struktur data.

Bab IV Implementasi

Bab ini berisi implementasi dari perancangan perangkat lunak.

Bab V Pengujian dan Evaluasi

Bab ini membahas pengujian dengan metode pengujian objektif untuk mengetahui hasil dari implementasi terhadap metode lain.

Bab VI Kesimpulan dan Saran

Bab ini berisi kesimpulan dari hasil pengujian yang dilakukan. Bab ini membahas saran-saran untuk pengembangan sistem lebih lanjut.

Daftar Pustaka

Merupakan daftar referensi yang digunakan untuk mengembangkan Tugas Akhir.

Lampiran

Merupakan bab tambahan yang berisi daftar istilah yang penting pada sistem ini.

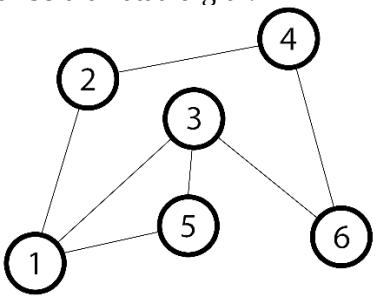
[Halaman ini sengaja dikosongkan]

BAB II
TINJAUAN PUSTAKA

Bab ini menjelaskan tentang tinjauan pustaka yang menjadi dasar pembuatan tugas akhir. Penjelasan secara khusus masing-masing tinjauan pustaka dapat dilihat pada masing-masing subbab berikut ini.

2.1. *Teori Graf*

Teori graf adalah ilmu yang mempelajari sifat-sifat grafik. Graf sendiri terdiri dari simpul (*vertex*) yang dihubungkan oleh sisi (*edge*). Persoalan graf dapat dikembangkan dengan memberi bobot pada tiap sisi. Bobot ini bisa melambangkan hal-hal berbeda, seperti panjang jalan pada graf tersebut. Sebuah graf dengan sisi yang berarah juga merupakan pengembangan dari persoalan graf. Graf seperti itu disebut graf berarah atau digraf.



Gambar 2.1 Representasi Teori Graf

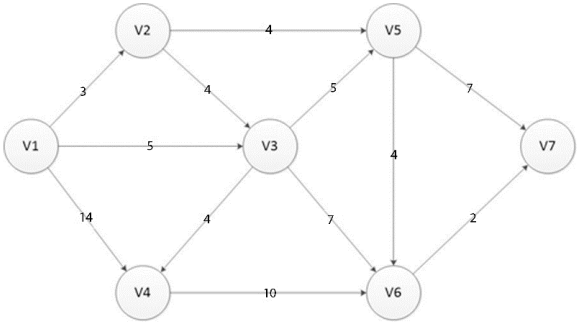
Graf dinotasikan sebagai $G=(V, E)$, dimana V berisi himpunan *vertex* dan E berisi himpunan *edge* pada graf tersebut. Sebagai contoh, graf diatas dinyatakan sebagai $G=(V, E)$ dimana $V=\{1,2,3,4,5,6\}$ dan $E=\{\{1,2\}, \{1,3\}, \{1,5\}, \{2,4\}, \{3,5\}, \{3,6\}, \{4,6\}\}$.

2.2. *Algoritma Dijkstra*

Algoritma Dijkstra adalah algoritma untuk mencari *shortest path* antar *vertex* pada sebuah digraf. Algoritma Dijkstra bekerja

dengan membuat jalur ke satu simpul optimal pada tiap langkahnya. Jadi pada langkah ke *n*, setidaknya ada n *vertex* yang kita ketahui jalur terpendeknya. Langkah-langkah algoritma Dijkstra dapat dilakukan sebagai berikut:

1. Tentukan titik mana yang akan menjadi *vertex* awal, lalu beri bobot jarak pada *vertex* pertama ke *vertex* terdekat satu per satu, Dijkstra akan melakukan pengembangan pencarian dari satu titik ke titik lain dan ke titik selanjutnya tahap demi tahap.
2. Beri nilai bobot (jarak) untuk setiap titik ke titik lainnya, lalu set nilai 0 pada *vertex* awal dan nilai tak hingga terhadap *vertex* lain (belum terisi)
3. Set semua *vertex* yang belum dilalui dan set *vertex* awal sebagai "*Vertex* keberangkatan"
4. Dari *vertex* keberangkatan, pertimbangkan *vertex* tetangga yang belum dilalui dan hitung jaraknya dari titik keberangkatan. Jika jarak ini lebih kecil dari jarak sebelumnya (yang telah terekam sebelumnya) hapus data lama, simpan ulang data jarak dengan jarak yang baru
5. Saat kita selesai mempertimbangkan setiap jarak terhadap *vertex* tetangga, tandai *vertex* yang telah dilalui sebagai "*Vertex* dilewati". *Vertex* yang dilewati tidak akan pernah di cek kembali, jarak yang disimpan adalah jarak terakhir dan yang paling minimal bobotnya.
6. Set "*Vertex* belum dilewati" dengan jarak terkecil (dari *vertex* keberangkatan) sebagai "*Vertex* Keberangkatan" selanjutnya dan ulangi langkah 5.



Gambar 2.2 Contoh Graf Representasi Algoritma Dijkstra

Tabel 2.1 Tahap Pengerjaan Representasi Algoritma Dijkstra

Iteration	Unvisited (Q)	Visited (S)	Current	*Vertex* : Min = (dist[*vertex*], prev[*vertex*])iteration						
				V1	V2	V3	V4	V5	V6	V7
	Initialization	{-}		(0,-)0	(∞,-)0	(∞,-)0	(∞,-)0	(∞,-)0	(∞,-)0	(∞,-)0
1	{V2,V3,V4,V5,V6,V7}	{V1}	{V1}		(3,V1)1	(5,V1)1	(14,V1)1	(∞,V1)1	(∞,V1)1	(∞,V1)1
2	{V3,V4,V5,V6,V7}	{V1,V2}	{V2}			(5,V1)1	(14,V1)1	(7,V2)2	(∞,V2)2	(∞,V2)2
3	{V4,V5,V6,V7}	{V1,V2,V3}	{V3}				(9,V3)3	(7,V2)2	(12,V3)3	(∞,V3)3
4	{V5,V6,V7}	{V1,V2,V3,V4}	{V4}					(7,V2)2	(12,V3)3	(∞,V3)3
5	{V6,V7}	{V1,V2,V3,V4,V5}	{V5}						(11,V5)5	(14,V5)5
6	{V7}	{V1,V2,V3,V4,V5,V6}	{V6|							(13,V6)6

Jadi *shortest path* untuk *weighted* digraf di atas adalah *V1>V2>V5>V6>V7*. [3]

2.3. *Python*

Python adalah sebuah *interpreter*, berbasis objek, dan bahasa pemrograman tingkat tinggi. Sebagai bahasa pemrograman tingkat tinggi, python juga didukung oleh struktur data tingkat tinggi pula. Python adalah bahasa pemrograman yang mudah dipelajari. Salah satu aspek yang ditekankan oleh python adalah

kemudahan sumber kode untuk dibaca. Python dapat digunakan untuk membuat *script* maupun menghubungkan antar komponen. Python mendukung modul dan paket yang dapat membuat sekumpulan kode untuk digunakan kembali. Python memiliki banyak *library* dan didistribusikan secara gratis. Sampai saat ini python sudah tersedia dalam 2 versi yaitu versi 2.x dan versi 3.x.

Python menawarkan produktivitas pada penggunanya. Karena python adalah *interpreter*, python tidak memakan biaya untuk kompilasi sehingga proses pengubahan, pengujian, dan *debug* menjadi lebih cepat. Melakukan *debug* pada python sangat mudah karena tidak akan mengakibatkan *segmentation fault* akan tetapi akan menimbulkan *exception* apabila terdapat kesalahan dalam penulisan kode. Ketika program tidak menangkap *exception*, maka python akan menampilkan *stack trace* yang dapat kita gunakan untuk menganalisa kesalahan yang terjadi. [4]

2.4. *Struktur Data*

Dalam istilah ilmu komputer, struktur data adalah cara penyimpanan, pengorganisasian, dan pengaturan data di dalam media penyimpanan komputer sehingga data tersebut dapat digunakan secara efisien.

Dalam teknik pemrograman, struktur data berarti tata letak data yang berisi kolom-kolom data, baik itu kolom yang tampak oleh pengguna ataupun kolom yang hanya digunakan untuk keperluan pemrograman yang tidak tampak oleh pengguna. Setiap baris dari kumpulan kolom-kolom tersebut dinamakan catatan (*record*). Lebar kolom untuk data dapat berubah dan bervariasi. Ada kolom yang lebarnya berubah secara dinamis sesuai masukan dari pengguna dan juga ada kolom yang lebarnya tetap. [5]

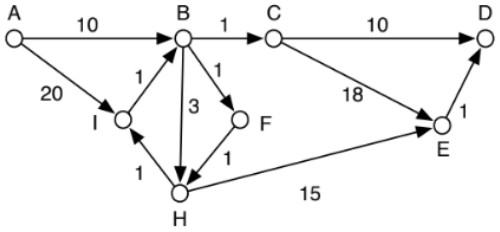
2.5. *Jaringan Client-Server*

Jaringan client-server didefinisikan sebagai suatu arsitektur jaringan komputer dimana perangkat client melakukan proses meminta data, dan server yang memiliki tugas untuk memberikan respon berupa data terhadap *request* tersebut.

Perangkat client biasanya berupa perangkat komputer dengan aplikasi *software* jaringan yang telah terinstal guna untuk meminta dan menerima data melalui jaringan. Salah satu contoh aplikasi *software* yang paling sering digunakan untuk meminta dan menerima data pada jaringan ialah *web browser*, dimana pengguna dapat melakukan request untuk sebuah halaman web, melalui aplikasi *web browser*. Perangkat lain yang dapat pula dikategorikan sebagai client ialah perangkat *mobile* seperti *smartphone* atau tablet. [6] [7]

2.6. *Top-k shortest path*

Top-k shortest path (KSP) bertujuan untuk menemukan *k shortest path* di graf berbobot. Rute akan dicari dengan menggunakan algoritma Dijkstra secara berulang-ulang sampai k shortest path ditemukan.



Gambar 2.3 Contoh Graf Representasi *Top-K shortest path*

Sebagai contoh, jika diminta 3 *shortest path* dari *A* ke *D* pada graf pada Gambar 2.3, maka akan didapat *A>B>C>D*, *A>B>F>H>E>D*, dan *A>B>H>E>D* dengan masing-masing rute memiliki jarak 21, 28, dan 29. [8] [9]

2.7. *Diversified Path Finding*

Diversified Path Finding merupakan ilmu yang digunakan untuk mencari rute-rute yang tidak serupa dengan menggunakan fungsi similaritas.

Sebagai contoh, jika 3 *shortest path* dari *A* ke *D* pada graf di Gambar 2.3 adalah *A>B>C>D*, *A>B>F>H>E>D*, dan *A>B>H>E>D*, karena rute *A>B>F>H>E>D* dan *A>B>H>E>D* memiliki tingkat keragaman yang cukup tinggi, maka meskipun rute *A>B>C>E>D* dengan jarak yang lebih jauh, dapat memiliki kesempatan untuk masuk dalam urutan *shortest path*. Akan tetapi, hal ini juga akan tergantung dari batasan similaritas yang digunakan. [10] [11] [12]

2.8. *Algoritma Min-Max*

Algoritma *Min-Max* adalah salah satu algoritma untuk melakukan normalisasi data. Normalisasi data adalah sebuah metode untuk mengatasi ketidakseimbangan nilai yang terdapat pada suatu himpunan data. Nilai-nilai yang tidak seimbang tersebut harus diubah ke dalam suatu rentang yang sama. Cara kerja algoritma ini adalah dengan melakukan penskalaan data pada rentang tertentu, dimana rentang yang digunakan umumnya adalah 0 hingga 1. Persamaan untuk algoritma ini adalah persamaan (2.1). [13]

$$X_n = \frac{X_0 - X_{min}}{X_{max} - X_{min}} \qquad (2.1)$$

dimana,

X_n = nilai baru untuk variabel *X*
X_0 = nilai awal untuk variabel *X*
X_{min} = nilai paling kecil pada himpunan data
X_{max} = nilai paling besar pada himpunan data

BAB III
ANALISIS DAN PERANCANGAN SISTEM

Bab ini menjelaskan tentang analisis dan perancangan mengenai sistem yang akan dibuat.

3.1. Analisis sistem

Analisis sistem terbagi menjadi 2 bagian, yaitu analisis permasalahan yang diangkat pada tugas akhir ini dan deskripsi umum sistem yang dibangun.

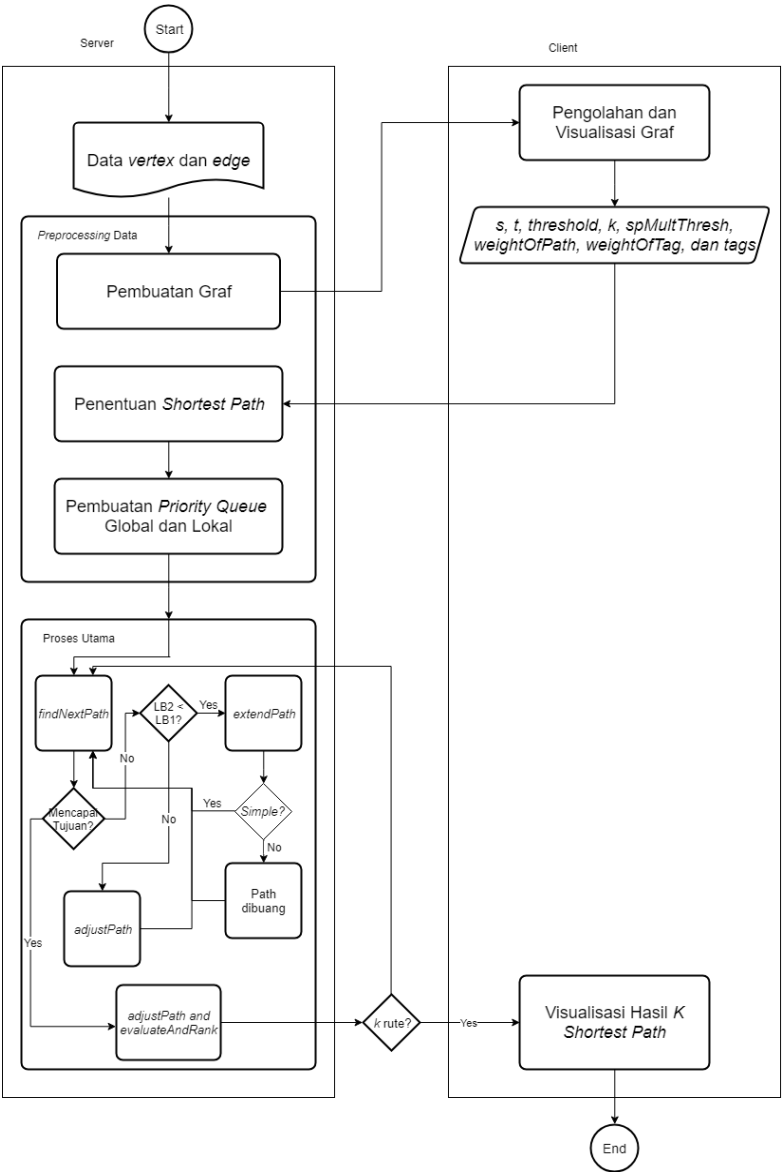
3.1.1. Analisis Permasalahan

Permasalahan yang diangkat pada tugas akhir ini adalah pencarian *k shortest path* pada suatu graf dengan tingkat similaritas yang berbeda. Tingkat similaritas ditentukan dari *threshold* yang diberikan pengguna. Rute yang diberikan selain dinilai dari jarak yang ditempuh, juga dari jumlah *tag* yang terpenuhi pada rute tersebut.

3.1.2. Deskripsi umum sistem

Fitur yang diimplementasikan pada tugas akhir ini terdiri dari 3 proses utama, yaitu, *preprocessing* data untuk membantu meringankan penghitungan pada proses utama, proses utama dimana terjadinya penghitungan *shortest path*, dan visualisasi kepada pengguna dengan mengirimkan data akhir kepada pengguna.

Agar lebih mudah dipahami, alur kerja sistem pada penelitian ini dapat dilihat pada diagram alur pada gambar 3.1



Gambar 3.1 Diagram Alur Deskripsi Umum Sistem

3.2. Perancangan Sistem

Perancangan sistem menjelaskan perancangan proses pada sistem, yaitu *preprocessing* data, proses utama, dan visualisasi hasil kepada pengguna.

3.2.1. *Preprocessing* Data

Preprocessing data adalah proses awal yang membantu meringankan beban proses utama. *Preprocessing* data pada sistem ini terdiri dari 3 tahapan utama, yaitu:

3.2.1.1. Pembuatan Graf

Proses pembuatan graf dilakukan dengan mengekstraksi data *vertex* dan *edge* yang tersedia. Masing-masing *edge* memiliki *node* yang mengandung *tag* untuk membantu pengguna menentukan tujuan yang diinginkan. Graf yang digunakan pada sistem ini adalah *Bidirectional Weighted Graph* dimana terdapat 2 *edge* berbalikan arah dengan bobot yang sama untuk 2 vertex yang berbeda.

3.2.1.2. Penentuan *Shortest Path*

Shortest path dari *vertex s* ke *vertex t* akan dijadikan patokan untuk mencari *shortest path* selanjutnya, karena *shortest path* selanjutnya merupakan perpanjangan dari tiap *vertex* yang ada pada *vertex-vertex* pada *shortest path* pertama. Dalam penentuan *shortest path* pertama, akan digunakan algoritma Dijkstra.

3.2.1.3. Pembuatan *Priority Queue* Global dan *Priority Queue* Lokal

Priority queue ini digunakan untuk menampung tiap rute yang sedang berada pada proses perpanjangan. *Priority queue* global *Q* awalnya berisi rute deviasi dari *shortest path* dari *vertex s* menuju ke *vertex t*. Tiap rute pada *Q* akan dikelompokkan ke dalam *priority queue*

lokal *LQ*(*v*) dimana *v* adalah *vertex* akhir atau *tail* pada rute tersebut. Setiap rute yang berada di *LQ*(*v*) diurutkan berdasarkan *lower-bound*-nya. *Priority queue* global *Q* yang berisi *priority queue* lokal diurutkan berdasarkan *lower-bound*-nya juga. Dengan begitu, rute pertama yang akan dikomputasi merupakan rute dengan *lower-bound* paling minimal dari semua *priority queue* lokal. Selain *lower-bound*, *LQ*(*v*) juga berisi data-data rute seperti panjang rute *P.length*, detil rute *P.rt*, dan klasifikasi rute *P.cls*.

3.2.2. Proses Utama

Setiap rute P pada *priority queue* yang memiliki *lower-bound* terkecil, selama belum mencapai tujuan *t*, maka rute tersebut akan diperpanjang, jika P setelah diperpanjang memiliki *loop* maka rute tersebut dibuang. Jika P sudah mencapai *t*, maka P dimasukkan ke dalam set hasil akhir. Proses ini berjalan hingga ditemukan *k shortest path*.

Untuk mempercepat proses komputasi ini, dibuatlah 2 *lower-bound*, yaitu *shortest path lower bound* dan *diversified path lower bound*. Untuk tiap rute P yang telah diselidiki sebagian, kita hitung *shortest path lower bound*-nya agar kita dapat mengetahui jarak minimal yang dapat dibuat dari P. Selain itu kita juga hitung *diversified path lower bound*-nya berdasarkan fungsi similaritas dan *threshold* yang dipilih. Semakin tinggi *diversified path lower bound* dari suatu rute, maka dapat dipastikan bahwa rute tersebut memiliki banyak *overlap* dengan rute yang ada di set hasil akhir, sehingga rute-rute ini nantinya akan ditunda dulu proses komputasinya.

3.2.2.1. *Shortest Path Lower Bound*

Shortest path lower bound (LB_1) dapat dicari dengan bantuan *reverse shortest path tree*. Untuk setiap rute *P*(*s*, *v*), dimana *P* adalah rute yang sedang diperpanjang, *s*

adalah *vertex* asal dan *v* adalah *vertex* terakhir pada rute tersebut, LB_1 (*P*(*s*, *v*)) adalah *L*(*s*, *v*) + *v*.dis, dimana *L*(*s*, *v*) adalah panjang rute dari *s* ke *v* dan *v*.dis adalah *reverse shortest path* dari *vertex v* menuju tujuan.

3.2.2.1.1. *Reverse Shortest Path Tree*

Reverse Shortest Path Tree digunakan untuk membantu mencari *shortest path lower bound* (LB_1) pada proses utama. Misalkan pada sebuah graf *G*(*V*, *E*) dan sebuah tujuan *t*, maka *reverse shortest path tree* dari *t*, *G*(*t*) adalah *G*(*V'*, *E'*) dimana *V'* berisi *t* dan *vertex-vertex* yang dapat dicapai melalui rute dari *t*, dan *E'* berisi *edge-edge* pada rute terpendek dari *vertex-vertex* pada *V'* menuju *t*. Maka, setiap *vertex u* pada *V'* hanya memiliki 1 *shortest path* menuju *t* dengan melewati *u*' sebagai *edge* keluarnya. Kita dapat menyebut *u*' *parent* dari *u* (*u.parent*), *u*.sp sebagai *shortest path* dari *u* menuju ke *t*, dan *u*.dis sebagai panjang dari *u*.sp.

3.2.2.1.2. Klasifikasi Rute

Tidak semua rute perlu diperpanjang meski rute tersebut memiliki *lower-bound* yang minimal. Ketika suatu rute dengan *lower-bound* minimal diperpanjang menjadi *cyclic* (memiliki *loop*), maka rute itu dianggap tidak sederhana dan tidak akan diperpanjang lebih lanjut. Untuk menghindari pemanjangan rute yang tidak diperlukan, dibuatlah konsep klasifikasi rute. Setiap rute yang diturunkan dari sebuah rute lainnya dimasukkan ke kelasnya masing-masing. Tiap kelas dinotasikan dengan *p.v* dimana *p* adalah rute yang diturunkan, dan *v* adalah *vertex* yang menjadi titik deviasi dari rute

p. Dengan kata lain, sebuah rute yang diturunkan dari rute *p* dengan titik deviasi *v* dimasukkan ke dalam kelas yang sama. Misalkan sebuah rute P_1 : *A*>*B*>*C*>*D* diturunkan menjadi rute *A*>*B*>*C* dan *A*>*B*>*E*>*C*, maka kedua rute tersebut dapat dimasukkan ke dalam kelas $1.B$. Selanjutnya, kita hanya perlu memperpanjang rute dengan *lower-bound* terkecil dari masing-masing kelas, sedangkan rute lainnya bisa kita non-aktifkan untuk sementara. Setelah *shortest path* P_n selanjutnya ditemukan, barulah semua rute non-aktif yang terdominasi oleh P_n tadi diaktifkan kembali.

3.2.2.2. *Diversified Path Lower Bound*

Untuk membedakan keragaman tiap rute, dibuatlah *diversified path lower bound* yang dinotasikan dengan LB_2. Terdapat 5 fungsi yang bisa membantu menentukan keragaman antar rute, yaitu

i. $$LB_2(\mathrm{P}) = \underset{P\prime\in\Psi}{max}\left\{\mathrm{L}\left(S_p \cap S_p{}'\right) \mathrm{X}\left(1+\frac{1}{\tau}\right) - \mathrm{L}(P')\right\} \quad (3.1)$$

ii. $$LB_2(\mathrm{P}) = \underset{P\prime\in\Psi}{max}\begin{cases}\frac{L(S_p\cap S_{p\prime})L(P\prime)}{2\tau\mathrm{L}(P')-L(S_p\cap S_{p\prime})} & 2\tau\mathrm{L}(P')> L(S_p\cap S_{p\prime}) \\ \infty & 2\tau\mathrm{L}(P') \le L(S_p\cap S_{p\prime})\end{cases} \quad (3.2)$$

iii. $$LB_2(\mathrm{P}) = \underset{P\prime\in\Psi}{max}\left\{\frac{\mathrm{L}(S_p\cap S_p{}')^2}{\tau^2 L(P\prime)}\right\} \quad (3.3)$$

iv. $$LB_2(\mathrm{P}) = \underset{P\prime\in\Psi}{max}\begin{cases}\frac{L(S_p\cap S_{p\prime})}{\tau} & L(P) \ge \mathrm{L}(P') \\ L(P) & L(P) < \mathrm{L}(P')\end{cases} \quad (3.4)$$

v. $$LB_2(\mathrm{P}) = \underset{P\prime\in\Psi}{max}\begin{cases}\infty & \mathrm{L}(S_p\cap S_p{}') \ge \tau\mathrm{L}(P') \\ L(P) & \mathrm{L}(S_p\cap S_p{}') < \tau\mathrm{L}(P')\end{cases} \quad (3.5)$$

Rute dengan tingkat keragaman *vertex* yang tinggi dengan rute yang ada di queue hasil akan memiliki LB_2 yang tinggi, dimana rute tersebut akan ditunda dulu proses pemanjangannya. Hal tersebut akan meningkatkan efisiensi dalam menemukan *shortest path* yang beragam.

3.2.2.3. Algoritma Akhir

Berdasarkan 2 *lower-bound* dan klasifikasi rute pada subbab sebelumnya, dibuatlah kerangka kerja akhir ini. Awalnya, dibuatlah set hasil akhir *finalpath* kosong yang nantinya akan berisi rute-rute dengan *ranking* terbaik.

3.2.2.3.1. *findNextPath*

Algoritma *findNextPath* akan memperpanjang rute yang telah diselidiki sebagian pada dengan *lower-bound* terkecil hingga mencapai tujuan, lalu mengembalikan rute itu sebagai kandidat rute terbaik. Sebelum memperpanjang rute P, *diversified path lower bound* (LB_2) P dihitung terlebih dahulu. Jika LB_2 lebih kecil dari pada LB_1-nya, P dapat diperpanjang melalui *tail*-nya dengan memanggil algoritma *extendPath* hingga *P* mencapai tujuan atau *P* tidak sederhana. Jika *P* mencapai tujuan, rute-rute yang terdominasi oleh *P* diaktifkan kembali, serta mengubah klasifikasi rute deviasi dari *P* dengan memanggil algoritma *adjustPath.* Tetapi, jika LB_2 lebih besar dari LB_1, nilai LB_2 akan dimasukkan ke dalam LB_1, dan rute akan dikembalikan ke *queue* untuk ditunda dulu pemanjangannya. Selain itu, rute yang sedang tidak aktif harus diaktifkan kembali, karena rute-rute tersebut bisa saja memiliki *lower-bound* yang minimal. Kemudian, rute dengan *lower*-bound terkecil harus dipilih ulang.

3.2.2.3.2. *extendPath*

Algoritma *extendPath* akan memperpanjang *P* pada *tail*-nya (*y*) melalui *y.parent* dan membentuk rute sederhana baru melalui *vertex* di sekitar *y*. Untuk setiap rute lain *P'* yang

berada pada *LQ*(*y*), jika rute tersebut memiliki klasifikasi yang sama, *P'* akan dinonaktifkan ketika *P'* memiliki panjang yang lebih besar dibandingkan *P*. Untuk setiap *edge* (*y, u*), jika *u* tidak ada dalam set rute dan *u* bukan *y.parent*, maka dapat dibuat rute baru *P''* yang diturunkan dari *P*. Jika *u* sudah terdominasi oleh klasifikasi dari *P*, maka kita nonaktifkan *P''*. Rute akan berhenti diperpanjang apabila terdapat *loop* pada proses pemanjangan pada *y.parent*. Jika tidak, maka rute akan diperpanjang lebih lanjut.

3.2.2.3.3. *adjustPath*

Algoritma *adjustPath* akan menyesuaikan rute *P* dengan mengaktifkan kembali rute-rute yang terdominasi oleh *P*. Jika *P* mencapai tujuan, setiap rute yang merupakan rute deviasi dari *P* akan diubah klasifikasinya menjadi *p.v*, dimana *p* adalah *P*.rt, dan *v* adalah *vertex* deviasi dari *P*.

3.2.2.3.4. *evaluateAndRank*

Setelah algoritma *findNextPath* menghasilkan potensi *shortest path*, *P*, rute tersebut harus dicek keragamannya dengan membandingkannya dengan rute *shortest path* sudah ada pada set akhir menggunakan fungsi similaritas. Jika *P* lolos uji similaritas, maka *P* akan dimasukkan ke dalam *finalpath* diurutkan berdasarkan tingkat terpenuhinya preferensi pengguna menggunakan algoritma *evaluateAndRank*. Saat mencari rute, selama jumlah rute di *finalpath* belum mencapai *k*, rute yang sudah selesai bisa langsung dimasukkan ke dalam *finalpath*. Jika jumlah rute sudah mencapai *k*, maka untuk rute selanjutnya, harus

ada asumsi bahwa setiap *tag* yang diminta pengguna terpenuhi. Jika dengan asumsi tersebut rute *P* bisa masuk dalam *ranking* rute terbaik, maka nilai *evaluateAndRank* sebenarnya akan dihitung dan rute akan dimasukkan ke dalam *finalpath* jika rankScale termasuk *k* rute terbaik. Tetapi jika tidak, proses penghitungan rute akan selesai dengan *finalpath* sebagai set rute terbaik sesuai dengan bobot jarak dan *tag* keinginan pengguna.

Algoritma *evaluateAndRank* dapat dicari dengan persamaan (3.6).

$$evaluateAndRank = \frac{spThreshold - x}{spThreshold - X_{min}} * WOP + \frac{TF}{total} * WOT \quad (3.6)$$

dimana,

X	= Panjang rute *P*
X_{min}	= Panjang *shortest path* dari *s* ke *t*
spThreshold	= *spMultThresh* * X_{min}
spMultThresh	= Batas kelipatan X_{min} yang dapat dievaluasi
TF	= Jumlah *tag* yang terpenuhi pada rute *P*
total	= Jumlah *tag* yang diminta pengguna
WOP	= *weightOfPath* atau bobot jarak
WOT	= *weightOfTag* atau bobot *tag*

3.2.3. Visualisasi Perangkat Bergerak

Untuk menampilkan hasil penelitian ini, digunakan aplikasi pada perangkat bergerak melalui arsitektur jaringan *client-server*. Pengguna memasukkan data alamat IP server agar dapat terhubung melalui soket. Setelah terhubung, server akan mengirimkan data graf kepada pengguna untuk digambarkan visualisasi grafnya. Pengguna kemudian dapat mengisi data lokasi awal, lokasi tujuan, *threshold*, jumlah rute, dan *spMultThresh* yang diinginkan. Pengguna juga dapat memberikan *tag* lokasi beserta bobot untuk penilaian *ranking* yang diinginkan secara opsional. Kemudian data dikirim ke server dan *shortest path* akan dikomputasi. Data *shortest path* kemudian akan dikembalikan ke pengguna dan diolah untuk ditampilkan ke antarmuka pengguna.

BAB IV
IMPLEMENTASI

Bab ini akan menjelaskan tentang implementasi Tugas Akhir berdasarkan rancangan perangkat lunak. Proses implementasi mengacu pada rancangan perangkat yang telah dilakukan sebelumnya, namun juga dimungkinkan terjadinya perubahan-perubahan jika dirasa perlu. Implementasi dilakukan dalam bahasa Python untuk sisi server dan Java untuk sisi pengguna

.

4.1. Lingkungan Implementasi Perangkat Lunak

Lingkungan implementasi adalah lingkungan di mana fitur temu kembali informasi dibangun. Lingkungan implementasi dibagi menjadi dua yaitu perangkat keras dan perangkat lunak.

4.1.1. Perangkat Keras

Lingkungan implementasi perangkat keras dari tugas akhir ini adalah sebagai berikut:

Tabel 4.1 Tabel Perangkat Keras Implementasi Sistem

Tipe	Lenovo G40-70
Prosesor	Intel® Core ™ i7-4510U CPU @ 2.00GHz 2.60GHz
Memori (RAM)	4.00 GB

4.1.2. Perangkat Lunak

Lingkungan implementasi perangkat lunak dari tugas akhir ini adalah sebagai berikut:

Tabel 4.2 Tabel Perangkat Lunak Implementasi Sistem

Sistem Operasi	Windows 10 Education 64-bit (build 17134)
Bahasa Pemrograman	Python 2.7.14, Java

Text Editor	Visual Studio Code 1.30.2

4.2. Implementasi Program Preprocessing data

Subbab ini akan menjelaskan mengenai implementasi algoritma pembuatan graf, penentuan *shortest path* dan terpanjang, dan penentuan *reverse shortest path tree*.

4.2.1. Implementasi Algoritma Pembuatan Graf

Algoritma 1 : *GenerateGraph*()
Output : *dataVertex, dataEdge*, graph *G*

```
1   open "VertexData.csv"
2   while not EOF("VertexData.csv") do
3      dataVertex.insert(Line)
4   close "VertexData.csv"
5
6   open "EdgeData.csv"
7   while not EOF("EdgeData.csv") do
8      dataEdge.insert(Line)
9   close "EdgeData.csv"
10
11  foreach edge in dataEdge do
12     if edge.vertex1 = edge.vertex2 then
13        G[edge.vertex1].insert((0, edge.vertex2))
14     else
15        G[edge.vertex1].insert((edge.cost, edge.vertex2))
16
17  return (dataVertex, dataEdge, G)
```

Kode Sumber 4.1 Kode Sumber Algoritma Pembuatan Graf

Graf akan dibentuk dari data-data *vertex* dan *edge* yang telah di*generate* secara acak pada Lampiran 0.1. Data *vertex* dan *edge* yang telah di*generate* akan dimasukkan ke dalam sebuah file csv yang nantinya akan diektraksi untuk proses pembuatan graf.

4.2.2. Implementasi Algoritma Penentuan *Shortest path*

Algoritma 2	:	*ShortestPathFunc*()
Input	:	Graph *G*, Source *s*, destination *t*.
Output	:	Shortest path from *s* to *t*.

```
1   create array set visited;
2   create priority queue SPQ;
3   SPQ.insert(0, s, []);
4   while SPQ is not empty do
5      P ← SPQ.extractMin();
6      if P.node is not in visited then
7         visited.insert(P.node);
8         P.path ← P.rt + P.node;
9         if P.node = t then
10           return (P.cost, P.rt);
11        foreach edge in G[node] do
12           if edge.neighbour is not in visited then
13              Q.insert(P.cost+edge.cost, edge.neighbour, P.rt);
14  return "inf";
```

Kode Sumber 4.2 Kode Sumber Algoritma Penentuan *Shortest path*

Algoritma ini menggunakan data graf untuk menentukan semua *shortest path* dan terpanjang dari tiap *vertex* ke *vertex* lainnya. Jika tidak ada *shortest path* dan terpanjang yang valid, maka akan didapat nilai "*inf*".

4.2.3. Implementasi Algoritma Pembuatan *Priority Queue* Global dan Priority *Queue* Lokal

Algoritma *inputLQ* akan mencari rute cabang dari *shortest path* pertama. Dari setiap rute cabang akan dicari LB_1-nya agar menjadi patokan dalam mengambil rute untuk diperpanjang. Rute yang akan diperpanjang adalah rute dengan LB_1 terkecil.

Algoritma 3	:	*InputLQ*()
Input	:	Graph *G*, Source *s*, destination *t*, shortest path *SP*.
Output	:	Shortest path from *s* to *t*.

```
1  Create priority queue Q and local priority queue LQ[.];
2  foreach path P' : v1, v2, →…→vi, vi+1 that deviates from SP, set P'.rt ← P', P'.cost ← L(P'), P'.lb ← LB1(P'), P'.cls ← SP.rt . vi, and LQ[vi+1].insert(P'), Q.insert(LQ[vi+1]);
```

Kode Sumber 4.3 Algoritma Pembuatan *Priority Queue* Global dan *Priority Queue* Lokal

4.3. Implementasi Proses Utama

4.3.1. Implementasi Algoritma Penentuan *Reverse Shortest Path Tree*

Algoritma *constructPartialSPT* akan mengambil data *shortest path* untuk dijadikan *reverse shortest path tree* dari *vertex i* (semua *vertex* pada graf) menuju ke *vertex* tujuan.

Algoritma 4	:	*ConstructPartialSPT*()
Input	:	Reverse Shortest Path Tree *RT*, graph *G*, destination *t*, vertex *v*.
Output	:	*v*.dis

```
1  if RT[v] ≠ -1 then
2     return RT[v]
3  else
4     RT[v] ← shortestPathFunc(G, v, t)
5     return RT[v]
```

Kode Sumber 4.4 Algoritma Penentuan *Reverse Shortest Path Tree*

4.3.2. Implementasi Algoritma *findNextPath*

Algoritma 5	:	*FindNextPath*()
Input	:	Graph *G*, destination *t*, threshold *threshold*, priority queue *Q*.

Output	:	The next shortest path

```
1   while Q is not empty do
2     LQ[v] ← Q.extractMin(), P ← LQ[v].extractMin()
3     if LQ[v] is not empty then Q.insert(LQ[v])
4     while P.tail ≠ t do
5       compute LB2(P)
6       if LB2(P) > P.lb then
7         P.lb ← LB2(P)
8         AdjustPath(P)
9         LQ[P.tail].insert(P)
10        Q.insert(LQ[P.tail]) if LQ[P.tail] is not in Q
11        break
12      else if not ExtendPath(P) then
13        break
14
15    if P.tail = t then
16      AdjustPath(P)
17      return P
```

Kode Sumber 4.5 Kode Sumber Algoritma *findNextPath*

4.3.3. Implementasi Algoritma *extendPath*

Algoritma 6	:	*ExtendPath*()
Input	:	Path *P* to be extended.
Output	:	**true/false**

```
1   y ← P.tail
2   foreach P′ in LQ[y], P′.cls = P.cls and P′.cost ≥ P.cost do
3     Mark P′ inactive
4   foreach edge(y, u), u not in P.rt and u ≠ y.parent do
5     create a new path P″
6     P″.rt ← P.rt + u
7     P″.cost ← P.cost + weight(y, u)
8     P″.lb ← LB1(P″)
9     P″.cls ← P.cls
10    if u has been covered by P.cls then
11      mark P″ inactive
12    else
13      LQ[u].insert(P″)
14      if LQ[u] is not in Q then Q.insert(LQ[u])
15  if y.parent is in P.rt then
```

```
16    return false
17  else
18    P.rt ← P.rt + y.parent
19    P.cost ← P.cost + weight(y, y.parent)
20    return true
```

Kode Sumber 4.6 Kode Sumber Algoritma *extendPath*

4.3.4. Implementasi Algoritma *adjustPath*

Algoritma 7	:	*AdjustPath*()
Input	:	Path *P* to be adjusted.
Output	:	**None**

```
1   u ← the deviation vertex of P
2   y ← P.tail
3
4   foreach vertex v, v in P(u, y) do
5     Activate the inactive paths in LQ[v] that are dominated by P, and if P
      reaches the destination, find all the pahs with prefix P(s, v) in all local
      queues, change their classifications to P.v
```

Kode Sumber 4.7 Kode sumber Algoritma *adjustPath*

4.3.5. Implementasi Pengurutan *Ranking* Rute

Algoritma 8	:	*EvaluateAndRank*()
Input	:	Path *P* to be evaluated, first shortest path *SP*, shortest path upper bound *spThreshold*.
Output	:	Ranking value of *P*, fulfilled tags in *P*

```
1   Search which tags are fulfilled by P
2
3   tag ← fulfilledTag.len/requestedTag.len
4   if P.len > spThreshold then
5     weight ← 0
6   else
7     weight ← (spThreshold - P.cost) / (spThreshold - SP.cost)
8
9   return (tag*weightOfTag/100+weight*weightOfPath/100, fulfilledTag)
```

Kode Sumber 4.8 Pengurutan *Ranking* Rute dengan Algoritma *evaluateAndRank*

BAB V
PENGUJIAN DAN EVALUASI

Bab ini membahas pengujian dan evaluasi pada sistem yang dikembangkan. Hasil yang didapatkan dari tahap pengujian akan dilakukan evaluasi sehingga dapat ditarik kesimpulan untuk bab selanjutnya.

5.1. Lingkungan Pengujian Sistem

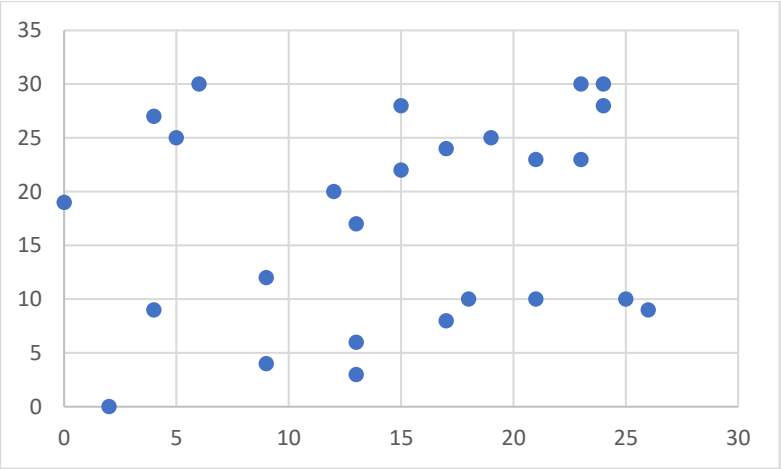
Lingkungan pengujian sistem pada pengerjaan Tugas Akhir ini dilakukan pada lingkungan dan alat kakas seperti yang tertera:

Tabel 5.1 Tabel Lingkungan Pengujian Sistem

Spesifikasi	Deskripsi
CPU	Intel® Xeon ® CPU E5-2650 v4 @ 2.20GHz
Sistem Operasi	Ubuntu 16.04.5 LTS
Memori Primer	16.00 GB

5.2. Jenis Data Pengujian

Pada pengujian ini akan digunakan data berjenis buatan yang bersifat independen. Pada dataset ini, *vertex* dan *edge* pada graf akan diacak dalam rentang yang telah ditentukan. Selain itu juga digunakan data acak sebagai input pengguna. Adapun data independen dapat digambarkan melalui gambar 5.1



Gambar 5.1 Representasi data Independen

5.3. Parameter Pengujian

Pengujian dilakukan terhadap beberapa parameter yaitu jumlah vertex(n), jumlah edge(m), jumlah *k*, jumlah tag(tags), nilai *threshold*, nilai *spMultThresh*, dan nilai *weighOfPath*. Secara *default*, adapun detail parameter pengujian seperti pada tabel 5.2.

Tabel 5.2 Tabel parameter pengujian

Parameter	Nilai
Jumlah *vertex*(*n*)	1.000
Jumlah *k*	5
Jumlah *tag*(*tags*)	5
Threshold	0.5
spMultThresh	3
weightOfPath	40% atau 0.4

5.4. Skenario Pengujian

Pengujian dilakukan sebanyak 10 kali untuk tiap skenario menggunakan data *vertex*, dan *edge* sintetis yang dibuat dengan data acak. Data *edge* berisi data 2 *vertex*, *weight edge* tersebut, dan *tag* yang akan digunakan untuk menghitung *ranking* dari tiap *shortest path* yang ditemukan.

Tabel 5.3 Graf Pengujian

Jumlah *Vertex*(*n*)	Jumlah *Edge*(*d*)
500	25.520
1.000	54.366
5.000	475.440
10.000	935.684

Selain dataset pada parameter pengujian, dibuatlah juga dataset acak sebagai pengganti input pengguna dimana dataset terdiri dari variabel *s*, dan *t*. Variabel *s* dan *t* merupakan asal dan tujuan yang nilainya akan diacak sesuai dengan data *vertex* dan *edge* yang ada.

Adapun skenario pengujian menggunakan variasi dari berbagai parameter seperti *n, k, threshold, spMultThresh, tags,* dan *weightOfPath*. Berikut adalah skenario pengujian yang dilakukan:

1. Variasi jumlah *vertex*(*n*) : 500, 1000, 5000, 10000. Untuk lebih jelasnya dapat dilihat pada Tabel 5.4.

Tabel 5.4 Tabel Skenario Variasi nilai n

Skenario	Jumlah *Vertex*
A01	500
A02	1.000
A03	5.000
A04	10.000

2. Variasi jumlah *k* : 2, 5, 8, 10, 12. Untuk lebih jelasnya dapat dilihat pada Tabel 5.5.

Tabel 5.5 Tabel Skenario Variasi nilai k

Skenario	*k*
B01	2
B02	5
B03	8
B04	10
B05	12

3. Variasi *threshold* : 0.25, 0.50, 0.75. Untuk lebih jelasnya dapat dilihat pada Tabel 5.6.

Tabel 5.6 Tabel Skenario Variasi nilai threhsold

Skenario	*Threshold*
C01	0.25
C02	0.50
C03	0.75

4. Variasi *spMultThresh* : 2, 3, 5, 7. Untuk lebih jelasnya dapat dilihat pada Tabel 5.7.

Tabel 5.7 Tabel Skenario Variasi nilai spMultThresh

Skenario	*spMultThresh*
D01	2
D02	3
D03	5
D04	7

5. Variasi jumlah *tag* : 2, 3, 5, 7. Untuk lebih jelasnya dapat dilihat pada Tabel 5.8.

Tabel 5.8 Tabel Skenario Variasi jumlah *tag*

Skenario	Jumlah Tag
E01	2
E02	3
E03	4
E04	5
E05	6

6. Variasi bobot *tag* dan jarak untuk menentukan *ranking* rute : 0.2, 0.4, 0.6, 0.8. Untuk lebih jelasnya dapat dilihat pada Tabel 5.9.

Tabel 5.9 Tabel Skenario Variasi nilai *weightOfPath*

Skenario	*weightOfPath*	*weightOfTag*
F01	0.2	0.8
F02	0.4	0.6
F03	0.6	0.4
F04	0.8	0.2

5.5. *Hasil Pengujian*

Pada bagian ini akan ditampilkan hasil uji coba performa. Performa yang dilihat adalah waktu total yang dibutuhkan untuk menjalankan *query* dan besar jumlah memori yang dipakai.

5.5.1. *Preprocessing* Data

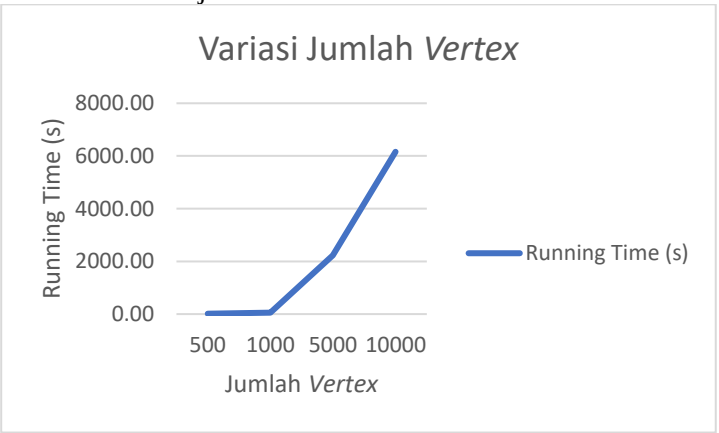
Tabel 5.10 Tabel Rata-rata Waktu Preprocessing

Jumlah *Vertex*(*n*)	Waktu *Preprocessing* (detik)
500	2,49
1.000	18,38
5.000	268,66
10.000	325,11

Tabel 5.10 menampilkan rata-rata waktu preprocessing untuk variasi jumlah vertex. Dari tabel tersebut dapat disimpulkan bahwa semakin besar jumlah vertex dan edge, maka akan semakin besar waktu *preprocessing* yang dibutuhkan.

5.5.2. Performa *KSPD*

1. Skenario variasi jumlah *vertex*:

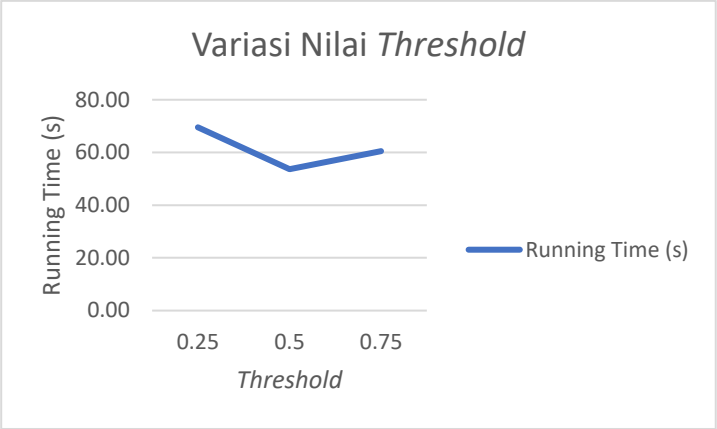


Gambar 5.2 Hasil Pengujian Variasi Jumlah *Vertex*

2. Skenario variasi *k*:

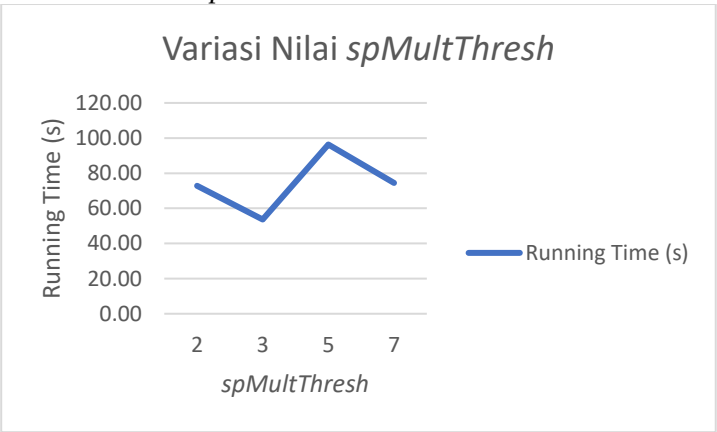
Gambar 5.3 Hasil Pengujian Variasi Jumlah *K*

3. Skenario variasi *threshold*:



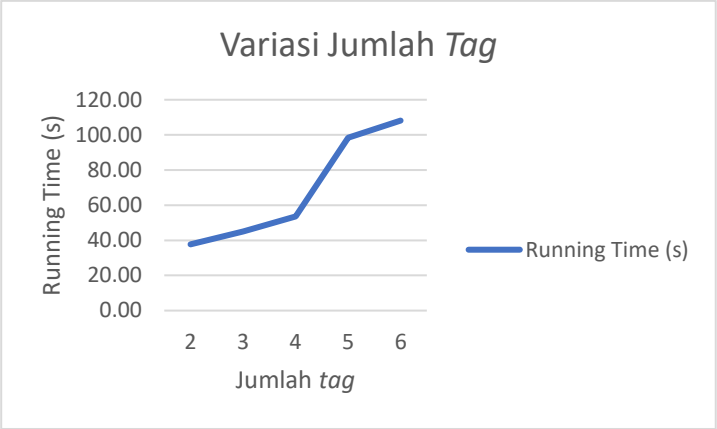
Gambar 5.4 Hasil Pengujian Variasi nilai *threshold*

4. Skenario variasi *spMultThresh*:



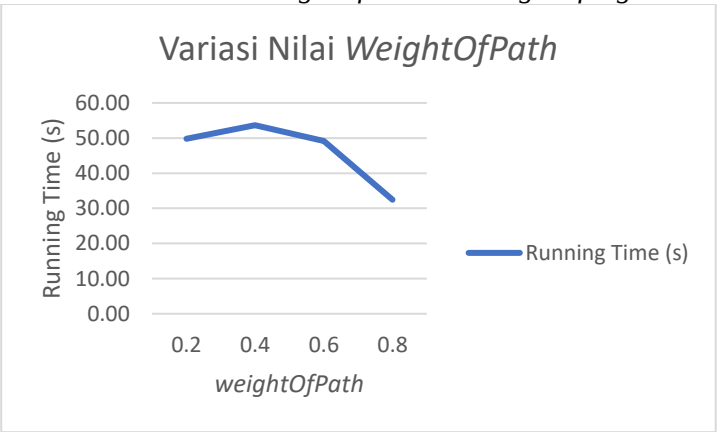
Gambar 5.5 Hasil Pengujian Variasi Nilai *spMultThresh*

5. Skenario variasi jumlah *tag*:



Gambar 5.6 Hasil Pengujian Variasi Jumlah *Tag*

6. Skenario variasi nilai *weightOfPath* dan *weightOfTag*:



Gambar 5.7 Hasil Pengujian Variasi Nilai *weightOfPath*

Pada gambar 5.2, 5.3, 5.4, 5.5, 5.6, dan 5.7 ditampilkan grafik hasil uji coba performa kecepatan sistem untuk variasi nilai parameter yang sudah ditentukan pada subbab sebelumnya.

Untuk detil performa penggunaan memori sistem dapat dilihat pada tabel 5.11.

Tabel 5.11 Hasil Rata-rata Penggunaan Memori Uji Coba Performa KSPD

Kode	Memori (MB)
A01	72,07
A02	134,87
A03	644,57
A04	980,15
B01	138,37
B02	134,87
B03	169,04
B04	177,81
B05	182,84
C01	151,39
C02	134,87
C03	145,78
D01	152,60
D02	134,87
D03	158,46
D04	155,26
E01	126,29

E02	125,58
E03	134,87
E04	164,70
E05	165,16
F01	136,45
F02	134,87
F03	132,37
F04	125,58

BAB VI
KESIMPULAN DAN SARAN

Bab ini akan memaparkan kesimpulan yang diambil selama pengerjaan Tugas Akhir serta saran-saran tentang pengembangan yang dapat dilakukan terhadap Tugas Akhir ini di masa yang akan datang.

6.1. Kesimpulan

1. Parameter yang paling mempengaruhi kinerja masalah *top-k shortest path* dengan keragaman adalah jumlah *vertex*, karena ada banyaknya pilihan rute yang harus ditelusuri. Selain itu jumlah *tag* juga termasuk dalam parameter yang cukup mempengaruhi kinerja masalah ini, karena adanya banyak rute dengan jarak dan jumlah *tag* terpenuhi yang sama.
2. Masalah ini dapat diimplementasikan dengan menggunakan python sebagai server dan java sebagai client perangkat bergerak.

6.2. Saran

1. Penggunaan database atau metadata untuk distribusi data graf.
2. Penggunaan struktur data yang lebih baik.
3. Penambahan fitur pengecekan edge dengan bobot terbesar pada rute yang didapat.

[Halaman ini sengaja dikosongkan]

DAFTAR PUSTAKA

[1] J. Y. Yen, "Finding the k shortest loopless paths in a network," *Management Science,* vol. 17, no. 11, pp. 712-716, 1971.

[2] V. Akgün, E. Erkut dan B. Rajan, "On finding dissimilar paths," *European Journal of Operational Research,* vol. 121, no. 2, pp. 232-246, 2000.

[3] E. W. Dijkstra, "A note on two problem in connexion with graphs," *Numerische Mathematik,* vol. 1, no. 1, pp. 269-271, 1959.

[4] "What is Python? Executive Summary," python.org, [Online]. Available: https://www.python.org/doc/essays/blurb/. [Diakses 20 3 2018].

[5] I. Efendi, "Pengertian Struktur Data," [Online]. Available: https://www.it-jurnal.com/pengertian-struktur-data/. [Diakses 13 July 2018].

[6] "Client-server architecture," [Online]. Available: https://www.britannica.com/technology/client-server-architecture. [Diakses 20 March 2018].

[7] M. A. Basri, "Pengertian Jaringan Client Server Beserta Kelebihan dan Kekurangannya," [Online]. Available: https://www.nesabamedia.com/pengertian-jaringan-client-server/. [Diakses 14 7 2018].

[8] D. Eppstein, "Finding the k Shortest Paths," *SIAM Journal on Computing,* vol. 28, no. 2, pp. 652-673, 2006.

[9] J. Hershberger, M. Maxel dan S. Suri, "Finding the k shortest simple paths: A new algorithm and its implementation," *ACM Transactions on Algorithms,* vol. 3, no. 4, p. 45, 2007.

[10] L. Qin, J. X. Yu dan L. Chang, "Diversifying top-k results," *Proceedings of the VLDB Endowment,* vol. 5, no. 11, pp. 1124-1135, 2012.

[11] M. R. Vieira, H. L. Razente, M. C. N. Barioni, M. Hadjieleftheriou, D. Srivastava, C. Traina dan V. J. Tsotras, "On Query Result Diversification," *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering,* pp. 1163-1174, 2011.

[12] H. Liu, C. Jin dan A. Zhou, "Finding Top-K Shortest Path with Diversity," *IEEE Transactions on Knowledge and Data Engineering,* vol. 30, no. 3, pp. 488-502, 2017.

[13] Y. Dodge, The Oxford Dictionary of Statistical Terms, 2006.

LAMPIRAN

```
import string
import random
import csv
import collections
import time
import heapq
import os
import psutil
import math

class Edge:
    def __init__(self, vertex1=None, vertex2=None,
cost=None):
        self.vertex1 = vertex1
        self.vertex2 = vertex2
        self.cost = cost

def npr(n, r):
    return math.factorial(n)/math.factorial(n-r)

def ncr(n, r):
    return npr(n,r)/math.factorial(r)

def shortestPath(graph, source, sink):
    queue, visited = [(0, source, [])], set()
    heapq.heapify(queue)
    while queue:
        (cost, node, path) = heapq.heappop(queue)
        if node not in visited:
            visited.add(node)
            path = path + [node]
            if node == sink:
                return (cost, path)
            for c, neighbour in graph[node]:
                if neighbour not in visited:
```

```
                    heapq.heappush(queue, (cost+c,
neighbour, path))
    return float("inf")

numberOfVertices = input("jumlah vertex : ")

timestart = time.time()
processMemory = psutil.Process(os.getpid())
WORDS = open("words.txt", "r").read().splitlines()

vertex = []
edge = []

pair = []
paircost = collections.defaultdict()
for i in
range(numberOfVertices*random.randrange(25,50)):
    tag = []
    vertexName1 = random.randrange(1,
numberOfVertices+1)
    vertexName2 = random.randrange(1,
numberOfVertices+1)
    if vertexName1!=vertexName2:
        for j in range(random.randrange(1, 5)):
            while True:
                temp = WORDS[random.randrange(0,
len(WORDS))]
                if "#"+str(temp) not in tag:
                    tag.append("#"+str(temp))
                    break
        cost = random.randrange(1,50)
        paircost[str((vertexName1, vertexName2))] =
((vertexName1, vertexName2),cost, str(tag))
        paircost[str((vertexName2, vertexName1))] =
((vertexName2, vertexName1),cost, str(tag))
```

```
with open("EdgesData"+str(numberOfVertices)+".csv", "w")
as myFile:
    myFile.truncate()
    for i in paircost:
        myFile.write("\""+str(i)+"\","+str(paircost[i][1
])+",\""+str(paircost[i][2])+"\"\n")

timeend = time.time()
print processMemory.memory_info().rss
print timeend-timestart
```

Kode Sumber 0.1 Proses *Preprocessing* Pembuatan Graf

```
from __future__ import division
import socket
import threading
import os
import select
import sys
import collections
import heapq
import csv
import time
import psutil

def recvall(sock):
    data=[]
    while True:
        part = sock.recv(1024)
        if part == "":
            return "Disconnected"
        if part[-14:len(part)]=="stopReadBuffer":
            data.append(part[0:len(part)-14])
            break
        else:
            data.append(part)
    return ''.join(data)

def generateGraph():
    vertex=[]
    for i in range(1, numberOfVertices+1):
        vertex.append(str(i))

    dataEdge=[]
    with open("EdgesData"+str(numberOfVertices)+".csv",
"r") as myFile:
        reader = csv.reader(myFile)
        for i in reader:
            dataEdge.append(i)
```

```
    edges = collections.defaultdict(list)
    for i in range(len(dataEdge)):
        dataEdge[i][0] = dataEdge[i][0].replace("\'",
"")
        dataEdge[i][0] = dataEdge[i][0].replace(" ", "")
        dataEdge[i][0] = dataEdge[i][0].replace("(", "")
        dataEdge[i][0] = dataEdge[i][0].replace(")", "")
        dataEdge[i][0] = dataEdge[i][0].split(",")
        dataEdge[i][2] = dataEdge[i][2].replace("\'",
"")
        dataEdge[i][2] = dataEdge[i][2].replace("(", "")
        dataEdge[i][2] = dataEdge[i][2].replace(")", "")
        dataEdge[i][2] = dataEdge[i][2].replace("[", "")
        dataEdge[i][2] = dataEdge[i][2].replace("]", "")
        dataEdge[i][2] = dataEdge[i][2].replace(" ", "")
        dataEdge[i][2] = dataEdge[i][2].split(",")
        edges[str((dataEdge[i][0][0],dataEdge[i][0][1]))
] = Edge(str(dataEdge[i][0][0]), str(dataEdge[i][0][1]),
int(dataEdge[i][1]), dataEdge[i][2])

    graph = collections.defaultdict(list)
    for i in edges.values():
        if i.vertex1==i.vertex2:
            graph[i.vertex1].append((0,i.vertex2))
        else:
            graph[i.vertex1].append((i.cost,i.vertex2))

    return (vertex, edges, graph)

def _heappop_max(heap):
    lastelt = heap.pop()
    if heap:
        returnitem = heap[0]
        heap[0] = lastelt
        heapq._siftup_max(heap, 0)
        return returnitem
    return lastelt
```

```
class Path:
    def __init__(self, rt, length, cls, tail):
        self.rt = rt
        self.length = length
        self.cls = cls
        self.tail = tail
        self.status = []
    def getAttr(self):
        return str(self.length) + ", " + str(self.rt) +
", " + str(self.cls) + ", " + str(self.status)

class Edge:
    def __init__(self, vertex1=None, vertex2=None,
cost=None, tag=None):
        self.vertex1 = vertex1
        self.vertex2 = vertex2
        self.cost = cost
        self.tag = tag

class Server:
    def __init__(self):
        self.host = "localhost"
        self.port = 8888
        self.size = 1024
        self.server = None
        self.threads = [self.server]

    def open_socket(self):
        try:
            self.server = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
            self.server.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, 1)
            self.server.bind((self.host, self.port))
            self.server.listen(5)
        except socket.error, (value, message):
```

```
            if self.server:
                self.server.close()
                print "couldn't open socket : " +
message
                sys.exit(1)

    def run(self):
        self.open_socket()
        input = [self.server]
        running = 1
        print "Server Ready!!!"
        try:
            while running:
                inputready, outputready, exceptready =
select.select(input, [], [])
                for sock in inputready:
                    if sock == self.server:
                        cs =
Client(self.server.accept())
                        cs.start()
                        print str(cs.address) + " has
connected\n"
                        self.threads.append(cs)
                    else:
                        break

            self.server.close()
            for c in self.threads:
                c.join()

        finally:
            self.server.close()
            sys.exit()

class Client(threading.Thread):
    def __init__(self, (client, address)):
        threading.Thread.__init__(self)
```

```
        self.client = client
        self.address = address
        self.size = 1024
        self.s=""
        self.t=""
        self.threshold=0.0
        self.k=0
        self.spMultThres = 0
        self.weightOfTag=0.0
        self.weightOfPath=0.0
        self.now=0
        self.finalpath=[]
        self.requestedTag=[]
        self.inactive=[]
        self.GQ=[]
        self.LQ = collections.defaultdict(list)
        self.covered = collections.defaultdict(list)
        self.reversepath=collections.defaultdict(str)
        self.running=True

    def contains(self,data1,data2):
        sameEdge = []
        total = 0
        for i in range(len(data1)-1):
            for j in range(len(data2)-1):
                if
data1[i]+data1[i+1]==data2[j]+data2[j+1]:
                    sameEdge.append([data1[i],
data1[i+1]])
        for i in sameEdge:
            for j in edges2:
                if i==j[0]:
                    total+=j[1]
        return total

    def evaluateAndRank(self, pathToBeEvaluated, SP,
spThreshold):
```

```
        tagFulfilled=collections.defaultdict()
        for i in self.requestedTag:
            for j in range(len(pathToBeEvaluated[1])-1):
                if i in
edges[str((pathToBeEvaluated[1][j],
pathToBeEvaluated[1][j+1]))].tag:
                    tagFulfilled[i]=(pathToBeEvaluated[1
][j], pathToBeEvaluated[1][j+1])
                    break
        tag =
eval(str(len(tagFulfilled.keys())/len(self.requestedTag)
))
        if pathToBeEvaluated[0]>spThreshold:
            weight = 0
        else:
            weight = eval(str((spThreshold-
pathToBeEvaluated[0])/(spThreshold-SP[0])))
        return
(tag*self.weightOfTag/100+weight*self.weightOfPath/100,
len(tagFulfilled.keys()), tagFulfilled)

    def shortestPathFunc(self, g, s, t):
        queue, visited = [(0, s, [])], set()
        heapq.heapify(queue)
        while queue:
            (cost, node, path) = heapq.heappop(queue)
            if node not in visited:
                visited.add(node)
                path = path + [node]
                if node == t:
                    return (cost, path)
                for c, neighbour in g[node]:
                    if neighbour not in visited:
                        heapq.heappush(queue, (cost+c,
neighbour, path))
        return (float("inf"), float("inf"))
```

```
    def constructPartialSPT(self, G, t, v):
        if self.reversepath[v]!=-1:
            return self.reversepath[v]
        else:
            self.reversepath[v] =
self.shortestPathFunc(graph, v, t)
            return self.reversepath[v]

    def inputLQ(self, edges, shortestPath):
        if shortestPath!=float("inf"):
            stpcost, stppath = shortestPath
            for i in range(len(stppath)-2,-1,-1):
                x=stppath[i]
                costtemp,pathtemp =
self.shortestPathFunc(graph, self.s, x)
                for c, neighbour in graph[x]:
                    if neighbour !=
self.constructPartialSPT(graph, self.t, x)[1][1] and
neighbour not in pathtemp and
self.constructPartialSPT(graph, self.t,
neighbour)!=float("inf"):
                        self.LQ[neighbour].append((costt
emp+c+self.constructPartialSPT(graph, self.t,
neighbour)[0], Path(pathtemp+[neighbour], costtemp+c,
(stppath,x), neighbour)))
                        heapq.heapify(self.LQ[neighbour]
)
                        self.GQ.append(self.LQ[neighbour
])
        else:
            return

    def findNextPath(self, G, t, threshold):
        while len(self.GQ)!=0:
            heapq.heapify(self.GQ)
            temp = heapq.heappop(self.GQ)
            while len(temp)==0:
```

```
                if len(self.GQ)!=0:
                    temp=heapq.heappop(self.GQ)
                else:
                    return None
            P = heapq.heappop(temp)

            if len(temp)!=0 and temp not in self.GQ:
                self.GQ.append(temp)
            LB2Array=[]
            for x in heapq.nlargest(self.k,
self.finalpath):
                total = self.contains(P[1].rt, x[1][1])
                LB2Array.append(int(total*(1+1/self.thre
shold)-x[1][0]))
            while P[1].tail!=self.t:
                LB2 = max(LB2Array)
                if LB2>P[0]:
                    tempLB1=LB2
                    self.adjustPath(P)
                    self.LQ[P[1].tail].append((tempLB1,
P[1]))
                    if self.LQ[P[1].tail] not in
self.GQ:
                        self.GQ.append(self.LQ[P[1].tail
])
                    break
                elif not self.extendPath(P):
                    break

            if P[1].tail==self.t:
                self.adjustPath(P)
                return (P[1].length,P[1].rt)

    def extendPath(self, P):
        for x in self.LQ[P[1].tail]:
            (tempLB1, tempPath) = x
```

```
            if tempPath.length>=P[1].length and
tempPath.cls==P[1].cls:
                self.inactive.append(self.LQ[P[1].tail][
self.LQ[P[1].tail].index(x)])
                self.LQ[P[1].tail].__delitem__(self.LQ[P
[1].tail].index(x))
                P[1].status=tempPath.rt

        for c, neighbour in graph[P[1].tail]:
            if neighbour not in P[1].rt and
neighbour!=self.constructPartialSPT(graph, self.t,
P[1].tail)[1][1] and self.constructPartialSPT(graph,
self.t, neighbour)!=float("inf"):
                tempP = Path(P[1].rt+[neighbour],
P[1].length+c, P[1].cls, neighbour)
                tempP.status = P[1].status
                tempPLB1 = P[0]-
self.constructPartialSPT(graph, self.t,
P[1].tail)[0]+c+self.constructPartialSPT(graph, self.t,
neighbour)[0]
                if self.covered[neighbour]==P[1].cls:
                    self.inactive.append((tempPLB1,
tempP))
                else:
                    self.LQ[neighbour].append((tempPLB1,
tempP))
                    heapq.heapify(self.LQ[neighbour])
                    self.covered[neighbour] = P[1].cls
                    if self.LQ[neighbour] not in
self.GQ:
                        self.GQ.append(self.LQ[neighbour
])

        if len(self.constructPartialSPT(graph, self.t,
P[1].tail)[1])>1:
            if self.constructPartialSPT(graph, self.t,
P[1].tail)[1][1] in P[1].rt:
```

```
                return False
            else:
                P[1].rt = P[1].rt +
[self.constructPartialSPT(graph, self.t,
P[1].tail)[1][1]]
                P[1].length = P[1].length +
self.constructPartialSPT(graph, self.t, P[1].tail)[0] -
self.constructPartialSPT(graph, self.t,
self.constructPartialSPT(graph, self.t,
P[1].tail)[1][1])[0]
                self.covered[neighbour] = P[1].cls
                P[1].tail =
self.constructPartialSPT(graph, self.t, P[1].tail)[1][1]
                return True
        else:
            return

    def adjustPath(self, P):
        if len(P[1].status)!=0:
            for x in self.inactive:
                if len(x)!=0:
                    if x[1].rt==P[1].status:
                        self.LQ[P[1].status[len(P[1].sta
tus)-1]].append(x)
                        if
self.LQ[P[1].status[len(P[1].status)-1]] not in self.GQ:
                            self.GQ.append(self.LQ[P[1].
status[len(P[1].status)-1]])
                        self.inactive.__delitem__(self.i
nactive.index(x))

        if P[1].tail==self.t:
            for i in self.GQ:
                for j in i:
                    if len(j[1].rt) < len(P[1].rt):
                        for l in range(1, len(j[1].rt)):
                            if j[1].rt[l]==P[1].rt[l]:
```

```
                                self.LQ[j[1].rt[len(j[1]
.rt)-1]][self.LQ[j[1].rt[len(j[1].rt)-
1]].index(j)][1].cls=(P[1].rt, j[1].rt[l])
                            else:
                                break
                    else:
                        for l in range(1, len(P[1].rt)):
                            if j[1].rt[l]==P[1].rt[l]:
                                self.LQ[j[1].rt[len(j[1]
.rt)-1]][self.LQ[j[1].rt[len(j[1].rt)-
1]].index(j)][1].cls=(P[1].rt, j[1].rt[l])
                            else:
                                break

    def run(self):
        self.client.send(str(edges2)+"stopReadBuffer")
        data = recvall(self.client)
        if data=="Disconnected":
            print str(self.address) + " is Disconnected"
        while data!="Disconnected":
            self.GQ=[]
            self.LQ.clear()
            self.covered.clear()
            self.reversepath.clear()
            self.inactive=[]
            self.requestedTag=[]
            self.now=0
            self.finalpath=[]

            temp1, temp2 = data.split("; ")
            self.s, self.t, self.threshold, self.k,
self.spMultThres, self.weightOfTag, self.weightOfPath =
temp1.split(", ")

            self.s = str(self.s.upper())
            self.t = str(self.t.upper())
            self.threshold=float(self.threshold)
```

```
            self.k=int(self.k)
            self.spMultThres=int(self.spMultThres)
            self.weightOfTag=float(self.weightOfTag)
            self.weightOfPath=float(self.weightOfPath)
            if temp2!="[]":
                temp2 = temp2.replace(" ","")
                temp2 = temp2.replace("[","")
                temp2 = temp2.replace("]","")
                temp2 = temp2.replace("(","")
                temp2 = temp2.replace(")","")
                temp2 = temp2.replace("\'","")
                temp2 = temp2.split(",")
                for i in temp2:
                    if i[0]!='#':
                        i = '#'+i
                    self.requestedTag.append(i)
            else:
                self.requestedTag.append("#")
            print str(self.address) + " request : " +
str(self.s) + ", " + str(self.t) + ", " +
str(self.threshold) + ", " + str(self.k) + ", " +
str(self.spMultThres) + ", " + str(self.weightOfTag) +
", " + str(self.weightOfPath)
            if self.requestedTag[0]=="#":
                print "With no tag"
            else:
                print "With tag : " +
str(self.requestedTag)

            for i in vertex:
                self.reversepath[i]=-1

            timeStartSP = time.time()
            self.shortestPath =
self.shortestPathFunc(graph, self.s, self.t)
            timeEndSP = time.time()
```

```
            print "SP time : "+str(timeEndSP -
timeStartSP), self.shortestPath
            if self.shortestPath[0]!=float("inf"):
                spThreshold =
self.spMultThres*self.shortestPath[0]

                temp =
self.evaluateAndRank(self.shortestPath,
self.shortestPath, spThreshold)
                heapq.heappush(self.finalpath, (temp[0],
self.shortestPath, temp[1], temp[2]))

                timeStartInputLQ = time.time()
                self.inputLQ(edges, self.shortestPath)
                heapq.heapify(self.GQ)
                timeEndInputLQ = time.time()
                print "Input LQ time :
"+str(timeEndInputLQ-timeStartInputLQ)

                timeStart = time.time()
                while len(self.GQ)!=0:
                    nextPath=self.findNextPath(graph,
self.t, self.threshold)
                    if nextPath==None:
                        break
                    flag=0
                    for x in heapq.nlargest(self.k,
self.finalpath):
                        total =
self.contains(nextPath[1], x[1][1])
                        if
float(total)/(x[1][0]+nextPath[0]-total) <
float(self.threshold):
                            flag+=1
                    if flag==len(heapq.nlargest(self.k,
self.finalpath)):
                        self.now+=1
```

```
                        finalPathTemp =
self.evaluateAndRank(nextPath, self.shortestPath,
spThreshold)
                        finalPathTemp =
(finalPathTemp[0], nextPath, finalPathTemp[1],
finalPathTemp[2])
                        if finalPathTemp not in
self.finalpath:
                            if
len(self.finalpath)<=self.k-1:
                                heapq.heappush(self.fina
lpath, finalPathTemp)
                            else:
                                temp = ((spThreshold -
nextPath[0])/(spThreshold -
self.shortestPath[0]))*self.weightOfPath/100+self.weight
OfTag/100
                                temp2 =
heapq.nlargest(self.k,self.finalpath)[self.k-1]
                                if temp>temp2[0]:
                                    if
finalPathTemp[0]>temp2[0]:
                                        heapq.heapreplac
e(self.finalpath, finalPathTemp)
                                else:
                                    break
                timeEnd = time.time()
                processMemory =
psutil.Process(os.getpid())
                print processMemory.memory_info().rss
                print "KSPD time : "+str(timeEnd -
timeStart)+"\n"
            if len(self.finalpath)>0:
                if len(self.finalpath)<self.k:
                    show = len(self.finalpath)
                else:
                    show = self.k
```

```
                self.client.send(str(heapq.nlargest(show
, self.finalpath))+"stopReadBuffer")
            else:
                print "Tidak ada path yang valid\n"
                self.client.send("Tidak ada path yang
validstopReadBuffer")
            data = recvall(self.client)
            if data=="Disconnected":
                print str(self.address) + " is
Disconnected"
        self.client.close


if __name__ == "__main__":

    numberOfVertices = input("Jumlah Vertex : ")
    timeStartPreProcessing1 = time.time()
    vertex, edges, graph = generateGraph()

    edges2=[]
    for x in edges.values():
        edges2.append(([x.vertex1, x.vertex2], x.cost))

    timeEndPreProcessing1 = time.time()
    print "Create Graph Preprocessing :
"+str(timeEndPreProcessing1-timeStartPreProcessing1)
    s = Server()
    s.run()
```

Kode Sumber 0.2 Proses Algoritma KSPD

BIODATA PENULIS

Bramastya Dewa Indraswara, lahir pada tanggal 16 Nopember 1997, Surabaya. Penulis merupakan anak kedua dari dua bersaudara. Penulis pernah menempuh pendidikan di SD MIMI, SMP Kristen Petra 2, dan SMA Kristen Petra 1. Saat ini penulis sedang menempuh pendidikan perguruan tinggi negeri di Institut Teknologi Sepuluh Nopember Surabaya di jurusan Teknik Informatika, Fakultas Teknologi Informasi, angkatan tahun 2015.

Dalam Rangka menyelesaikan S1, penulis mengambil bidang minat kuliah Komputasi Berbasis Jaringan (KBJ), dan memiliki minat dalam bidang komputasi bergerak dan jaringan.

bramas16@gmail.com