**PROPOSAL PENELITIAN**

****

**PENERAPAN ALGORITMA DIJKSTRA UNTUK PENENTUAN RUTE TERPENDEK**

**Studi Kasus : Lokasi Kampus Universitas Budi Luhur**

**Oleh :**

**Noni Juliasari, M.Kom (020018)**

**FAKULTAS TEKNOLOGI INFORMASI**

**UNIVERSITAS BUDI LUHUR**

**FEBRUARI 2014**

**HALAMAN PENGESAHAN**

1. Judul Penelitian : Penerapan Algoritma Dijkstra Untuk Penentuan Rute

 Terpendek Studi Kasus : Lokasi Kampus

 Universitas Budi Luhur

1. Bidang Penelitian : ICT
2. Ketua Peneliti
3. Nama Lengkap : Noni Juliasari, M.Kom (980018)
4. NIP : 020018
5. Disiplin Ilmu : Komputer
6. Pangkat/Golongan : Lektor
7. Jabatan : Dosen Tetap
8. Fakultas/Jurusan : Fakultas Teknologi Informasi/Sistem Informasi
9. Alamat Rumah : Graha Karang Tengah Blok F/78 RT.005/03

 Pondok Pucung, Karang Tengah, Banten

1. Telepon/e-mail : 021 94798562/noni.juliasari@budiluhur.ac.id
2. Melibatkan Mahasiswa

Nama : Putri Rahayu

NIM : 1011502653

1. Jumlah biaya diusulkan : **Rp.**

 Jakarta, 20 Maret 2014

 Mengetahui,

 DEKAN

 Fakultas Teknologi Informasi Ketua Tim Pengusul

 Goenawan Brotosaputro, S.Kom, MSc Noni Juliasari, M.Kom

Menyetujui,

Deputi Rektor Bidang Akademik

Dr.Ir.Wendi Usino, MSc, MM

**DAFTAR ISI**

Halaman Pengesahan……………………………………………………………… 2

Daftar Isi………………………………………………………………………….. 3

Ringkasan………………………………………………………………………… 4

Bab 1. Pendahuluan……………………………………………………………… 5

* 1. Latar Belakang…………………………………………………………… 5
	2. Tujuan……………………………………………………………………. 6
	3. Batasan masalah………………………………………………………….. 6

Bab 2. Tinjauan Pustaka…………………………………………………………. 6

Bab 3. Metode Penelitian………………………………………………………… 8

Bab 4. Biaya dan Jadwal Penelitian……………………………………………… 9

1. Anggaran Biaya………………………………………………………….. 9
2. Jadwal Kegiatan…………………………………………………………. 10

Daftar Pustaka…………………………………………………………………… 10

Lampiran-lampiran

1. Anggaran Penelitian……………………………………………………… 11
2. Susunan Organisasi Tim Peneliti dan Pembagian Tugas………………… 12
3. Biodata Tim Peneliti……………………………………………………… 13

**RINGKASAN**

Pencarian jalur terpendek (shortest path finding) dalam perjalanan merupakan masalah sehari-hari yang biasa ditemui disaat akan mencari jalur perjalanan terdekat ke tempat tujuan. Dalam keseharian, pencarian rute terpendek digunakan misalnya oleh seorang pengendara kendaraan bermotor, pejalan kaki bahkan trayek angkutan umum. Permsalahan yang diangkat dalam penelitian ini adalah mencoba memformulasikan pencarian jalur terpendek yang nantinya dapat dimanfaatkan oleh dosen Universitas Budi Luhur yang bertugas mengajar antar kampus yang berlokasi di Petukangan, Roxy dan Salemba. Banyaknya rute di Jakarta ini, membuat dosen Budi Luhur tidak tepat untuk menentukan rute manakah yang paling pendek dari kampus pusat ke kampus tujuan. Persoalan penentuan lintasan terpendek ini bisa diselesaikan dengan algoritma Dijkstra dengan menggunakan pengoptimasian dari priority queue. Algoritma Dijkstra adalah jenis algoritma optimasi yang memecahkan permasalahan solusi optimum dan sebuah algoritma yang dipakai dalam memecahkan permasalahan rute terpendek untuk sebuah graph berarah dengan bobot-bobot sisi yang bernilai positif.

**BAB 1. PENDAHULUAN**

1. Latar Belakang

Dalam kehidupan sehari-hari, sering dilakukan perjalanan dari satu tempat ke tempat lain dengan mempertimbangkan efisiensi jarak atau rute, sehingga diperlukan ketepatan dalam menentukan jalur terpendek antara satu tempat. Hasil penentuan rute terpendek akan menjadi pertimbangan dalam pengambilan keputusan untuk menunjukkan rute yang akan ditempuh. Hasil yang didapatkan juga membutuhkan kecepatan dan keakuratan dengan bantuan komputer. Untuk meggunakan atau memfungsikan sebuah komputer maka harus terdapat program yang terdistribusi di dalamnya.

Banyaknya rute di Jakarta ini membuat masyarakat Jakarta kadang tidak tepat dalam menentukan rute yang efisien untuk perjalanan dari kota ke kota lain. Universitas Budi Luhur adalah sebuah Universitas yang berpusat di Jl. Raya Ciledug-Petukangan Utara, Jakarta Selatan. Universitas Budi Luhur mempunyai cabang yang berlokasi di ITC Roxy Mas dan di Salemba Raya(Matraman). Dari kampus pusat ke kampus Roxy memiliki banyak rute begitu juga dengan kampus Salemba. Rute pendek untuk mencapai tujuan ke kampus Roxy atau Salemba menjadi kendala yang dihadapi oleh dosen Budi Luhur yang mengajar di beberapa lokasi kampus, sedangkan jalur yang dilewati oleh dosen-dosen kadang tidak selalu tepat sementara waktu tempuh yang diharapkan adalah minimal agar tidak terlambat tiba di kelas.

Oleh karena itu, penelitian ini dimaksudkan untuk mengembangkan sistem pencarian rute terpendek yang akan membantu pengguna dalam melakukan pencarian dan menemukan rute menuju lokasi kampus yang diinginkan. Sistem pencarian ini juga menghadirkan solusi pencarian jalur terpendek dengan mengimplementasikan algoritma Dijkstra dengan pengoptimasian Priority Queue. Algoritma Dijktra adalah jenis algoritma optimasi yang memecahkan permasalahan solusi optimum dan sebuah algoritma yang dipakai dalam memecahkan permasalahan rute terpendek untuk sebuah graph berarah dengan bobot-bobot sisi yang bernilai positif. Sedangkan Priority Queue adalah suatu bentuk struktur data yang berbasiskan struktur queue pada umumnya.

1. Perumusan Masalah

Masalah yang akan diangkat sebagai objek penelitian ini antara lain :

* 1. Bagaimana menentukan rute perjalanan terpendek dari satu lokasi kampus ke lokasi kampus tujuan?
	2. Bagaimana mendapatkan informasi rute ke tujuan dengan mudah?
	3. Bagaimana mendapatkan data jarak antar satu tempat ke tempat lain secara tepat?
1. Tujuan Penelitian

Tujuan penelitian ini adalah :

1. Merancang aplikasi pencarian rute terpendek dalam menentukan rute perjalanan antar lokasi kampus Budi Luhur.
2. Mengimplementasikan algoritma Dijkstra dalam pencarian rute terpendek
3. Manfaat Penelitian

Penelitian ini dapat bermanfaat bagi dosen maupun mahasiswa yang ingin menempuh perjalanan antar lokasi kampus Budi Luhur untuk penyelenggaraan kegiatan akademiknya.

1. Batasan Masalah
	1. Penelitian ini dibatasi dengan permasalahan pada jarak saja.
	2. Daerah yang dicakup hanya lokasi kampus Budi Luhur saja yaitu Petukangan, Roxy dan Salemba.
	3. Pembahasan ditujukan pada pencarian rute dari satu titik awal ke satu titik akhir, tidak dari satu titik awal ke banyak titik secara bersamaan.
	4. Keluaran yang dihasilkan adalah hasil dari algoritma Dijkstra yang diimplementasikan dalam suatu program sederhana dengan menggunakan aplikasi web.
	5. Rute Petukangan ke Roxy atau Roxy ke Petukangan menghasilkan rute yang sama, begitupun dengan Salemba.

**BAB 2. TINJAUAN PUSTAKA**

1. Landasan Teori

Algoritma Dijkstra

1. Sejarah Algoritm Dijkstra

Pada tahun 1959 sebuah tulisan sepanjang tiga halaman yang berjudul *A Note on Two Problems in Connexion with Graphs* diterbitkan pada jurnal *Numerische Mathematik*. Pada tulisan ini, Edsger W. Dijkstra seorang ilmuwan komputer berumur dua puluh Sembilan tahun mengusulkan algoritma-algoritma untuk solusi dari dua masalah teoritis graf dasar : *the minimum weight*. Algoritma Dijkstra untuk masalah jalan terpendek adalah satu dari algoritma-algoritma paling ternama pada ilmu komputer dan sebuah algoritma paling popular pada operasi pencarian (OR). (Fakhri, 2008).

1. Graf

Teori graf merupakan pokok bahasan yang memiliki banyak terapan sampai saat ini. Graf digunakan untuk merepresentasikan objek-objek diskrit dan hubungan objek-objek tersebut.

Secara matematis graf didefinisikan sebagai pasangan himpunan (V,E), ditulis dengan notasi G = (V,E), yang dalam hal ini V adalah himpunan tidak kosong dari simpul-simpul (*vertex*) dan E adalah himpunan sisi (*edge*) yang menghubungkan sepasang simpul. (Fauzi, 2011)

Simpul pada graf dapat dinyatakan dengan huruf, bilangan atau gabungan keduanya. Sedangkan sisi-sisi yang menghubungkan simpul u dengan simpul v dinyatakan dengan pasangan (u, v) atau dinyatakan dengan lambang e1, e2, e3 dan seterusnya. Dengan kata lain, jika e adalah sisi yang menghubungkan simpul u dengan simpul v, maka e dapat ditulis sebagai e = (u, v).



Gambar 2.1: Sebuah Graf Sederhana

1. Jenis-Jenis Graf

Klasifikasi pada graf cukup luas, klasifikasi tersebut bergantung pada factor-faktor yang membedakannya. Berdasarkan orientasi arah pada sisinya, maka secara umum graf dibedakan atas dua jenis sebagai berikut :

1. Graf tidak berarah (*undirected graph*)

Graf yang sisinya tidak memiliki orientasi arah disebut graf tidak berarah. Pada graf tidak berarah, urutan pasangan simpul yang dihubungkan oleh sisi tidak diperhatikan. Jadi (u, v) = (v, u) adalah sisi yang sama.



 Gambar 2.2: Graf Tidak Berarah

1. Graf berarah (*direct graph*)

Graf yang setiap sisinya diberikan orientasi arah disebut graf berarah, pada graf berarah (u, v) dan (v, u) menyatakan dua buah sisi yang berbeda. Dengan kata lain dapat ditulis (u, v) ≠ (v, u)



 Gambar 2.3: Graf Berarah

1. Graf Berbobot (*Weight Graph*)

Dalam memodelkan suat masalah ke dalam graf, ada informasi yang ditambahkan pada *arc graph*. Misalkan pada graf yang menggambarkan peta jalan raya antar kota-kota dapat ditambahkan sebuah bilangan pada setiap *arc* untuk menunjukkan jarak antara kedua kota yang dihubungkan oleh *arc* terebut. Graf berbobot adalah suatu graf tanpa *arc* parallel dimana setiap *arc*-nya berhubungan dengan suatu bilangan riil tak negative yang menyatakan bobot *arc* (w(a)) tersebut.



 Gambar 2.4: Graf Berbobot

1. *Path* (Lintasan)

Lintasan adalah hubungan antar titik atau *node* dalam sebuah *graph*. Suatu lintasan yang berawal dan berakhir pada *node* yang sama, maka akan disebut dengan lintasan tertutup (*close path*), jika *node* awal dan  *node* akhir dari lintasan tersebut berbeda maka disebut dengan lintasan terbuka (*open path*).

Lintasan terpendek (*shortest path*) adalah lintasan yang memiliki total bobot minimum untuk mencapai suatu tempat dari tempat tertentu. Lintasan terpendek dapat dicari dengan menggunakan *graph*. *Graph* yang digunakan adalah *graph* yang berbobot, yaitu *graph* yang setiap sisinya diberikan suatu nilai atau bobot. Bobot pada sisi *graph* dapat menyatakan waktu, biaya, dan sebagainya.



Gambar 2.5: *Shortest Path Graph*

Gambar 2.5 meninjukan lintasan terpendek (garis tebal) dari *node* 1 ke *node* 5 dengan jalur **1-3-6-5** dengan bobot dua puluh (20).

1. Rumusan Algoritma Dijkstra

Misalkan :

V(G) = {v1,v­2,v3,…,vn}

L = Himpunan *node* V(G) yang sudah terpilih dalam jalur terpendek

D(j) = Jumlah bobot jarak terkecil dari v1 ke vj

W(i,j) = Bobot garis dari *node* vi ke vj

W\*(1,j) = Jumlah bobot jarak terkecil dari v1 ke vj

Secara formal, algoritma Dijkstra untuk mencari jarak terpendek adalah :

1. L = { }

V = {v2,v3, …,vn}

1. Untuk i = 2, 3, …, n, lakukan D(i) = W(1,i)
2. Selama vn ∈ L lakukan :
3. Pilih node vk V-L dengan D(k) terkecil

L = L ∪ {vk}

1. Untuk setiap vj V-L lakukan :

Jika D(j) > D(k) + W(k,j) maka ganti D(j) dengan D(k) + W(k,j)

1. Untuk setiap vj ∈ V, W\*(1,j) = D(j)

Menurut algoritma diatas, jarak terpendek dari *node* v1 ke vn adalah melalui *node-node* dalam L secara berurutan, dan jumlah bobot jarak terkecilnya adalah D(n).

1. **Penerapan Pencarian Algoritma Dijkstra**

Algoritma ini bertujuan untuk menemukan jalur terpendek berdasarkan bobot terkecil dari satu titik ke titik lainnya. Misalkan titik menggambarkan gedung dan garis menggambarkan jalan, maka algoritma *Dijkstra* melakukan kalkulasi terhadap semua kemungkinan bobot terkecil dari setiap titik.



Gambar 2.6: Contoh keterhubungan antar titik dalam algoritma Dijkstra

Pertama-tama tentukan titik mana yang akan menjadi *node* awal, lalu beri bobot jarak pada *node* pertama ke *node* terdekat satu per satu, Dijkstra akan melakukan pengembangan pencarian dari satu titik ke titik lain dan ke titik selanjutnya tahap demi tahap. Inilah urutan logika dari algoritma Dijkstra :

1. Beri nilai bobot (jarak) untuk setiap titik lainnya, lalu set nilai 0 pada *node* awal dan nilai tak terhingga terhadap *node* lain (belum terisi)
2. Set semua *node* “Belum Terlewati” dan set node awal sebagai “*Node* Keberangkatan”
3. Dari *node* keberangkatan, pertimbangkan *node* tetangga yang belum dilewati dan hitung jaraknya dari titik keberangkatan. Sebagai contoh, jika titik keberangkatan 1 ke 2 memiliki jarak 7 dan dari 2 ke 3 berjarak 10, maka jarak ke 3 melewati jarak ke 2 menjadi 7+10 = 17. Jika jarak ini lebih kecil dari jarak sebelumnya (yang terekam sebelumnya) maka hapus data lama, simpan ulang data dengan jarak yang baru.
4. Saat selesai mempertimbangkan setiap jarak terhadap node tetangga, tandai *node* yang telah dilewati sebagai “*Node* Dilewati”. *Node* dilewati tidak akan pernah di cek ulang, jarak yang disimpan adalah jarak yang terakhir dan yang paling minimal bobotnya.
5. Set “*Node* Belum Dilewati” dengan jarak terkecil (dari *node* keberangkatan) sebagai “*Node* Keberangkatan” selanjutnya dan lanjutkan dengan kembali ke step 3.

Dibawah ini penjelasan langkah per langkah pencarian jalur terpendek secara rinci dimulai dari *node* awal sampai *node* tujuan dengan nilai jarak terkecil.

1. *Node* awal 1, *Node* tujuan 5. Setiap edge yang terhubung antar *node* telah diberi nilai.



Gambar 2.7: Contoh kasus Dijkstra – langkah 1

1. Dijsktra melakukan kalkulasi terhadap *node* tetangga yang terhubung langsung dengan *node* keberangkatan (*node* 1), dan hasil yang didapat adalah *node* 2 karena bobot nilai *node* 2 paling kecil dibandingkan nilai pada *node* lain, nilai = 7 (7+0).



Gambar 2.8: Contoh kasus Dijkstra – langkah 2

1. *Node* 2 diset menjadi *node* keberangkatan dan ditandai sebagai *node* yang telah dilewati. Dijkstra melakukan kalkulasi kembali terhadap kumpulan *node* tetangga yang terhubung dengan *node* yang telah dilewati. Dan kalkulasi Dijkstra menunjukkan bahwa *node* 3 yang menjadi *node* keberangkatan selanjutnya karena bobotnya yang paling kecil dan hasil kalkulasi terakhir, nilai = 9 (0+9).



Gambar 2.9: Contoh kasus Dijkstra – langkah 3

1. Perhitungan berlanjut dengan *node* 3 ditandai menjadi *node* yang telah dilewati. Dari semua *node* dilewati, *node* selanjutnya yang ditandai menjadi *node* dilewati adalah *node* 6 karena nilai bobot yang terkecil 2, nilai = 11 (9+2).



Gambar 2.10: Contoh kasus Dijkstra *–* langkah 4

1. *Node* 6 menjadi *node* dilewati dijkstra melakukan kalkulasi kembali dan menemukan bahwa *node* 5 (*node* tujuan) telah tercapai melalui *node* 6. Jalur terpendeknya adalah **1-3-6-5**, dan nilai bobot yang didapat adalah **20 (11+9)**. Bila *node* tujuan telah tercapai maka kalkulasi Dijkstradinyatakan selesai.



 Gambar 2.11: Contoh kasus Dijkstra – langkah 5

Contoh penerapan lainnya dengan langkah sebagai berikut :

1. Pada awal inisialisasikan *node* asal (V1) dan *node* tujuan (V5).
2. Buat 2 buah *list* yaitu *open list* dan *closed list*. Keduanya tidak ada data atau kosong dengan menggunakan format {*node*, bobot, *node* induk}.
3. Masukan V1 ke *open list*.
4. Pilih 1 *node* dengan bobot terkecil pada *open list*, tambahkan ke dalam *closed list*.
5. Cari *node* yang bertetangga langsung dengan *node* sebelumnya, yang masuk terakhir dalam *closed list*. Tambahkan bobot dengan *node* yang terkait, apabila sudah ada dalam *closed list* abaikan.
6. Apabila dalam *open list* terdapat *node* yang sudah ada maka bandingkanlah, lalu carilah yang terkecil, dan perbarbarui. Bila ternyata jumlah bobotnya sama dalam *node* yang sama, maka abaikan.
7. Apakah data dalam *open list* kosong? Jika belum maka ulangi langkah d.
8. Dalam *closed* *list* cari V5, telusuri dan balikan urutan *node*.
9. Lintasan terpendek ditemukan bersama bobotnya.

Contoh penerapannya adalah sebagai langkah awal inisialisasikan *node* 1 sebagai *node* awal (V1) dan *node* tujuan (V2), buat 2 buah *list*, *open list* dan *closed list*.

Tabel 2.1: Hasil Iterasi ke-1

|  |  |
| --- | --- |
| *Open List* | *Closed List* |
| { 1,0,null } | - |

Dari tabel 2.1 masukan *node* 1 ke dalam *open list*, *node* 1 memiliki bobot 0, dan *node* induknya *null* karena sebagai awal proses pencarian.

Tabel 2.2: Hasil Iterasi ke-2

|  |  |
| --- | --- |
| *Open List* | *Closed List* |
| { 2,7,1 }{ 6,14,1 }{ 3,9,1 } | { 1,0,null } |

Dari tabel 2.2 proses dari bobot terkecil lebih dahulu, dikarenakan ada satu yaitu *node* 1 maka masukan ke dalam *closed list*. *Node*  yang bertetangga dengan *node* 1 adalah *node* 2, *node* 3 dan *node* 6, lalu masukan ke dalam *open list*, *node* 2 memiliki bobot 7 dengan *node* induknya *node* 1, *node* 3 memiliki bobot 9 dengan *node* induknya *node* 1 dan *node* 6 memiliki bobot 14 dengan *node* induknya *node* 1

Tabel 2.3: Hasil Iterasi ke-3

|  |  |
| --- | --- |
| *Open List* | *Closed List* |
| { 6,14,1 }{ 3,9,1 }{ 4,22,2 }{3,17,2} | {1,0,null}{ 2,7,1 } |

Dari tabel 2.3 proses bobot terkecil dahulu yaitu *node* 2 dan masukan ke dalam *closed list*, *node* yang bertetangga dengan *node* 2 adalah *node* 3 dan *node* 4 lalu masukan ke dalam *open list*, *node* 3 memiliki bobot 12 dengan *node* induk 2 dan *Node* 4 memiliki bobot 22 dengan *node* induknya *node* 2. Karna pada tabel 2.4 *node* 3 sudah masuk ke dalam *closed list* maka abaikan.

Tabel 2.4: Hasil Iterasi ke-4

|  |  |
| --- | --- |
| *Open List* | *Closed List* |
| {6,14,1}{ 6,11,3 }{ 4,20,3 } | { 1,0,*null* }{ 2,7,1 }{ 3,9,1 } |

Dari tabel 2.4 proses dari bobot terkecil lebih dahulu yaitu *node* 3 dan masukan ke dalam *closed list*, *node* yang bertetangga dengan *node* 3 adalah *node* 4 dan *node* 6, lalu masukan ke dalam *open list*. Karena *node* 4 dan *node* 6 sudah masuk ke dalam *open list* bandingkan bobotnya, ambil yang terkecil adalah yang memiliki *node* induknya *node* 3 sehingga yang dimasukan ke dalam *open list* adalah *node* 4 yang memiliki bobot 20 dan *node* 6 yang memiliki bobot 11.

Tabel 2.5: Hasil Iterasi ke-5

|  |  |
| --- | --- |
| *Open List* | *Closed List* |
| { 4,20,3 }{ 5,20,6 } | { 1,0,*null* }{ 2,7,1 }{ 3,9,1 }{ 6,11,3 } |

Dari tabel 2.5 proses dari bobot terkecil lebih dahulu yaitu *node* 6 dan masukan ke dalam *closed list*, *node* yang bertetangga dengan *node* 6 adalah *node* 5 lalu masukan ke dalam *open list*. *Node* 5 memiliki bobot 20 dengan *node* induknya *node* 6.

Tabel 2.6: Hasil Iterasi ke-6

|  |  |
| --- | --- |
| *Open List* | *Closed List* |
| { 5,20,6 } | { 1,0,*null* }{ 2,7,1 }{ 3,9,1 }{ 6,11,3 }{ 4,20,3 } |

Dari tabel 2.6 proses dari bobot terkecil lebih dahulu yaitu *node* 4 dan masukan ke dalam *closed list*, *node* yang bertetangga dengan *node* 4 adalah *node* 5 lalumasukan ke dalam *open list*. Karena *node* 5 sudah ada dalam *open list* maka abaikan.

 Tabel 2.7: Hasil Iterasi ke-7

|  |  |
| --- | --- |
| *Open List* | *Closed List* |
| - | { 1,0,*null* }{ 2,7,1 }{ 3,9,1 }{ 6,11,3 }{ 4,20,3 }{ 5,20,6 } |

Dari tabel 2.7 proses dari bobot terkecil lebih dahulu, karena tinggal 1 maka *node* 5 masukan ke dalam *closed list*, *node* yang bertetangga dengan *node* 5 adalah *node* 4 dan *node* 6 lalu masukan ke dalam *open list*. Karena *node* 4 dan *node* 6 sudah ada dalam *open list* maka abaikan.

Pencarian akan berhenti dikarenakan *open list* sudah kosong, didalam *closed list* telusuri V1 atau *node* tujuan dengan mengacu pada *node* induk, dan balikkan urutan *node*-nya, sehingga pencarian jalur terpendek dari *graph* yang dimulai dari *node* 1 ke *node* 5 diperoleh jalur **1-3-6-5** denfan julmah bobot **20**. Dapat dilihat pada gambar 2.9.



 Gambar 2.12 Sesudah pencarian jalur terpendek

Penerapan algoritma Dijkstra juga dapat dilakukan dengan *Matriks* M, misalkan sebuah *graph* berbobot n buah simpul dinyatakan dengan *matriks* ketetanggaan M=[mij], yang dalam hal ini :

|  |
| --- |
| mij = bobot sisi (i,j) (pada *graph* berarah mij=mji)mij = 0mij = ∞, jika tidak ada sisi dari simpul i ke simpul j |

Selain *matriks* M, algoritma *Dijkstra* juga dapat menggunakan larik S=[si] dengan menggunakan *pseudocode* sabagai berikut :

|  |
| --- |
| Si = 1, jika simpul i termasuk ke dalam lintasan terpendekSi = 0, jika simpul i tidak termasuk ke dalam lintasan terpendekdan larik/tabel D=[di] yang dalam hal ini,di = panjang lintasan dari simpul awal ke simpul iAlgoritma Lintasan Terpendek Dijkstra (mencari lintasan terpendek dari simpul a ke simpul lain)Langkah 0 (inisialisasi) :Inisialisasi si = 0 dan di = mij untuk I = 1,2,3,….,nLangkah 1 :Isi sa dengan 1 (karena simpul a adalah simpul asal lintasan terpendek, jadi sudah pasti terpilih)Isi da dengan ∞ (tidak ada lintasan terpendek dari simpul a ke a)Langkah 2,3,…,n-1 :Cari j sedemikian hingga sj=0 dan dj=min{d1,d2,….,dn}Isi sj dengan 1Perbaharui di, dengan i = 1,2,3,…,n dengan: di(baru) = min{di(lama),dj+mj} |

***Priority Queue***

*Priority Queues* adalah pengembangan lebih lanjut dari konsep *Queue* yaitu sebuah *queue* yang setiap komponennya terdiri atas *key* dan *value*. *Priority queue* merupakan struktur data yang sifatnya sangat mirip dengan antrian, yaitu penghapusan/pengurangan anggota selalu dilakukan pada anggota antrian yang terdepan dan penambahan anggota selalu dilakukan dari belakang antrian berdasarkan prioritas anggota tersebut (anggota yang memiliki prioritas lebih besar selalu berada di depan anggota yang memiliki prioritas lebih rendah).

1. Studi Literatur

Berbagai percobaan yang dilakukan oleh peneliti terdahulu, hasil percobaan yang dikemukakan menunjukkan berbagai kesimpulan mengenai algoritma Dijkstra. Untuk kebanyakan masalah, algoritma Dijkstra secara umum (tapi tidak selalu) berhasil untuk mencari solusi optimal dalam masalah pencarian rute terpendek.

Pada percobaan yang dilakukan oleh Fakhri (2008) dalam karya ilmiahnya yang berjudul “Penerapan Algoritma Dijkstra dalam Pencarian Maximum Flow Problem“ algoritma Dijkstra disini mampu memberikan solusi optimal untk kasus Maximum Flow Problem. Akan tetapi, sama halnya seperti algoritma Greedy pada umumnya, terdapat masalah karena algoritma ini memang dirancang untuk menyelesaikan masalah ini, tidak seperti algoritma Dijkstra yang dirancang untuk mnyelesaikan masalah shortest path secara global. Hal ini terutama dikarenakan sulit melakukan penentuan urutan lintasan yang akan diperiksa, sama seperti algoritma Greedy.

Dalam hasil percobaan yang dilakukan oleh Asri Yuliana (2013) dalam tugas akhirnya yang berjudul “Implementasi Algoritma Dijkstra Pencarian Rute Terpendek Trans Jogja” dari hasil percobaan kasus yang telah dilakukan dapat diketahui bahwa hasil pencarian rute terpendek Trans Jogja ini sama dengan hasil pencarian yang dilakukan secara manual karna data jarak antar shelter diperoleh dari google maps dengan menginputkan koordinat shelter awal dan shelter tetangganya ini berarti sistem berhasil menerapkan algoritma Dijkstra dalam pencarian rute terpendek Trans Jogja. Penelitian ini berhasil merancang sebuah sistem yang mampu mencari rute terpendek Trans Jogja dan memvisualisasikan peta shelter dengan google maps. Dalam tugas akhir disini dapat disimpulkan bahwa penerapan algoritma Dijkstra dalam pencaruian rute terpendek Trans Jogja yang dapat membantu pengguna untuk mencari rute Trans Jogja yang akan digunakan.

Pada tugas akhir yang dibuat Ifatul Faizah (2010) yang berjudul “Rancang Bangun Perangkat Lunak Penentuan Rute Perjalanan Wisata di Malang Menggunakan Algoritma Dijkstra” sama seperti tugas akhir yang sudah disebutkan diatas, algoritma Dijkstra disini memberikan solusi jalur terpendek dari suatu perjalanan dimana seseorang pengarah jalan ingin menentukan jalur terpendek antara dua kota, berdasarkan beberapa jalur alternative yang tersedia, dimana titik tujuan hanya satu. Dijkstra disini memiliki akurasi yang baik dalam proses pembuatan atau dijitasi jalan akan mempengaruhi akurasi hasil perutean, semakin akurat data jalan, maka smakin akurat pula hasil perutean yang akan ditampilkan. Dalam tugas akhir yang dibuat oleh Ifatul, disini menggunakan software Delphi berbeda dengan skrispi yang sedang saya buat disini menggunakan Netbeans.

Sebelum telah dilakukan penelitian mengenai hal ini, salah satu penelitian tersebut yaitu penelitian tugas akhir yang dilakukan oleh Yadi Rusyad Nurdin yang berjudul “Implenetasi Algoritma Dijkstra dalam Menentukan Rute Terpendek pada Dua Titik Lokasi Wilayah Menteng” namun penelitian tersebut memilikin beberapa kekurangangan diantaranya :

* 1. Penelitian yang dilakukan hanya sebatas menemukan jalur terpendek berdasarkan jarak, sedangkan pada permasalahan kemacetan yang terjadi saat ini sistem tersebut tidak dapat diterapkan.
	2. Ruang lingkup penelitian sangat kecil, sehingga tidak bisa menjawab persoalan masyartakat yang cenderung lebih luas.
	3. Titik lokasi dijadikan input bersifat tetap, tidak bisa dilakukan penambahan, pengurangan dan pengubahan data oleh sistem.

Dari Imron Fauzi (2011) yang berjudul “Penggunaan Algoritma Dijkstra dalam Pencarian Rute Tercepat dan Rute Terpendek (studi kasus : Pada Jalan Raya antara Wilayah Blok M dan Kota)” dimana pengguna hanya menginput lokasi asal dan lokasi tujuan. Dalam tugas akhir disini penulis menyertakan factor kecepatan dan waktu tempuh perjalanan, ruang lingkup yang lebih luas, bersifat on line serta titik lokasi yang dijadikan input dapat dilakukan perubahan. Sehingga sistem yang dihasilkan lebih bermanfaat bagi pengguna serta dapat diakses kapanpun dan dimanapun. Adapun penggunaan algoritma Dijkstra dalam penelitian ini dilakukan karena algoritma ini sangat cocok dan efisien dalam mencari path dengan bobot terkecil pada sebuah graf.

Dalam percobaan yang dilakukan oleh Riyadhush Sholichin dkk yang berjudul “ Implementasi Algoritma Dijkstra dalam Pencarian Lintasan Terpendek Lokasi Rumah Sakit, Hotel dan Terminal Kota Malang Berbasis Web” dimana dari hasil penelitian dan pembahasan dapat disimpulkan jika pembuatan program Implementasi Algoritma Dijkstra dalam Pencarian Lintasan Terpendek Lokasi Rumah Sakit, Hotel dan Terminal Kota Malang Berbasis Web dilakukan dengan mengkonversi prosedur algoritma Dijkstra menjadi script program kemudian disertakan dalam web. Program yang dihasilkan di simpan dalam sebuah folder kerja yang ditempatkan dalam server web untuk dapat dijalankan.

**BAB 3. METODE PENELITIAN**

Penelitian ini menggunakan pendekatan kualitatif yang berguna untuk mendapatkan manfaat seperti yang dijelaskan pada bab sebelumnya. Subjek dalam penelitian ini adalah lokasi kampus Universitas Budi Luhur yang tersebar di Ciledug, Roxy dan Salemba.

Penelitian ini terdiri dari beberapa tahap kegiatan yang mencakup pengumpulan data, analisa data serta perancangan sistem. Adapun tahapan penelitian disampaikan secara rinci seperti berikut ini:

1. Pengumpulan data

Pengumpulan data merupakan langkah-langkah yang amat penting dalam penelitian dimana data yang terkumpul digunakan sebagai bahan analisa dan pengujian hipotesis yang telah dirumuskan. Kegiatan ini meliputi pengumpulan data tentang lokasi-lokasi kampus Budi Luhur dan juga melakukan wawancara dengan beberapa dosen dan juga supir (*driver*) unit transportasi kampus yang biasa melakukan perjalanan di antara lokasi kampus tersebut untuk mendapatkan data-data yang dibutuhkan seperti nama jalan yang biasa dilewati. Pengumpulan data juga dilakukan dengan melihat langsung (observasi) alternatif jalan yang ada serta mencatat panjang jalan untuk dilakukan analisa.

1. Pemrosesan (Analisa) Data

Tahap berikutnya adalah melakukan pemrosesan data, data-data yang ada diolah, dianalisa dan dikelompokkan sesuai dengan kebutuhan yang digunakan dalam proses selanjutnya yaitu pemrograman.

1. Perancangan sistem

Perancangan system meliputi perancangan basisdata, perancangan input, output serta pemrograman. Semua data yang telah dikumpulkan diolah dan dimasukkan ke dalam program.Setelah tahapan rancangan selesai dilakukan maka dilanjutkan dengan tahapan pengujian sistem. Hal ini untuk mengetahui apakah sistem sudah berjalan dengan baik dan sesuai dengan rancangan sistem yang telah dibuat. Pengujian sistem dilakukan dengan menggunakan metode white box testing dari sisi pengembang dan metode black box testing dari sisi pengguna.

**BAB 4. BIAYA DAN JADWAL PENELITIAN**

1. Anggaran Biaya

|  |  |  |
| --- | --- | --- |
| No | Jenis Pengeluaran | Biaya yang diusulkan (Rp) |
| 1. | Gaji dan Upah | 3.000.000 |
| 2. | Lain-lain | 1.482.000 |
| Jumlah | 4.482.000 |

1. Jadwal Kegiatan

|  |  |  |
| --- | --- | --- |
| No | Jenis Kegiatan | Bulan |
|  |  | April | Mei | Juni | Juli | Agust |
| 1.  | Pengajuan Proposal | X |  |  |  |  |
| 2. | Identifikasi, analisa dan mengolah data |  | X | X |  |  |
| 3. | Membuat design aplikasi |  | X | X | X |  |
| 4. | Membuat aplikasi |  | X | X | X |  |
| 5. | Testing aplikasi |  |  |  | X |  |
| 6. | Membuat laporan penelitian |  |  |  |  | X |

**DAFTAR PUSTAKA**

1. Huda, Miftakhul dan Bunafit Komputer. 2010. Membuat Aplikasi Database dengan Java, MySQL dan Netbeans. Jakarta : Elex Media Komputindo.
2. Peranginangin, Kasiman. 2006. Aplikasi Web dengan PHP dan MySQL. Yogyakarta : Andi.
3. Fakhri, “Penerapan Algoritma Dijkstra dalam Pencarian Solusi Maximum Flow Problem” Jurnal Skripsi Elektro dan Informatika, (Bandung, 2008).
4. Imron Fauzi, “Penggunakan Algoritma Dijkstra dalam Pencarian Rute Terpcepat dan Rute Terpdenek ” Jurnal Skripsi Sains dan Teknologi, (Jakarta, 2011).
5. Asri Yuliana, “Implementasi Algoritma Dijkstra Pencarian Rute Terpendek Trans Jogja” Jurnal Skripsi Sains dan Teknologi, (Yogyakarta, 2013).
6. Ifatul Faizah, “Rancangan Bangun Perangkat Lunak Penentuan Rute Perjalanan Wisata di Malang Menggunakan Algoritma Dijkstra” Jurnal Skripsi Sains dan Teknologi, (Malang, 2010).
7. Asri Yuliana, “Implementasi Algoritma Dijkstra Pencarian Rute Terpendek Trans Jogja” Jurnal Skripsi Sains dan Teknologi, (Yogyakarta, 2013).
8. Riyadhush Solichin, Mohammad Yasindan Lucky dan Tri Oktaviana, “Implementasi Algoritma Dijkstra dalam Pencarian Lintasan Terpendek Lokasi Rumah Sakit, Hotel dan Terminal Kota Malang Berbasis Web” Jurnal Teknologi Infomasi, Vol.1, No.1 2012, Universitas Negeri Malang

**LAMPIRAN-LAMPIRAN**

**ANGGARAN PENELITIAN**

|  |
| --- |
| 1.Honor |
| Pelaksana | Honor/jam (Rp) | Waktu (jam/minggu) | Minggu | Honor (Rp) |
| Ketua | 25.000 | 6 | 20 | 3.000.000 |
| Sub total (Rp) | 3.000.000 |
| 2.Lain-lain |
| Kegiatan | Justifikasi | Kuantitas | Harga Satuan(Rp) | Biaya (Rp) |
| Pembuatan Program | Jasa pembuatan system | 1 | 1.000.000 | 1.000.000 |
| Penggandaan Proposal(fotocopy) | Menggandakan proposal penelitian dari review sampai selesai revisi | 100 | 100 | 10.000 |
| Jilid Proposal 4 rangkap | Jilid sederhana | 4 | 3000 | 12.000 |
| Penggandaan Laporan untuk 4 rangkap(fotocopy) | Menggandakan laporan penelitian 2 rangkap | 600 | 100 | 60.000 |
| Softcover laporan LRPM, FTI, 2 Peneliti | Jilid softcover laporan penelitian | 4 | 25.000 | 100.000 |
| Konsumsi seminar lokal | Presentasi hasil penelitian | 10 | 30.000 | 300.000 |
| Sub Total (Rp.) | 1.482.000 |

**SUSUNAN ORGANISASI TIM PENELITI DAN PEMBAGIAN TUGAS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No** | **Nama/NIP** | **Program Studi** | **Alokasi Waktu (Jam/Minggu)** | **Uraian Tugas** |
| 1 | Noni Juliasari, M.Kom | Teknik Informatika | 6 | * Mengajukan proposal
* Identifikasi data
* Menganalisa data
* Merancang aplikasi
* Membuat laporan
 |
| 2 | Putri Rahayu | Teknik Informatika | 6 | * Membangun aplikasi
* Testing aplikasi
 |

**BIODATA TIM PENELITI**

1. **Identitas Diri**
	1. Nama Lengkap (dengan gelar) : Noni Juliasari, M.Kom
	2. Jenis Kelamin : Perempuan
	3. Jabatan Fungsional : Dosen Tetap
	4. NIP : 020018
	5. Tempat, Tanggal Lahir : Jakarta, 5 Juli 1980
	6. E-mail : noni.juliasari@budiluhur.ac.id
	7. Nomor Handphone : 021 94798562
	8. Alamat : Perum.Graha Karang Tengah Blok F78

 Rt.005/03, Pd.Pucung, Kr.Tengah, TGR

1. **Riwayat Pendidikan**

|  |  |  |  |
| --- | --- | --- | --- |
|  | S1 | S2 | S3 |
| Nama Perguruan Tinggi | Universitas Budi Luhur | Universitas Indonesia |  |
| Bidang Ilmu | Teknik Informatika | Teknologi Informasi |  |
| Tahun Masuk Lulus | 1998-2002 | 2004-2006 |  |

1. **Pengalaman Penelitian (5 tahun terakhir)**

|  |  |  |  |
| --- | --- | --- | --- |
| **No.** | **Tahun** | **Judul Penelitian** | **Pendanaan** |
| **Sumber** | **Jumlah(Rp)** |
| 1 | 2011 | Optimasi Mapping Kelompok Belajar Mahasiswa Baru Dengan Algoritma GenetikStudi Kasus : Fakultas Teknologi Informasi, Universitas Budi Luhur | Univ.Budi Luhur | 9.268.000 |
| 2 | 2012 | Rancangan Sistem Pelayanan Terpadu (One Stop Service) Dengan Pendekatan Knowledge Capture Pada Data MahasiswaStudi Kasus : Universitas Budi Luhur | Univ.Budi Luhur | 7.524.000 |
| 3 | 2013 | Perancangan Data Warehouse Untuk Pengukuran Kinerja Pengajaran Dosen Studi Kasus : Fakultas Teknologi InformasiUniversitas Budi Luhur | Univ.Budi Luhur | 7.360.000 |

 **Jakarta, 20 Maret 2014**

**(Noni Juliasari, M.Kom)**