PERBANDINGAN ALGORITMA *BREADTH FIRST SEARCH* DAN *DIJKSTRA* UNTUK PENENTUAN RUTE TERPENDEK PENGIRIMAN BARANG UNILEVER

Anwari[1], Hozairi[2]
[1)] Program Studi Sistem Informasi, Universits Islam Madura, Pamekasan, Indonesia
[2)] Program Studi Teknik Informatika, Universits Islam Madura, Pamekasan, Indonesia
anwari.uim@gmail.com, dr.hozairi@gmail.com

ABSTRAK

Algoritma *Breadth First Search* dan *Dijkstra* merupakan algoritma untuk menentukan rute terpedek dari suatu verteks ke verteks yang lainnya pada suatu graph yang berbobot dimana jarak antar verteks adalah bobot atau nilai dari tiap edge atau arc pada graph tersebut. Penelitian bertujuan untuk membandingkan algoritma *Breadth First Search* dan *Dijkstra* dengan studi kasus untuk menyelesaikan permasalahan rute pengiriman barang unilever di Kabupaten Pamekasan. Percobaan yang telah dilakukan dengan menggunakan algoritma *Breadth First Search* menghasilkan 0,04 % lebih cepat dari *Dijkstra* sedangkan *Dijkstra* 0,06 % lebih lama dari pada *Breadth First Search* dengan hasil akumulasi jarak lebih besar

Keyword : *Breadth First Search, Dijkstra, Rute*

1. PENDAHULUAN

Unilever adalah sebuah perusahaan yang memiliki tugas untuk menyalurkan produk atau barang ke beberapa distributor sesuai dengan schedule dan rute yang telah ditetapkan sesuai order. Permasalahan yang sering dialami oleh Unilever adalah keterlambatan dalam pengiriman. Keterlambatan ini dapat menyebabkan distributor tidak dapat melayani permintaan konsumen secara maksimal. Penyebab dari keterlambatan ini adalah karena Unilever tidak menganalisa rute terpendek untuk mengirim barang ke setiap konsumen yang wilayahnya menyebar.

Studi kasus dalam penelitian ini adalah proses distribusi Unilever ke konsumen di Kabupaten Pamekasan. Banyaknya rute menuju konsumen membuat layanan unilever sering mengalami keterlambatan, hal tersebut sangat dipengaruhi faktor karena belum mengetahui rute jalan di Kota Pamekasan dan kesulitan menentukan rute menuju konsumen yang optimal (terpendek) untuk mencapai ke semua tujuan sesuai jadwal. Oleh karena itu, dibutuhkan solusi yang dapat mengatasi penentuan rute terpendek atau tercepat untuk mencapai tempat-tempat tujuan yang telah dijadwalkan.

Penelitian ini bertujuan membuat sebuah sistem aplikasi yang dapat menentukan jarak optimal yang nantinya dapat ditempuh oleh para sales atau pengirim barang dengan mengimplentasikan sebuah algoritma yang telah dibuat kedalam sistem tersebut yang berbasis Web. Algoritma *Breadth First Search* dan *Dijkstra* merupakan salah satu algoritma yang digunakan untuk menyelesaikan masalah dalam penentuan rute terpendek. Untuk menyelesaikan masalah penentuan rute terpendek tersebut kita dapat mempresentasikan masalah yang ada menjadi struktur *graph*, dimana titik menyatakan kota dan sisi menyatakan jalur yang menghubungkan dua buah kota. Setiap sisi yang ada diberikan bobot yang menyatakan jarak antara kedua kota tersebut.

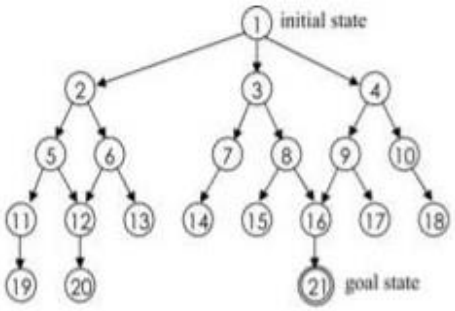
Algoritma *Breadth First Search* dan *Dijkstra* mempunyai ide yang hampir mirip maka dari itu nantinya akan di lakukan perbandingan dengan alasan mencari algoritma yang paling tepat dalam memecahkan masalah penentuan rute terpendek dalam pengiriman barang, yang termasuk pada masalah *Travelling Salesman Problems.*

Sistem yang dibangun berupa sebuah sistem yang dapat memberikan suatu solusi bagi para *sales* atau pengirim barang dengan memberikan hasil perhitungan jarak terpendek yang nantinya akan menjadi pengambilan keputusan bagi sales untuk menentukan ruter jalan mana yang lebih optimal untuk dilewati.

2. TINJAUAN PUSTAKA

2.1. Algoritma *Breadth First Search* (BFS)

Algoritma *Breadth First Search* (BFS) dapat ditelusuri dengan menggunakan daftar (list), open, dan closed, untuk menelusuri gerakan pencarian di dalam ruang keadaan. Prosedur untuk *Breadth First Search* dapat dilihat di Gambar 1.



Gambar 1. *Algoritma BFS dalam Graph*

Berdasarkan Gambar 1, state 21 merupakan tujuannya (goal) sehingga bila ditelusuri

menggunakan prosedur Breadth First Search, diperoleh:

1) Open = [1]; closed = [ ].
2) Open = [2, 3, 4]; closed = [1].
3) Open = [3, 4, 5, 6]; closd = [2, 1].
4) Open = [4, 5, 6, 7, 8]; closed = [3, 2, 1].
5) Open = [5, 6, 7, 8, 9, 10]; closed = [4, 3, 2, 1].
6) Open = [6, 7, 8, 9, 10, 11, 12]; closed = [5, 4, 3, 2, 1].
7) Open = [7, 8, 9, 10, 11, 12, 13] (karena 12 telah di-open).
8) closed = [6, 5, 4, 3, 2, 1].
9) Open = [8, 9, 10, 11, 12, 13, 14]; closed = [7, 6, 5, 4, 3, 2, 1].
10) Dan seterusnya sampai state 21 diperoleh atau open = [ ].

Ada beberapa keuntungan menggunakan algoritma *Breadth First Search*, sebagai berikut: (a) tidak akan menemui jalan buntu, (b) jika ada satu solusi maka *Breadth First Search* akan menemukannya, (c). Jika ada lebih dari satu solusi maka solusi minimum akan ditemukan.

Namun disamping keutungan diatas ada tiga persoalan utama berkenaan dengan *Breadth First Search* yaitu : (a) Membutuhkan memori yang lebih besar karena menyimpan semua node dalam satu pohon, (b) membutuhkan sejumlah besar pekerjaan, khususnya jika lintasan solusi terpendek cukup panjang, karena jumlah node yang perlu diperiksa bertambah secara eksponensial terhadap panjang lintasan, (c) tidak relevannya operator akan menambah jumlah node yang harus diperiksa.

Oleh karena itu proses *Breadth First Search* mengamati node di setiap level graph sebelum bergerak menuju ruang yang lebih dalam maka mula-mula semua keadaan akan dicapai lewat lintasan yang terpendek dari keadaan awal. Oleh sebab itu, proses ini menjamin ditemukannya lintasan terpendek dari keadaan awal ke tujuan (akhir). Lebih jauh karean mula-mula semua keadaan ditemukan melalui lintasan terpendek sehingga setiap keadaan yang ditemui pada kali kedua didapati pada sepanjang sebuah lintasan yang sama atau lebih panjang. Kemudian, jika tidak ada kesempatan ditemukannya keadaan yang identik pada sepanjang lintasan yang lebih baik maka algoritma akan menghapusnya.

2.2. Algoritma *Dijkstra*

Algortima ini ditemukan oleh Adsger W. Dikstra yang merupakan salah satu varian bentuk algoritma popular dalam pemecahan persoalan terkait dengan masalah optimasi dan bersifat sederhana. Algortima ini menyelesaikan maslah mencari sebuah lintasan terpendek (sebua lintasan yang mempunyai panjang minimum dari verteks a ke z dalam graph berbobot, bobot tersebut adalah bilangan positif jadi tidak dapat dilalui oleh node negatif, namun jika terjadi demikian, maka penyelesaian yang diberikan adalah infiniti (Tak Hingga).

Algoritma *Dijkstra* melibatkan pemasangan label pada verteks. Misalkan L(v) menyatakan label dari verteks v. Pada setiap pembahasan, beberapa verteks mempunyai label sementara dan yang lain mempunyai label tetap. Misalnya T menyatakan himpunan verteks yang mempunyai label sementara. Dalam menggambarkan algoritma tersebut verteks-verteks yang mempunyai label tetap akan dilingkari. Selanjutnya, jika L(v) adalah lavel tetap dari verteks v maka L(v) merupakan panjang lintasan terpendek dari a ke v. Sebelumnya semua verteks mempunyai label sementara. Setiap iterasi dari algoritma tersebut mengubah status satu label dari sementara ke tetap. Pada bagian ini L(z) merupakan panjang lintasan terpendek dari a ke z. Pada algoritma Dijkstra node digunakan, karena algoritma Dijkstra menggunakan graph berarah untuk penentuan rute listasan terpendek.

Berikut ini dijelaskan langkah-langkah pada algoritma *Dijkstra*, untuk menghitung jarak terpendek semua simpul terhadap suatu simpul A maka :

1. Tetapkan jarak semua simpul terhadap simpul A, yaitu *infinity* atau tak-hingga untuk simpul yang lain dan 0 untuk simpul A.
2. Tandai semua simpul dengan status belum dikunjungi. Jadikan simpul awal sebagai simpul terkini.
3. Untuk node terkini, hitung jarak semua tetangga simpul ini dengan menghitung jarak (dari awal simpul). Misalnya, jika saat ini node (C) memiliki jarak dari simpul A sebesar 6, dan sisi yang menghubungkannya dengan node lain (B) adalah 2, jarak ke B melalui C akan menjadi 6 + 2 = 8. Jika jarak ini kurang dari jarak yang sebelumnya (takhingga di awal) maka nilai jarak simpul B dengan simpul A akan berubah.
4. Setelah selesai mengecek semua tetangga dari simpul terkini, simpul terkini ditandai dangna status sudah dikunjungi.
5. Mengulang langkah tiga hingga lima, hingga semua simpul telah dikunjungi.

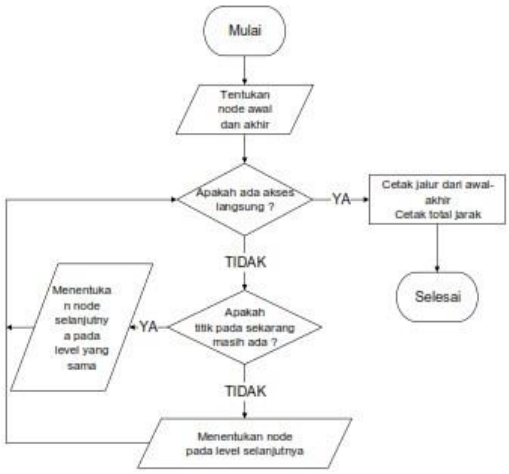
3. METODE PENELITIAN

3.1. Flowchart Algoritma BFS

Dari flowchat di atas, dapat dijelaskan langkah-langkah proses algoritma yang terjadi adalah sebagai berikut. Sebelum melakukan proses pencarian tentunya node awal dan akhir sudah ditentukan, algoritma mengecek satu persatu dari level yang paling dekat terdahulu ke level yang berada dibawahnya, kemudian menyimpannya kedalam tabel cek dan mengecek apakah node yang dilalui adalah sulusi, jika node yang dilalui adalah solusi maka program akan berhenti dan mencetak jalur yang dilalui dengan total jarak keseluruhan, namun apabila bukan solusi maka algoritma

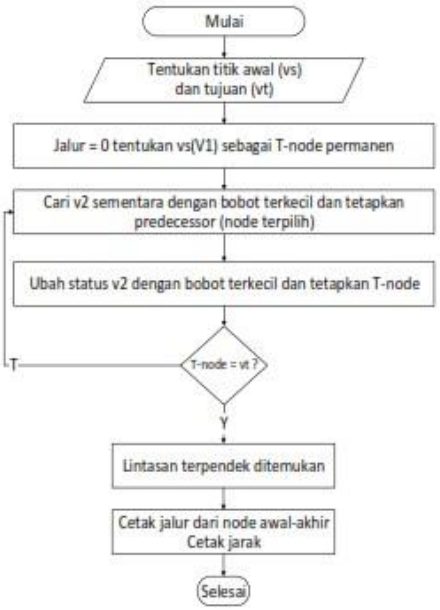
melajutkan pengecekan ke simpul atau node pada level selanjunya dan menyipan kembali kedalam tabel cek sampai tujuan akhir tercapai.

Proses pengecekan data dilakukan dengan cara pengambilan data dari tabel rute menggunakan fungsi query sql dan ditampung kedalam tabel cek sebagai media penampungan sementara. Untuk memenuhi proses algoritma seperti pada flowchart diatas maka, dibutuhkan beberapa tabel sebagai media penyimpanan dan pemprosesan data, yang diataranya adalah sebagai berikut, (tabel titik, tabel rute, dan tabel cek)



Gambar 2. Flowchart Algoritma *Breadth First Search*

3.2. Flowchat Algoritma Dijkstra



Gambar 3. Flowchart Algoritma *Dijkstra*

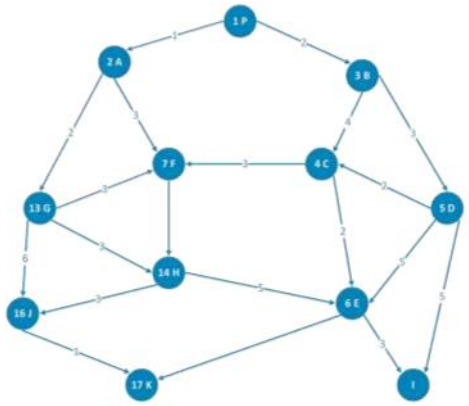
Berikut dapat dijelaskan proses yang terjadi dari flowchat pada algoritma dijkstra diatas dari awal sampai akhir. Pertama node asal (*Vs*) dan tujuan (*Vt*) ditentukan, kemudian node asal atau (*Vt*) dengan jarak sama dengan 0 ditentuka sebagai node permanen keberangkatan, kemudian algoritma mencari node selanjutnya dengan jarak minimum (*V2*) dan menetapkan sebagai node terpilih, simpan jalur yang dilalui kedalam *tabel cek* , maka tandai (V 2) ditetapkan sebagai node permanen dengan asumsi (V2) mempunyai jarak terkecil.

Dari hasil itu program kembali mengecek apakah sama dengan sulusi, jika iya lintasan terpendek ditentukan dan cetak jalur dari awal ke akhir serta cetak total jarak keseluruhan. Jika tidak sama dengan solusi maka kembali looping ke bobot yang terkecil sampai tujuan tercapai. Agar alur dalam flowchart diatas dapat terealisasikan maka diperlukan beberapa tabel database sebagai penyimpanan data yang akan diimputkan nantinya, diantaranya adalah, tabel titik, tabel rute, dan tabel cek. Dengan fungsi-fungsi dari masing-masing tabel tersebut seperti yang telah diuraikan dalam perancangan database diatas.

4. HASIL PENELITIAN

4.1. Proses Pencarian Rute

Proses pengimplementasian dari algoritma yang digunakan pada sistem ini. Berikut form proses pencarian dari masing algoritma yang diterapkan, dengan mengilustrasikan lintasan dan titik kedalam bentuk simulasi graph seperti gambar dibawan ini.



Gambar 4. Gambar Simulasi *Graph* Rute Pengiriman

Gambar 4 menjelasakan tentang proses simulasi graph yang merupakan ilustrasi rute dari beberapa titik pengiriman barang yang akan dilewati nantinya, dalam graph dijabarkan titik adalah tempat dan, sisi adalah jalan. Berikut keterangan dari titik pada Gambar 4.

Tabel 1. Keterangan Titik-titik pada *Graph*

Titik	Keterangan
P	Jl. Pintu Gerbang
A	Jl. Pademawu
B	Jl Trunojoyo
C	Jl. Raya Galis
D	Jl. Raya Larangan
E	Jl. Jelmak
F	Jl. Larangan Badung
G	Jl. Raya Palengaan
H	Jl. Raya Proppo
I	Jl. Raya Blumbungan
J	Jl. Raya Kanginan
K	Jl. Raya Tampung

Gambar 5. Form Proses Pencarian BFS

Gambar 6. Form Hasil Pencarian BFS

Iterasi Pengelompokan Rute Yang Berhubungan Dengan Titik **F** Program mencari secara keseluruhan atau melebar kemana rute terpendek ke titik **F**, pertama-tama program mencari titik yang mungkin berhubungan dengan titik **F**. Dari hasil pencarian tersebut disimpan ke tabel cek dan beri nilai 1 untuk titik yang sudah terpenuhi. Untuk lebih jelasnya berikut proses yang terjadi di database selama proses iterasi pertama.

id	awal	akhir	jarak	status	status2	urut	urut2
1	2	7	3	1	0	0	0
2	13	7	3	1	0	0	0
3	4	7	3	1	0	0	0
4	1	2	1	1	0	0	0
5	2	13	2	1	0	0	0
6	3	4	4	1	0	0	0
7	5	4	2	1	0	0	0
9	1	2	1	1	0	0	0
10	1	3	2	1	0	0	0
11	3	5	3	1	0	0	0
12	1	3	2	1	0	0	0

Gambar 7. Proses Iterasi Pengecekan *BFS*

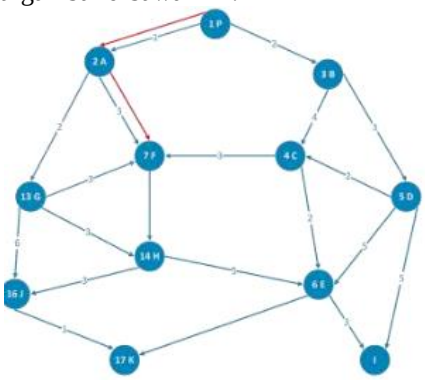
Dari hasil iterasi pertama pada field status terpenuhi secara keseluruhan pada 12 titik yang ada, sepertin yang sudah dijelaskan diatas. Pada iterasi kedua di field status2 tidak semua titik tepenuhi, karena disini merupakan proses pengecekan rute terpendek dari awal ke tujuan.

Pada iterasi kedua program mengecek P➔B = 2 km dari titik B program ke titik C sehingga lintasanya menjadi P➔B➔C = 6 km. Dari titik C ada pecabangan yaitu F (C-F) dan ke E (C-E), karena titk C➔F sudah dicek dan C➔E tidak ada jalur menuju tujuan dan akan semakin jauh, maka program kembali looping ke awal yaitu dari titi, P➔A = 1 km, dari titik A➔G sehingga lintasanya menjadi P➔A➔G = 3 km, dari titik G➔F dengan jarak 3 km dan lintasan bertambah menjadi P➔A➔G➔F dengan total jarak keseluruhan 6 km. Karena pada proses percobaan pertama ini menggunakan algoritma *Breadth First Search* yang mencari lintasan terpendek ke titik tujuan, maka program melakukan pengecekan kembali dari titik P➔A = 1 km dari titik A ke titik F dan lintasannya menjadi P➔A➔F dengan total jarak 4 km.

Berikut hasil pengecekan dari yang telah di uraikan diatas :
P➔A = 1
P➔A➔G = 3 km
P➔A➔G➔F = 6 km
P➔A➔F = 4 km

Dari lintasan yang terbentuk diatas maka akan diurutkan rute yang menuju ke tujan dari yang terbesar ke terkecil, sehinggan menghasilkan rute terpendek dari P□F adalah P□A□F dengan total jarak 4 km.

Berikut gambaran ilustrasi lintasan dalam bentuk simulasi graph pada proses pencarian dengan ***BFS***, seperti gambar dibawah ini.



Gamabr 8. Proses Pencarian *Dijkstra*

Gambar 9. Form Proses Pencarian *Dijkstra*

Iterasi Pengecekan Rute Yang Berhubungan Dengan Titik Tujuan F Untuk proses pada algoritma dijkstra diambil terlebih dahulu rute-rute yang memungkinkan ada jalur ke titik tujuan F, seperti pada gambar dibawah ini.

id	awal	akhir	jarak	status	status2	urut	urut2
1	2	7	3	1	0	0	0
2	13	7	3	1	0	0	0
3	4	7	3	1	0	0	0
4	1	2	1	1	0	0	0
5	2	13	2	1	0	0	0
6	3	4	4	1	0	0	0
7	5	4	2	1	0	0	0
9	1	2	1	1	0	0	0
10	1	3	2	1	0	0	0
11	3	5	3	1	0	0	0
12	1	3	2	1	0	0	0

Gambar 10. Terasi Pengecekan *Dijkstra*

Pada gambar diatas nilai 1 adalah nilai dimana penandaan titik yang terdapat jalur ke titik tujuan sudah terpenuhi dan nilai 0 belum terpenuhi. Berikut keterangan dari gambar 4.12 diatas dalam bentuk tabel dibawah ini.

id	awal	akhir	jarak	status	status2	urut	urut2
1	2	7	3	1	2	0	0
2	13	7	3	1	1	3	0
3	4	7	3	1	0	0	0
4	1	2	1	1	1	1	0
5	2	13	2	1		2	0
6	3	4	4	1	0	0	0
7	5	4	2	1	0	0	0
9	1	2	1	1	2	0	0
10	1	3	2	1	2	0	0
11	3	5	3	1	0	0	0
12	1	3	2	1	2	0	0

Gambar 11. Iterasi Penentuan Rute *Dijkstra*

Pertama program mengecek mana simpul yang terpendek antara simpul **P➔B** dengan jarak 2 km dan **P➔A** dengan jarak 1km, karena ini menggunakan algoritma Dijkstra yang mengambil konsep jarak terpendek dari titik keberangkatan ketitik selanjutnya maka otomatis program akan memilih rute **P➔A** dengan jarak 1 km, dan tetapkan simpul **A** sebagai node terpilih dan menjadi node keberangkatan selanjutnya. Dari simpul **A** ada 2 percabangan yaitu **A➔G** dengan jarak 2 km dan**A➔F** yang merupakan tujuan kita, dengan jarak 3 km, maka otomatis program akan memilih jalur **A➔G** dengan jarak 2 km dan tetapkan **G** sebagain node terpilih berikutnya, dari simpul **G** ada tiga percabangan yaitu :

G➔J = 6 km
G➔H = 3 km
G➔F = 3 km

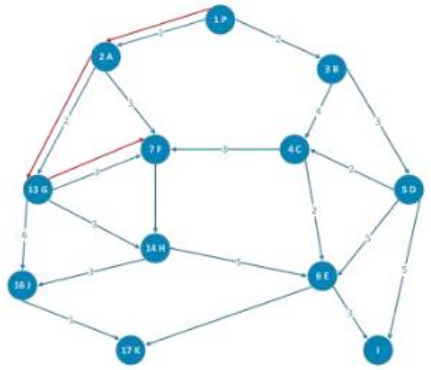
Karena dari kedua cabang **G➔J** dan **G➔H** tidak terdapat jalur ke tujuan, simpul **F** maka program memilih **G➔F** dengan jarak 3 km, dan menghasilkan rute **P➔B➔E➔F** dengan jarak 6 km. Berikut proses terbentuknya lintasan dari simpul awal ke tujuan.

P➔A = 1 km
P➔A➔F = 4 km
P➔A➔G = 3 km
P➔A➔G➔F = 6 km

Karena pada pecobaan kedua menggunakan algoritma dijkstra maka program akan menampilkan atau memilih lintasan P➔A➔G➔F = 6 km, inilah konsep algoritma dijkstra yang memilih jalur terpendek diantara beberapa simpul atau titik, dapat

Karena pada pecobaan kedua menggunakan algoritma dijkstra maka program akan menampilkan atau memilih lintasan P□A□G□F = 6 km, inilah konsep algoritma dijkstra yang memilih jalur terpendek diantara beberapa simpul atau titik, dapat membuat seseoran berputar lebih jauh padahal tujuan sudah didepan mata.

Berikut gambaran ilustrasi lintasan dalam bentuk simulasi graph pada proses pencarian dengan ***Dijkstra***, seperti gambar dibawah ini.



Gambar 12. Ilustrasi Proses Dijkstra Pada Graph

Dari hasil percobaan yang dilakukan untuk menentukan perbedaan jarak lintasan atau rute terpendek pada algoritma *Breadth First Search* dan *Dijkstra* untuk melakukan pengujian dipilih beberapa titik tujuan dengan 12 titik rute, dan lintasan yang sama.

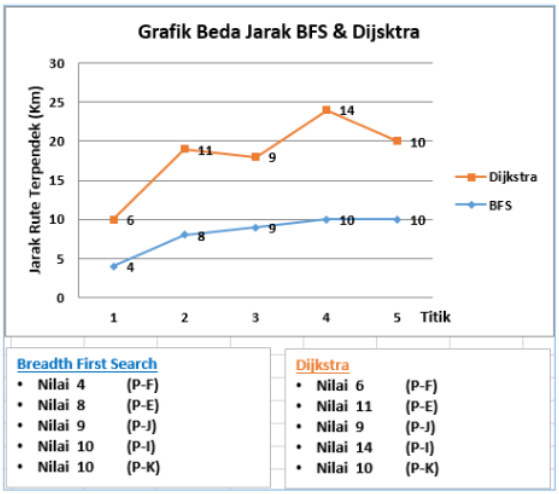
Berikut hasil percobaan yang dilakukan pada masing-masing algoritma dapat kita lihat pada tabel 4.4 dengan perbedaan jarak lintasan dan rute yang harus dilalui.

Tabel 2. Perbandingan Hasil Jarak Kedua Algoritma

No	Dari - Ke	Rute		Jarak (Km)	
		Bfs	Dijkstra	Bfs	Dijkstra
1	P – F	P-A-F	P-A-G-F	4	6
2	P – E	P-B-C-E	P-A-G-H-E	8	11
3	P – J	P-A-G-J	P-A-G-H-J	9	9
4	P – I	P-B-D-I	P-A-G-H-E-I	10	14
5	P – K	P-A-G-J-K	P-A-G-H-J-K	10	10

Dari data pada tabel 4.4 diatas dapat kami sajikan kedalam bentu grafik perbandingan jarak lintasan

yang diperoleh dari masing-masing algoritma, seperti pada gambar dibawah ini.



Gambar 13. Grafik Perbandingan Jarak

5. KESIMPULAN

Dari penelitian dan implementasi mengenenai perbandingan algoritma *Breadth First Search* dan *Dijkstra* berdasarkan jarak lintasannya, algoritma *Breadth First Search* menghasilkan jarak yang lebih kecil dengan rute atau lintasan yang terbentuk lebih sedikit (P-A-F) seperti pada percobaan 1, dari titik **P** yaitu (*kantor pusat*) ke **F** yaitu (*larangan badung*) jumlah jarak yang diperoleh adalah sebesar 4 km, sedangkan pada algoritma Dijkstra jarak yang diperoleh adalah 6 km dengan lintasan yang terbentuk lebih banyak (P-A-G-F), algoritma *Dijkstra* melakukan proses pencarian dari titik awal ketitik selanjutnya yang mempunyai bobot paling kecil sehingga membuat akumulasi total jarak keseluruhan menjadi lebih besar dari pada *Breadth First Search* dan mengasilkan rute lebih banyak yang harus dikunjungi seseorang. Bread*th First Search* mencari rute tercepat dari titik awal ke titik tujuan tanpa membuat seseorang berputar-putar lebih lama sehingga menghasilkan jarak yang lebih kecil dan rute yang harus dilewati lebih sedikit.

Jadi dapat kita ambil kesimpulan bahwa algoritma Breadth First Search lebih baik untuk digunakan pada pengiriman barang dengan satu tujuan, sedangkan algoritma *Dikjstra* lebih baik digunakan untuk banyak tujuan dengan banyak tempat yang harus dikunjungi. Berdasarkan data dari jumlah jarak yang dihasilkan dengan menggunakan Breadth First Search 0,04 % lebih cepat dari Dijkstra sedangkan Dijkstra 0,06 % lebih lama dari pada Breadth First Search dengan hasil akumulasi jarak lebih besar.

DAFTAR PUSTAKA

[1]. Ahmad Tanzeh. 2011. *Metodologi Penelitian Praktis*, Teras. Yogyakarta.

[2]. Yuliawati, Yuyun R. *2002. Pencarian Jarak Terpendek Menggunakan Metode BFS dan Hill Climbing. UNIKOM.*

[3]. Juliansya, Agung. 2010, *Ebook Struktur Data,* Universitas International Batam. Batam.

[4]. Noviansah, Nita. 2009, *Makalah Teori Graph Tavelling Salesman Problem (TSP)*, Institute Teknologi Nasional. Malang.

[5]. Syahriza Lubis, Henny. 2009, *Perbandingan Algoritma Dijkstra dan Greedy Untuk Lintasan Terpendek*, USU Universitas Sumatera Utara Medan.

[6]. Munir, Rinaldi. 2009, *Makalah Teori Graph*, Institute Teknologi Bandung. Bandung.

Inggiantowi, Hafid. 2008, *Perbandingan Algoritma Penelusuran DFS dan BFS pada Graph serta Aplikasinya*, Institute Teknologi Bandung. Bandung