 ISSN: 2684-8473

Implementasi algoritma *floyd warshall* pada pencarian lokasi agen bus, *tour and travel*, dan rental mobil berbasis android

Deny Wiria Nugraha[1], Amriana[2], Asri Arif[3]

Teknologi Informasi, Fakultas Teknik, Universitas Tadulako, Palu, Indonesia
[1] deny.wiria.nugraha@gmail.com; [2] amrianaa23@gmail.com; [3] asriarif2@gmail.com

INFORMASI ARTIKEL

Histori Artikel
Diterima : 15 Februari 2020
Direvisi : 22 Februari 2020
Diterbitkan : 4 April 2020

Kata Kunci:
Algoritma Floyd Warshall
Pencarian Lokasi
Agen Bus
Tour and Travel
Rental Mobil
Android

ABSTRAK

Penelitian ini bertujuan untuk membangun sebuah sistem yang dapat memberikan informasi lokasi agen bus, tour and travel, dan rental mobil yang ada di Kota Palu, dan sekaligus memberikan petunjuk jalur terpendek dengan menggunakan Algoritma Floyd warshall berbasis Android. Algoritma Floyd warshall adalah salah satu varian dari pemrograman dinamis, yaitu suatu metode yang melakukan pemecahan masalah dengan memandang solusi yang akan diperoleh sebagai suatu keputusan yang saling terkait, algoritma ini menghitung bobot terkecil dari semua jalur yang menghubungkan sebuah pasangan titik, dan melakukannya sekaligus untuk semua pasangan titik. Perhitungan pencarian jalur terpendek menggunakan algoritma Floyd Warshall langkah pertama menentukan jalur yang akan dilalui dan nilai bobotnya dibuat ke dalam tabel atau matriks, lakukan perhitungan periterasi sampai dengan iterasi terakhir, lakukan berulang sebanyak vertexnya, dan menentukan jalur terpendek berlaku dari hasil iterasi terakhir. Proses perhitungan algoritma Floyd Warshall di implementasikan ke dalam sistem yang telah dibuat menggunakan aplikasi Delphi 10.3 CE dengan memanfaatkan prosedur algoritma Floyd Warshall, Penelitian ini menggunakan data sebanyak 120 lokasi agen bus, tour and travel dan rental mobil di kota Palu berbasis dengan mengimplementasikan algoritma Floyd Warshall untuk mencari jalur terpendek yang diterapkan pada Delphi 10.3 CE. Pada Delphi 10.3 CE peneliti merasakan kemudahan dan lebih ringan dalam pengerjaan algoritma Floyd Warshall untuk pencarian jalur terpendek.

I. Pendahuluan

Agen bus, *tour and travel* dan rental mobil adalah biro usaha perjalanan dan jasa transportasi ke suatu wilayah melalui darat dan juga udara, sekarang ini agen bus, *tour and travel* dan rental mobil sangatlah banyak karena kebutuhan manusia yang ingin menggunakan jasa agen bus, *tour and travel* dan rental mobil ini, yang mana mereka ingin pergi ke suatu wilayah yang jauh membutuhkan transportasi melalui udara maka dari itu dibutuhkan agen *tour and travel* untuk membeli tiket, ataupun ingin memerlukan jasa perjalanan melalui darat bisa dengan memakai jasa agen bus melakukan perjalanan darat ke suatu wilayah dan yang terakhir agen rental mobil jasa rental mobil adalah penyewaan berupa kendaraan roda empat. Perkembangan agen bus, *tour and travel* dan rental mobil sangatlah banyak diberbagai daerah di Indonesia bahkan bisa dibilang semua wilayah pasti mempunyai agen bus, *tour and travel* dan rental mobil didaerahnya masing–masing.

Di Sulawesi Tengah khususnya Kota Palu agen bus, *tour and travel* dan rental mobil sangatlah banyak dan perkembangannya sangatlah pesat, akan tetapi banyak masyarakat Kota Palu terkadang bingung mencari lokasi agen bus, *tour and travel* dan rental mobil berada, maka untuk membantu bagi pengguna jasa bus, tour and travel dan rental mobil yang ada di Kota Palu dibangunlah sistem pencarian jalur terpendek lokasi agen bus, tour and travel dan rental mobil berbasis *mobile* dengan *platform Android* menggunakan algoritma *Floyd Warshall*. Sistem ini nantinya akan memberikan informasi tentang lokasi bus, *tour and travel* dan rental mobil yang ada di Kota Palu. Tidak hanya itu, sistem ini juga dapat menampilkan peta yang dapat membantu menemukan jalur terpendek menuju lokasi agen bus, *tour and travel* dan rental mobil.

II. Material dan Metode

A. Graf

Graf adalah kumpulan dari simpul dan busur yang secara matematis dinyatakan sebagai G = (V, E), dimana: G = *graph*, V = simpul atau vertex, E = busur atau edge [1], [2]. *Graph* terdiri dari *graph* berbobot dan tak berbobot. Kemudian, persoalan mencari jalur terpendek didalam graf merupakan salah satu persoalan optimisasi. Graf yang digunakan dalam pencarian jalur terpendek adalah graf berbobot (*weighted graph*), yaitu graf yang setiap sisinya diberikan suatu nilai atau bobot [3]. Kata terpendek berbeda-beda maknanya bergantung pada tipikal persoalan yang akan diselesaikan. Namun secara umum terpendek berarti meminimisasi bobot pada suatu lintasan dalam graf [4].

B. Algoritma Floyd Warshall

Algoritma *Floyd Warshall* adalah sebuah algoritma analisis graf untuk mencari bobot minimum dari graf berarah [5], [6]. Dalam pengertian lain algoritma *Floyd-Warshall* adalah suatu metode yang melakukan pemecahan masalah dengan memandang solusi yang akan diperoleh sebagai suatu keputusan yang saling terkait. Artinya solusi-solusi tersebut dibentuk dari solusi yang berasal dari tahap sebelumnya dan ada kemungkinan solusi lebih dari satu. Algoritma *Floyd Warshall* memiliki input graf berbobot (V,E), yang berupa daftar titik (*node*/*verteks* V) dan daftar sisi (*edge* E).

Jumlah bobot sisi- sisi pada sebuah lintasan adalah bobot sisi tersebut. Sisi pada E diperbolehkan memiliki bobot negatif, akan tetapi tidak diperbolehkan bagi graf ini untuk memiliki siklus dengan bobot negatif. Algoritma ini menghitung bobot terkecil dari semua sisi yang menghubungkan sebuah pasangan titik dan melakukannya sekaligus untuk semua pasangan titik. Prinsip yang dipegang oleh algoritma *Floyd Warshall* adalah prinsip optimalitas, yaitu jika solusi total optimal, maka bagian solusi sampai suatu tahap (misalnya tahap ke-i) juga optimal [7].

C. Model Pengembangan Perangkat Lunak

Model *waterfall* ini sebenarnya adalah "*Linear Sequential Model*", yang sering disebut juga dengan "*classic life cycle*" atau model *waterfall*. *Waterfall* atau air terjun adalah model yang dikembangkan untuk pengembangan perangkat lunak, membuat perangkat lunak. Model berkembang secara sistematis dari satu tahap ke tahap lain dalam model seperti air. Gambar 1 menunjukkan alur dari model pengembangan *waterfall.*

1) Requirements Definition

Tahap ini diawali dengan mencari kebutuhan dari keseluruhan yang akan diaplikasikan ke dalam bentuk *software.* Data yang dikumpulkan adalah data lokasi dan informasi agen bus *tour and travel* dan rental mobil di Kota Palu.

2) System and Software Design

Tahap ini dilakukan sebelum melakukan *coding*. Tahap ini bertujuan untuk memberikan gambaran apa yang seharusnya dikerjakan dan bagaimana tampilannya. Penulis membuat tampilan sistem dengan menggunakan aplikasi *Adobe Photoshop* CS5.

3) Implementation and Unit Testing

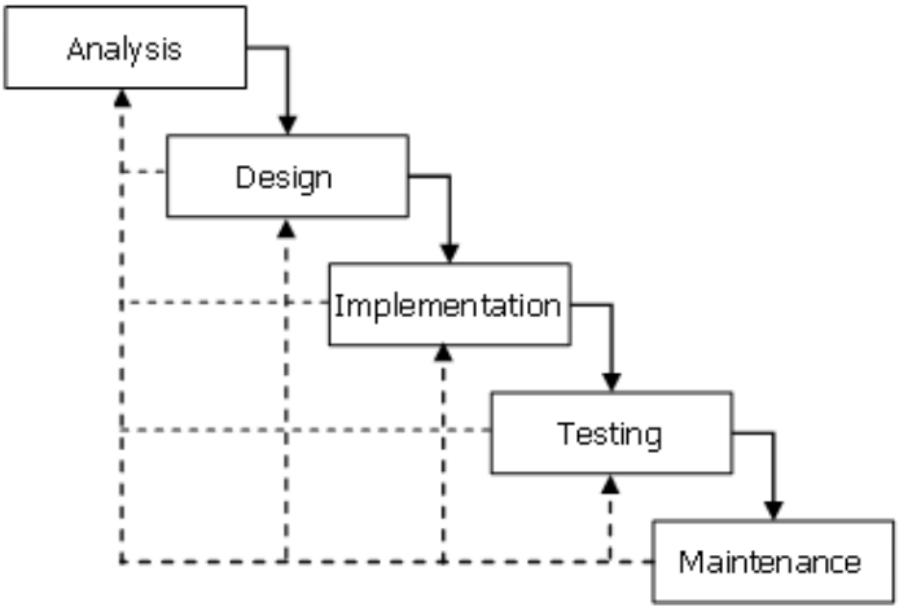
Pada tahap ini dilakukan pengimplementasian rancangan atau desain dengan menuliskan kode program yang merupakan penerjemahan *design* dalam bahasa yang bisa dikenali oleh komputer. Selain itu dalam tahap ini juga dilakukan pemeriksaan terhadap modul yang dibuat, apakah sudah memenuhi fungsi yang diinginkan atau belum. Penulis menggunakan *Delphi* 10.3 CE sebagai editor dan bahasa pemrograman *PASCAL* untuk pembuatan program.

4) Intergration and System Testing

Di tahap ini dilakukan penggabungan modul-modul yang sudah dibuat dan dilakukan pengujian ini dilakukan untuk mengetahui apakah *software* yang dibuat telah sesuai dengan desainnya dan masih terdapat kesalahan atau tidak. Penulis menggunakan *Blackbox Testing* untuk digunakan sebagai metode pengujian pada sistem ini. *Blackbox Testing* adalah pengujian yang dilakukan hanya mengamati hasil eksekusi melalui data uji dan memeriksa fungsional dari perangkat lunak.

5) Operation and Maintenance

Pemeliharan *software* merupakan tahap yang bertujuan untuk jangka panjang. Pada tahap ini juga terdapat tahap pengembangan sistem informasi untuk memastikan sistem tersebut sesuai dan berguna seperti yang diharapkan sebelumnya.

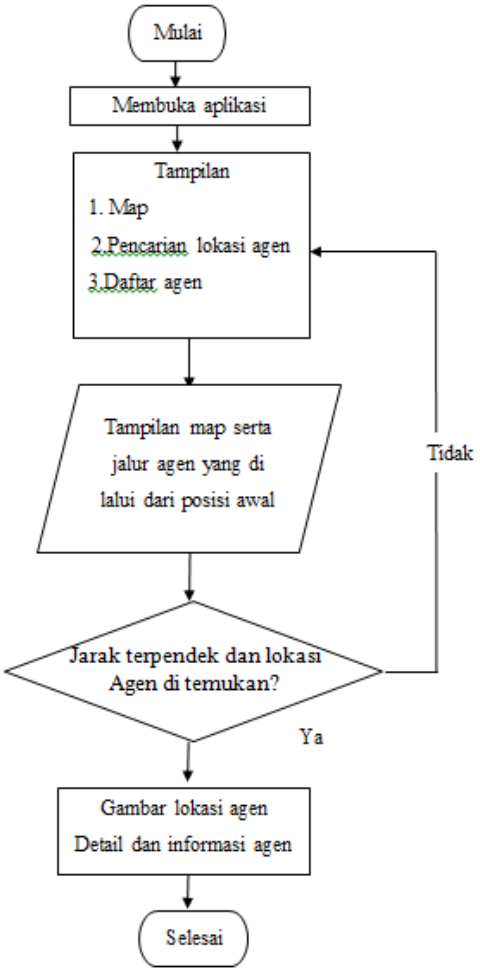


Gambar. 1. Model Waterfall [8]

III. Hasil dan Pembahasan

A. Flowchart Sistem

Alur dari system pencarian jalur terpendek menggunakan algoritma *Floyd Warshall*. Gambar 2 menunjukkan flowchart system yang digunakan pada penelitian ini.



Gambar. 2. Flowchart Sistem

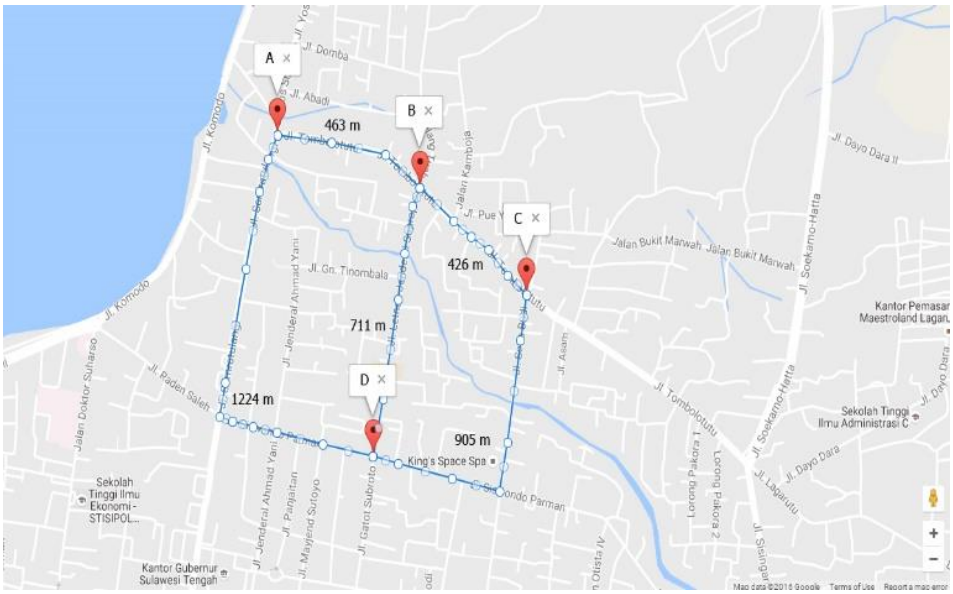
B. Pengujian algoritma Floyd Warshall pada sistem

Sebagai contoh mencari jalur terpendek agen menggunakan algoritma *Floyd Warshall* akan diuji pada sistem yang dibuat oleh penulis. Lokasi agen dapat dilihat pada Tabel 1. Pencarian jalur terpendek menggunakan algoritma *Floyd Warshall* akan dilakukan perhitungan manual dan akan diuji pada sistm yang telah dibuat oleh penulis apakah penentuan jalur terpendek dan jaraknya menghasilkan hasil yang sama. Algoritma ini bisa diterapkan pada sebuah sistem pencarian jalur terpendek dari suatu daerah ke daerah lainnya [9], [10].

Tabel 1. Lokasi Titik Pengujian

Nama Agen	Alamat	Latitude	Longitude
Rapan Maranu	Jl. Tombolotutu No 8. Palu	-0.8802150	119.8727970

Pencarian jalur terpendek termaksud dalam salah satu persoalan dalam teori graf. Algoritma *Floyd Warshall* dapat menghitung bobot terkecil dari semua jalur yang menghubungkan sebuah pasangan titik , dan melakukannya sekaligus untuk semua pasangan titik [11], [12]. Sebagai contoh kasus pencarian jalur terpendek agen rapan maranu pertama akan menggunakan titik A, B, C, D, titk D adalah lokasi pengguna dan titik A adalah lokasi tujuan. Contoh pencarian jalur terpendek menggunakan algoritma *Floyd Warshall* ditunjuk pada Gambar 3.



Gambar. 3. Peta Pengujian

Tabel 2. Matriks Jalur Peta

	A	B	C	D	
A	0	463	∞	1224	1
B	463	0	426	711	2
C	∞	426	0	905	3
D	1224	711	905	0	4
	1	2	3	4	

Langkah pertama kita jadikan jalur peta di atas menjadi sebuah tabel atau matriks. Matriks jalur tersebut dapat dilihat pada Tabel 2. Berdasarkan matriks tersebut, kotak abjad berwarna biru disamping kiri adalah *titik awal* dan kotak abjad berwarna kuning yang ada di atas adalah *titik tujuan*-nya. Sedangkan kotak angka biru dan kuning berfungsi untuk menentukan sebuah *index proses* ***(R0=1,R1=2,R2=3,*** *dan* ***R3=4)*** dan memudahkan posisi angka-angka yang ada didalam tabel dengan mengkombinasikan-nya dengan kotak abjad yang sama dengan warnanya. Karena dalam grafik diatas terdapat 4 buah titik, yaitu **A, B, C, D** maka akan ada 5 proses yang akan dilewati yaitu **R0, R1, R2, R3,** dan **R4** sebagai hasil akhir. perlu diingat bahwa hasil dari sebuah proses akan digunakan untuk proses berikutnya. Perhitungannya adalah sebagai berikut:

r = Index proses. => R0 = 1, R1 = 2, R2 = 3, R3 = 4, R4 = hasil
S = Titik awal. => a, b, c, dan d (kotak biru)
E = Titik tujuan. => A, B, C, dan D (kotak kuning)

1) r = R0 = 1 iterasi pertama

S = a(r) = 0	c(r) = ∞	E= A(r) = 0	C(r) = ∞
b(r) = 463	d(r) = 1224	B(r) = 463	D(r) = 1224

a) Titik Awal a

- Titik awal a ke titik tujuan A => a(A) = 0
  a(r) + A(r) = 0 + 0 = 0 => 0 sama dengan a(A) <tidak diganti>
- Titik awal a ke titik tujuan B => a(B) = 463
  a(r) + B(r) = 0 + 463 = 463 => 463 sama dengan a(B) <tidak diganti>
- Titik awal a ke titik tujuan C => a(C) = tak hingga
  a(r) + C(r) = 0 + tak hingga = tak hingga => tak hingga sama dengan a(C) <tidak diganti>
- Titik awal a ke titik tujuan D => a(D) = 1224
  a(r) + D(r) = 0 + 1224 = 1224 => 1224 sama dengan a(D) <tidak diganti>

b) Titik Awal b

- Titik awal b ke titik tujuan A => b(A) = 463
  b(r) + A(r) = 463 + 0 = 463 => 463 sama dengan b(A) <tidak diganti>
- Titik awal b ke titik tujuan B => b(B) = 0
  b(r) + B(r) = 463 + 463 = 926 => 926 sama dengan b(B) <tidak diganti>
- Titik awal b ke titik tujuan C => b(C) = 426
  b(r) + C(r) = 463 + tak hingga = tak hingga => tak hingga sama dengan b(C) <tidak diganti>
- Titik awal b ke titik tujuan D => b(D) = 711
  b(r) + D(r) = 463 + 1224 = 1687 => 1687 sama dengan b(D) <tidak diganti>

c) Titik Awal c

- Titik awal c ke titik tujuan A => c(A) = tak hingga
  c(r) + A(r) = tak hingga + 0 = tak hingga => tak hingga sama dengan c(A) <tidak diganti>
- Titik awal c ke titik tujuan B => c(B) = 426
  c(r) + B(r) = tak hingga + 463 = tak hingga => tak hingga sama dengan c(B) <tidak diganti>
- Titik awal c ke titik tujuan C => c(C) = 0
  c(r) + C(r) = tak hingga + tak hingga = tak hingga => tak hingga sama dengan c(C) <tidak diganti>
- Titik awal c ke titik tujuan D => c(D) = 905
  c(r) + D(r) = tak hingga + 1224 = tak hingga => tak hingga sama dengan c(D) <tidak diganti>

d) Titik Awal d

- Titik awal d ke titik tujuan A => d(A) = 1224
  d(r) + A(r) = 1224 + 0 = 1224 => 1224 sama dengan d(A) <tidak diganti>
- Titik awal d ke titik tujuan B => d(B) = 711
  d(r) + B(r) = 1224 + 463 = 1687 => 1687 sama dengan d(B) <tidak diganti>
- Titik awal d ke titik tujuan C => d(C) = 905
  d(r) + C(r) = 1224 + tak hingga = tak hingga => tak hingga sama dengan d(C) <tidak diganti>
- Titik awal d ke titik tujuan D => d(D) = 0
  d(r) + D(r) = 1224 + 1224 = 2448 => 2448 sama dengan d(D) <tidak diganti>

2) r = R1 = 2 iterasi kedua

S = a(r) = 463	c(r) = 426	E= A(r) = 463	C(r) = 426
b(r) = 0	d(r) = 711	B(r) = 0	D(r) = 711

a) Titik Awal a

- Titik awal a ke titik tujuan A => a(A) = 0

  a(r) + A(r) = 463 + 463 = 926 => 926 sama dengan a(A) <tidak diganti>
- Titik awal a ke titik tujuan B => a(B) = 463

  a(r) + B(r) = 463 + 0 = 463 => 463 sama dengan a(B) <tidak diganti>
- Titik awal a ke titik tujuan C => a(C) = tak hingga

  a(r) + C(r) = 463 + 426 = 889 => 889 sama dengan a(C) <diganti>
- Titik awal a ke titik tujuan D => a(D) = 1224

  a(r) + D(r) = 463 + 711 = 1174 => 1174 sama dengan a(D) <diganti>

b) Titik Awal b

- Titik awal b ke titik tujuan A => b(A) = 463

  b(r) + A(r) = 0 + 463 = 463 => 463 sama dengan b(A) <tidak diganti>
- Titik awal b ke titik tujuan B => b(B) = 0

  b(r) + B(r) = 0 + 0 = 0 => 0 sama dengan b(B) <tidak diganti>
- Titik awal b ke titik tujuan C => b(C) = 426

  b(r) + C(r) = 0 + 426 = 426 => 426 sama dengan b(C) <tidak diganti>
- Titik awal b ke titik tujuan D => b(D) = 711

  b(r) + D(r) = 0 + 711 = 711 => 711 sama dengan b(D) <tidak diganti>

c) Titik Awal c

- Titik awal c ke titik tujuan A => c(A) = tak hingga

  c(r) + A(r) = 426 + 463 = 889 => 889 sama dengan c(A) <diganti>
- Titik awal c ke titik tujuan B => c(B) = 426

  c(r) + B(r) = 426 + 0 = 426 => 426 sama dengan c(B) <tidak diganti>
- Titik awal c ke titik tujuan C => c(C) = 0

  c(r) + C(r) = 426 + 426 = 852=> 852 sama dengan c(C) <tidak diganti>
- Titik awal c ke titik tujuan D => c(D) = 905

  c(r) + D(r) = 426 + 711 = 1137 => 1137 sama dengan c(D) <tidak diganti>

d) Titik Awal d

- Titik awal d ke titik tujuan A => d(A) = 1224

  d(r) + A(r) = 711 + 463 = 1174 => 1174 sama dengan d(A) <diganti>
- Titik awal d ke titik tujuan B => d(B) = 711

  d(r) + B(r) = 711 + 0 = 711 => 711 sama dengan d(B) <tidak diganti>
- Titik awal d ke titik tujuan C => d(C) = 905

  d(r) + C(r) = 711 + 426 = 1137 => 1137 sama dengan d(C) <tidak diganti>
- Titik awal d ke titik tujuan D => d(D) = 0

  d(r) + D(r) = 711 + 711 = 1422=> 1422 sama dengan d(D) <tidak diganti>

3) r = R2 = 3 iterasi ketiga

S = a(r) = 889	c(r) = 0	E = A(r) = 889	C(r) = 0
b(r) = 426	d(r) = 905	B(r) = 426	D(r) = 905

a) Titik Awal a

- Titik awal a ke titik tujuan A => a(A) = 0

  a(r) + A(r) = 889 + 889 = 1778 => 1778 sama dengan a(A) <tidak diganti>
- Titik awal a ke titik tujuan B => a(B) = 463

  a(r) + B(r) = 889 + 426 = 1315 => 1315 sama dengan a(B) <tidak diganti>
- Titik awal a ke titik tujuan C => a(C) = 889

  a(r) + C(r) = 889 + 0 = 889 => 889 sama dengan a(C) <tidak diganti>
- Titik awal a ke titik tujuan D => a(D) = 1174

  a(r) + D(r) = 889 + 905 = 1794 => 1794 sama dengan a(D) <tidak diganti>

b) Titik Awal b

- Titik awal b ke titik tujuan A => b(A) = 463

  b(r) + A(r) = 426 + 889 = 1315 => 1315 sama dengan b(A) <tidak diganti>
- Titik awal b ke titik tujuan B => b(B) = 0

  b(r) + B(r) = 426 + 426 = 926 => 852 sama dengan b(B) <tidak diganti>
- Titik awal b ke titik tujuan C => b(C) = 426

  b(r) + C(r) = 426 + 0 = 426 => 426 sama dengan b(C) <tidak diganti>
- Titik awal b ke titik tujuan D => b(D) = 711

  b(r) + D(r) = 426 + 905 = 1331 => 1331 sama dengan b(D) <tidak diganti>

c) Titik Awal c

- Titik awal c ke titik tujuan A => c(A) = 889

  c(r) + A(r) = 0 + 889 = 889 => 889 sama dengan c(A) <tidak diganti>
- Titik awal c ke titik tujuan B => c(B) = 426

  c(r) + B(r) = 0 + 426 = 426 => 426 sama dengan c(B) <tidak diganti>
- Titik awal c ke titik tujuan C => c(C) = 0

  c(r) + C(r) = 0 + 0 = 0 => 0 sama dengan c(C) <tidak diganti>
- Titik awal c ke titik tujuan D => c(D) = 905

  c(r) + D(r) = 0 + 905 = 905 => 905 sama dengan c(D) <tidak diganti>

d) Titik Awal d

- Titik awal d ke titik tujuan A => d(A) = 1174

  d(r) + A(r) = 905 + 889 = 1794 => 1794 sama dengan d(A) <tidak diganti>
- Titik awal d ke titik tujuan B => d(B) = 711

  d(r) + B(r) = 905 + 426 = 1331 => 1331 sama dengan d(B) <tidak diganti>
- Titik awal d ke titik tujuan C => d(C) = 905

  d(r) + C(r) = 905 + 0 = 905 => 905 sama dengan d(C) <tidak diganti>
- Titik awal d ke titik tujuan D => d(D) = 0

  d(r) + D(r) = 905 + 905 = 1810 => 1810 sama dengan d(D) <tidak diganti>

4) r = R3 = 4 iterasi keempat

S = a(r) = 1174	c(r) = 905	E= A(r) = 1174	C(r) = 905
b(r) = 711	d(r) = 0	B(r) = 711	D(r) = 0

a) Titik Awal a

- Titik awal a ke titik tujuan A => a(A) = 0

  a(r) + A(r) = 1174 + 1174 = 2348 => 2348 sama dengan a(A) <tidak diganti>
- Titik awal a ke titik tujuan B => a(B) = 463

  a(r) + B(r) = 1174 +711 = 1885 => 1885 sama dengan a(B) <tidak diganti>
- Titik awal a ke titik tujuan C => a(C) = 889

  a(r) + C(r) = 1174 + 905 = 2079 => 2079 sama dengan a(C) <tidak diganti>
- Titik awal a ke titik tujuan D => a(D) = 1174

  a(r) + D(r) = 1174 + 0 = 1174 => 1174 sama dengan a(D) <tidak diganti>

b) Titik Awal b

- Titik awal b ke titik tujuan A => b(A) = 463

  b(r) + A(r) = 711 + 1174 = 1885 => 1885 sama dengan b(A) <tidak diganti>
- Titik awal b ke titik tujuan B => b(B) = 0

  b(r) + B(r) = 711 + 711 = 1422 => 1422 sama dengan b(B) <tidak diganti>
- Titik awal b ke titik tujuan C => b(C) = 426

  b(r) + C(r) = 711 + 905 = 1616 => 1616 sama dengan b(C) <tidak diganti>
- Titik awal b ke titik tujuan D => b(D) = 711

  b(r) + D(r) = 711 + 0 = 711 => 711 sama dengan b(D) <tidak diganti>

c) Titik Awal c

- Titik awal c ke titik tujuan A => c(A) = 889

  c(r) + A(r) = 905 + 1174 = 2079 => 2079 sama dengan c(A) <tidak diganti>
- Titik awal c ke titik tujuan B => c(B) = 426

  c(r) + B(r) = 905 + 711 = 1616 => 1616 sama dengan c(B) <tidak diganti>
- Titik awal c ke titik tujuan C => c(C) = 0

  c(r) + C(r) = 905 + 905 = 1810 => 1810 sama dengan c(C) <tidak diganti>
- Titik awal c ke titik tujuan D => c(D) = 905

  c(r) + D(r) = 905 + 0 = 905 => 905 sama dengan c(D) <tidak diganti>

d) Titik Awal d

- Titik awal d ke titik tujuan A => d(A) = 1174

  d(r) + A(r) = 0 + 1174 = 1174 => 1174 sama dengan d(A) <tidak diganti>
- Titik awal d ke titik tujuan B => d(B) = 711

  d(r) + B(r) = 0 + 711 = 711 => 711 sama dengan d(B) <tidak diganti>
- Titik awal d ke titik tujuan C => d(C) = 905

  d(r) + C(r) = 0 + 905 = 905 => 905 sama dengan d(C) <tidak diganti>
- Titik awal d ke titik tujuan D => d(D) = 0

  d(r) + D(r) = 0 + 0 = 0 => 0 sama dengan d(D) <tidak diganti>

Demikianlah proses dari **R0** sampai **R3**. Terdapat beberapa jarak yang diganti karena hasil penjumlahan yang menunjukan lebih dari nilai jarak yang sebenarnya S(E). Tabel matriks hasil akhir papat dilihat pada Tabel 3.

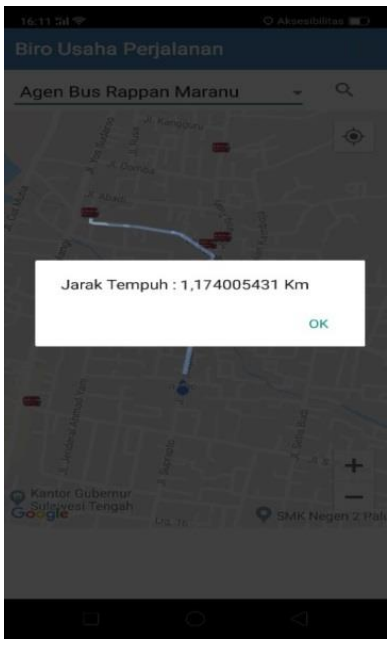
Tabel 3. Matriks hasil akhir

	A	B	C	D	
a	0	463	889	1174	1
b	463	0	426	711	2
c	889	426	0	905	3
d	1174	711	905	0	4
	1	2	3	4	

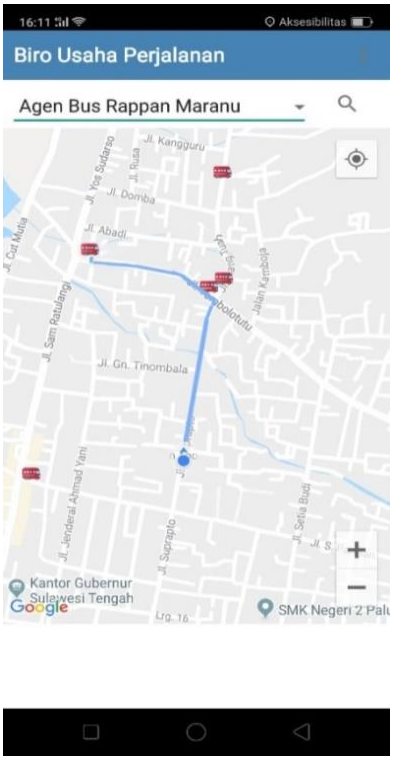
Dari hasil Tabel 3 dapat diketahui jalur terpendek manakah yang harus ditempuh dari titik **D** menuju ke titik **A**. Kemudian kita tulis nilai-nilai yang ada di Tabel 3 dalam grafik pada peta sesuai abjad pada matriks. Ditunjuk pada Gambar 4.

Gambar. 4. Hasil Perhitungan

Dengan begitu diketahui jalur yang harus dilewati dan memiliki jarak terpendek dari titik **D** menuju titik **A** adalah **D -> B -> A** dengan jarak **1,1 km**. Berikut adalah *screenshot* pencarian jalur terpendek dengan menggunakan algoritma *Floyd Warshall* yang telah diterapkan pada sistem berbasis *android* ditunjuk pada Gambar 5. dan Gambar 6.



Gambar. 5. *Screenshot* sistem pertama



Gambar. 6. *Screenshot* sistem kedua

IV. Kesimpulan

Berdasarkan pengujian dan analisis sistem implementasi algoritma *Floyd Warshall* pencarian jalur terpendek lokasi agen bus, *tour and travel*, dan rental mobil berbasis *Android* di kota Palu maka dapat diambil beberapa kesimpulan. Dengan menggunakan aplikasi Delphi 10.3 CE penulis berhasil mengimplementasikan algoritma *Floyd Warshall* untuk pencarian jalur terpendek lokasi agen bus, *tour and travel* dan rental mobil di kota Palu berbasis *Android*. Kemudian, Algoritma *Floyd Warshall* adalah algoritma pencarian jalur terpendek yang bersifat dinamis. Algoritma *Floyd Warshall* dapat menghitung bobot terkecil dari semua jalur yang menghubungkan sebuah pasangan titik , dan melakukannya sekaligus untuk semua pasangan titik.

Perhitungan pencarian jalur terpendek menggunakan algoritma *Floyd Warshall* langkah pertama menentukan jalur yang akan dilalui dan nilai bobotnya dibuat ke dalam tabel atau matriks, lakukan perhitungan periterasi sampai dengan iterasi terakhir, Jika hasil penjumlahan $\underset{ik}{d}$ dan $\underset{kj}{d}$ lebih kecil dari $\underset{ij}{d}, maka$ ganti nilai $\underset{ik}{d}$ menjadi $\underset{ik}{d} = \underset{ik}{d} + \underset{kj}{d}$, lakukan berulang sebanyak vertexnya, dan menentukan jalur terpendek yaitu dari hasil iterasi terakhir. Proses perhitungan algoritma *Floyd Warshall* tersebut di implementasikan ke dalam sistem yang telah dibuat menggunakan aplikasi Delphi 10.3 CE dengan memanfaatkan prosedur *pseudocode* algoritma Floyd Warshall. Penelitian ini menggunakan data sebanyak 120 lokasi agen bus, *tour and travel* dan rental mobil di kota Palu dengan mengimplementasikan algoritma *Floyd Warshall* untuk mencari jalur terpendek yang diterapkan pada Delphi 10.3 CE. Pada Delphi 10.3 CE peneliti merasakan kemudahan dan lebih ringan dalam pengerjaan algoritma *Floyd Warshall* untuk pencarian jalur terpendek.

Daftar Pustaka

[1] Kent D. Lee and S. Hubbard, Data Structures and Algorithms with Python. Springer, 2015.

[2] K. Mehlhorn, Data structures and algorithms 2: graph algorithms and NP-completeness, Vol. 2. Springer Science & Business Media, 2012.

[3] T. M. Chan, "More algorithms for all-pairs shortest paths in weighted graphs," SIAM J. Comput., vol. 39, no. 5, pp. 2075–2089, 2010.

[4] A. T. Fitria, "Implementasi Algoritma Dijkstra dalam Aplikasi untuk Menentukan Lintasan Terpendek Jalan Darat antar Kota di Sumatera Bagian Selatan," J. Sist. Inf., vol. 5, no. 2, pp. 611–621, 2013.

[5] S. Hougardy, "The Floyd–Warshall algorithm on graphs with negative cycles," Inf. Process. Lett., vol. 110, no. 8–9, pp. 279–281, 2010.

[6] A. Aini and A. Salehipour, "Speeding up the Floyd–Warshall algorithm for the cycled shortest path problem," Appl. Math. Lett., vol. 25, no. 1, pp. 1–5, 2012.

[7] A. Pradhan and G. Mahinthakumar, "Finding all-pairs shortest path for a large-scale transportation network using parallel Floyd-Warshall and parallel Dijkstra algorithms," J. Comput. Civ. Eng., vol. 27, no. 3, pp. 263–273, 2013.

[8] Y. Bassil, "A Simulation Model for the Waterfall Software Development Life Cycle," Int. J. Eng. Technol., vol. 2, no. 5, 2012.

[9] N. K. D. A. Jayanti, "Penggunaan Algoritma Floyd Warshall Dalam Masalah Jalur Terpendek Pada Penentuan Tata Letak Parkir," in Seminar Nasional Informatika, 2017, pp. 75–81.

[10] A. R. Hasibuan, "Penerapan Algoritma Floyd Warshall untuk Menentukan Jalur Terpendek dalam Pengiriman Barang," J. Ris. Komput., vol. 3, no. 6, 2016.

[11] I. Iftadi, W. A. Jauhari, and B. Nugroho, "Perancangan Peta Evakuasi Menggunakan Algoritma Floyd-Warshall untuk Penentuan Lintasan Terpendek: Studi Kasus," PERFORMA Media Ilm. Tek. Ind., vol. 10, no. 2, 2011.

[12] A. Andriani, "Rancang Bangun Sistem Informasi Rute Wisata Terpendek Berbasis Algoritma Floyd-Warshall," Bianglala Inform., vol. 2, no. 1, 2014.