BAB II

LANDASAN TEORI

2.1. Aplikasi *Mobile*

2.1.1. Definisi Aplikasi

Aplikasi adalah satu *unit* perangkat lunak yang dibuat untuk melayani kebutuhan akan beberapa aktivitas, seperti perangkat lunak untuk perusahaan, desain grafis dan untuk bermain *game* (Buyens, 2001).

2.1.2. Definisi *Mobile*

Mobile application juga biasa disebut dengan *mobile apps*, yaitu istilah yang digunakan untuk medeskripsikan aplikasi internet yang berjalan pada *smartphone* atau piranti *mobile* lainnya. Aplikasi *mobile* biasanya membantu para penggunanya untuk terkoneksi dengan layanan internet yang biasa diakses pada PC atau mempermudah mereka untuk menggunakan aplikasi internet pada piranti yang bisa dibawa (Turban, 2012).

2.1.3. Definisi Aplikasi Mobile

Aplikasi mobile berasal dari kata *application* dan *mobile*. *Application* yang artinya penerapan, lamaran, penggunaan. Secara istilah aplikasi adalah program siap pakai yang direka untuk melaksanakan suatu

fungsi bagi pengguna atau aplikasi yang lain dan dapat digunakan oleh sasaran yang dituju sedangkan mobile dapat di artikan sebagai perpindahan dari suatu tempat ketempat yang lain (Buyens, 2001).

Aplikasi *mobile* dapat diakses menggunakan perangkat nirkabel seperti telepon selular dan PDA. Aplikasi *mobile* juga dapat mempermudah dalam melakukan aktifitas sehari-hari seperti *chatting, browsing, shopping* dan lain sebagainya.

2.2. Android

Android merupakan sistem operasi yang dirancang untuk *mobile device* berbasis Linux seperti *smartphone* dan tablet. Android memberikan kesempatan bagi para pengembang untuk menciptakan aplikasi mereka sendiri yang dapat digunakan pada bermacam-macam *mobile device* (Supardi, 2015).

Android didirikan oleh Andy Rubin, Rich Miner, Nick Sears, dan Chris White pada tahun 2003. Kemudian pada tahun 2005, Andy Rubin dan penemu Google, Larry Page melakukan pertemuan di kantor Google. Pertemuan tersebut bukan pertemuan pertama, tiga tahun sebelumnya mereka sudah pernah berjumpa, pada saat Andy Rubin akan merilis *smartphone* yang akan dibuatnya. *Smartphone* tersebut diberi nama "*Sidekick*" yang memakai mesin pencari *default* dari Google. Kemudian pada bulan Juli 2005, Google membeli Android dengan estimasi harga sekitar USD 50 juta pada saat itu.

Dalam pengembangan Android, dibentuklah OHA *(Open Handset Aliance*) yang merupakan aliansi dari 34 perusahaan perangkar keras (*hardware*), perangkat lunak (*software*), dan perusahaan telekomunikasi, termasuk Google, HTC, Motorola, Intel, Nvdia, dan Qualcomm. Kemudian pada tanggal 5 November 2007, Android telah dirilis untuk pertama kalinya. Android bersama OHA menyatakan dukungannya terhadap pengembangan *open source* pada perangkat *mobile (mobile device)*. Hingga pada saat ini sudah banyak *vendor* yang menggunakan sistem operasi Android seperti, Samsung, LG, HTC, Asus dan masih banyak lagi.

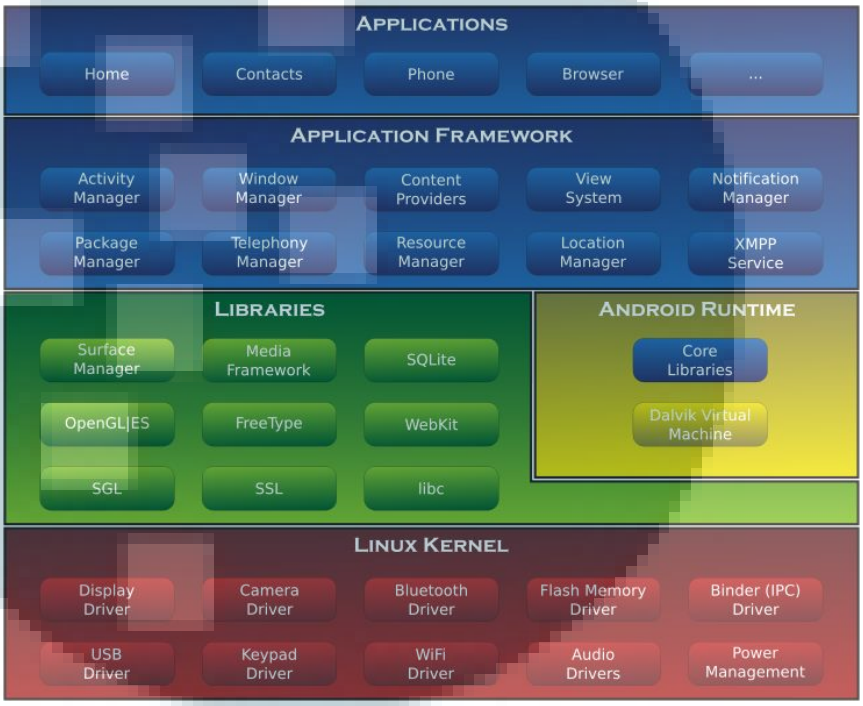
2.2.1. Arsitektur Android

Arsitektur Android terdiri dari *Applications* dan *Widgets*, *Applications Frameworks*, *Libraries*, Android *Run Time*, dan Linux Kernel. Berikut adalah penjelasan dari masing-masing arsitekturnya (Supardi, 2015):

a) *Applications* dan *Widgets* merupakan *layer* atau lapisan yang berhubungan hanya dengan aplikasinya saja.
b) *Application Frameworks* merupakan *Open Development Platform* yang ditawarkan oleh Android kepada para pengembang dalam merancang dan membuat aplikasi.
c) *Libraries* merupakan *layer* atau lapisan yang berisi fitur-fitur Android.

d) Android *Run Time* merupakan *layer* atau lapisan yang berfungsi untuk menjalankan aplikasi Android yang dalam prosesnya menggunakan implementasi Linux.

e) Linux Kernel merupakan *layer* pusat dari OS Android berada.



Gambar 2.1 Arsitektur Android
Sumber : www.wikipedia.org

2.2.2. Versi Android

Berikut adalah tabel versi Android beserta tanggal rilis dari masing-masing versi (www.wikipedia.org, 12 Mei 2016):

Tabel 2.1 Versi Android
Sumber : www.wikipedia.org

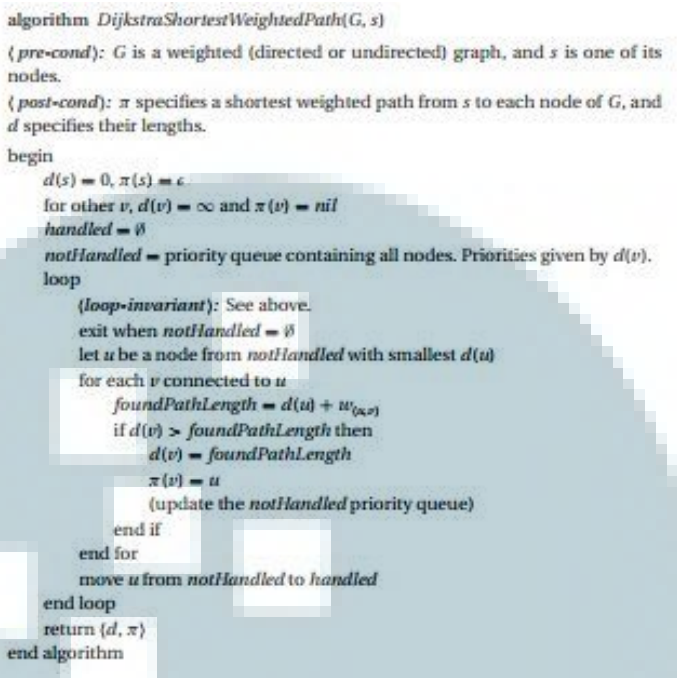
Code Name	Version Number	Initial Release Date
Cupcake	1.5	27 April 2009
Donut	1.6	15 September 2009
Éclair	2.0 – 2.1	26 Oktober 2009
Froyo	2.2 – 2.2.3	20 Mei 2010
Gingerbread	2.3 – 2.3.7	6 Desember 2010
Honeycomb	3.0 – 3.2.6	22 Februari 2011
Ice Cream Sandwich	4.0.1 – 4.0.4	18 Oktober 2011
Jelly Bean	4.1	9 Juli 2012
Kitkat	4.4	31 Oktober 2013
Lollipop	5.0 – 5.1.1	12 November 2014
Marshmallow	6.0 – 6.0.1	5 Oktober 2015

2.3. Algoritma Dijkstra

Algoritma Dijkstra adalah sebuah algoritma rakus (*greedy algorithm*) yang dipakai dalam memecahkan permasalahan jarak terpendek (*shortest path problem*) untuk sebuah graf berarah (*directed graph*) dengan bobot-bobot sisi (*edge weights*) yang bernilai tak-negatif (Thomas, 2001).

Algoritma Dijkstra menggunakan prinsip antrian berprioritas. Jadi rute yang akan ditelusuri oleh algoritma Dijkstra adalah rute yang memiliki prioritas tertinggi saja. Dalam penentuan rute atau simpul yang berprioritas ini, algoritma Dijsktra membandingkan setiap nilai atau bobot simpul yang berada dalam satu *level*. Kemudian nilai atau bobot dari simpul tersebut disimpan untuk dibandingkan dengan nilai dari rute yang baru akan ditemukan.

Berikut adalah cara penggunaan algoritma Dijkstra berdasarkan *pseudocode* algoritma Dijkstra dalam pencarian rute terpendek:



```
algorithm DijkstraShortestWeightedPath(G, s)
⟨pre-cond⟩: G is a weighted (directed or undirected) graph, and s is one of its nodes.
⟨post-cond⟩: π specifies a shortest weighted path from s to each node of G, and d specifies their lengths.
begin
    d(s) ← 0, π(s) ← ε
    for other v, d(v) ← ∞ and π(v) ← nil
    handled ← ∅
    notHandled ← priority queue containing all nodes. Priorities given by d(v).
    loop
        ⟨loop-invariant⟩: See above.
        exit when notHandled ← ∅
        let u be a node from notHandled with smallest d(u)
        for each v connected to u
            foundPathLength ← d(u) + w(u,v)
            if d(v) > foundPathLength then
                d(v) ← foundPathLength
                π(v) ← u
                (update the notHandled priority queue)
            end if
        end for
        move u from notHandled to handled
    end loop
    return ⟨d, π⟩
end algorithm
```

Gambar 2.2 *Pseudocode* Algoritma Dijkstra
Sumber : www.wikipedia.org

Berdasarkan *pseudocode* tersebut, terdapat 3 hal yang menggambarkan status dari setiap simpul yang ditelusuri. Berikut adalah penjelasannya:

a) *Node* atau simpul yang sudah ditemukan dan belum ditangani.

b) *Node* atau simpul yang belum ditemukan tapi belum ditangani.

c) *Node* atau simpul yang sudah ditemukan dan sudah ditangani.

Simpul yang dikunjungi merupakan simpul terpendek dari setiap *level* atau tahapan dari algoritma Dijkstra. Jadi jalur atau rute yang dibentuk oleh algoritma Dijkstra tersusun dari simpul yang telah ditelusuri berdasarkan tahapan masing-masing.

Berikut adalah langkah-langkah dari penggunaan algoritma Dijkstra berdasarkan *pseudocode*:

a) Menetapkan node atau simpul awal dengan status sudah ditemukan (*found*) dan ditangani (*handled*).
b) Melakukan penelusuran pada setiap *node* atau simpul yang kemungkinan bisa dijangkau secara langsung dari *node* atau simpul yang sedang dikunjungi. Jika titik yang didapatkan belum pernah ditemukan atau ditangani sebelumnya, maka statusnya dirubah menjadi sudah ditemukan (*founded*). Jika titik yang didapatkan sudah pernah ditemukan atau ditangani sebelumnya, maka nilai dari titik tersebut di-*update* berdasarkan nilai atau bobot yang terkecil.
c) Melakukan penelusuran terhadap titik yang memiliki nilai terkecil dari setiap simpul yang berstatus *found* dan kemudian mengunjungi simpul tersebut.
d) Lakukan pengulangan secara berurutan berdasarkan langkah-langkah diatas hingga semua simpul ditemukan.

Berikut adalah langkah-langkah algoritma Dijsktra dalam mencari rute terpendek dari sebuah jalur dalam sebuah graf:

Keterangan graf :

= Simpul atau *node* yang sudah ditemukan dan belum ditangani

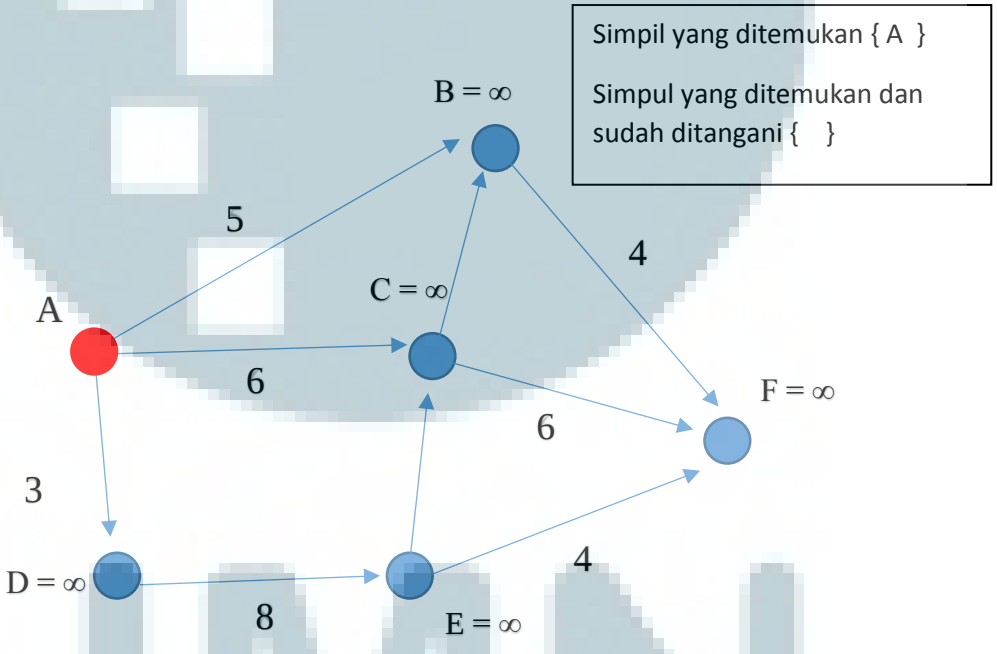
= Simpul atau *node* yang belum ditemukan dan belum ditangani

= Simpul atau *node* yang sudah ditemukan dan sudah ditangani

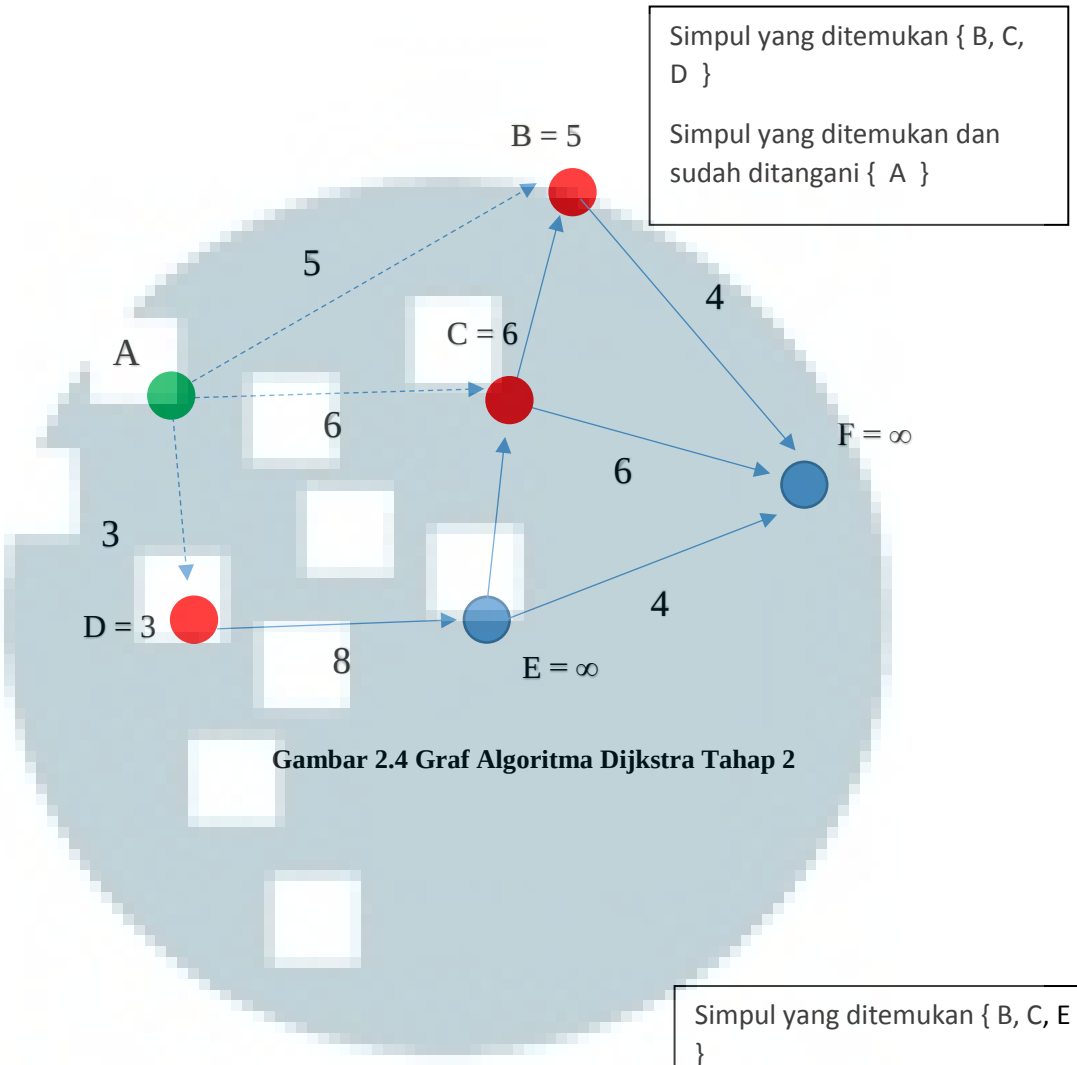
= Jalur yang sudah ditemukan dan belum ditangani

= Jalur yang belum ditemukan dan belum ditangani

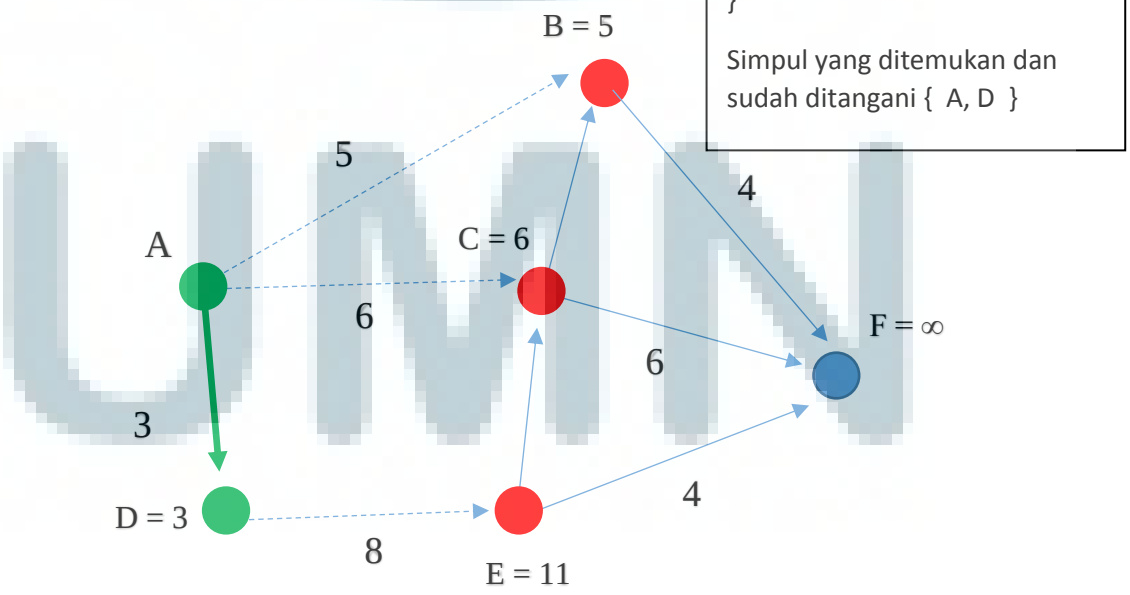
= Jalur yang sudah ditemukan dan sudah ditangani



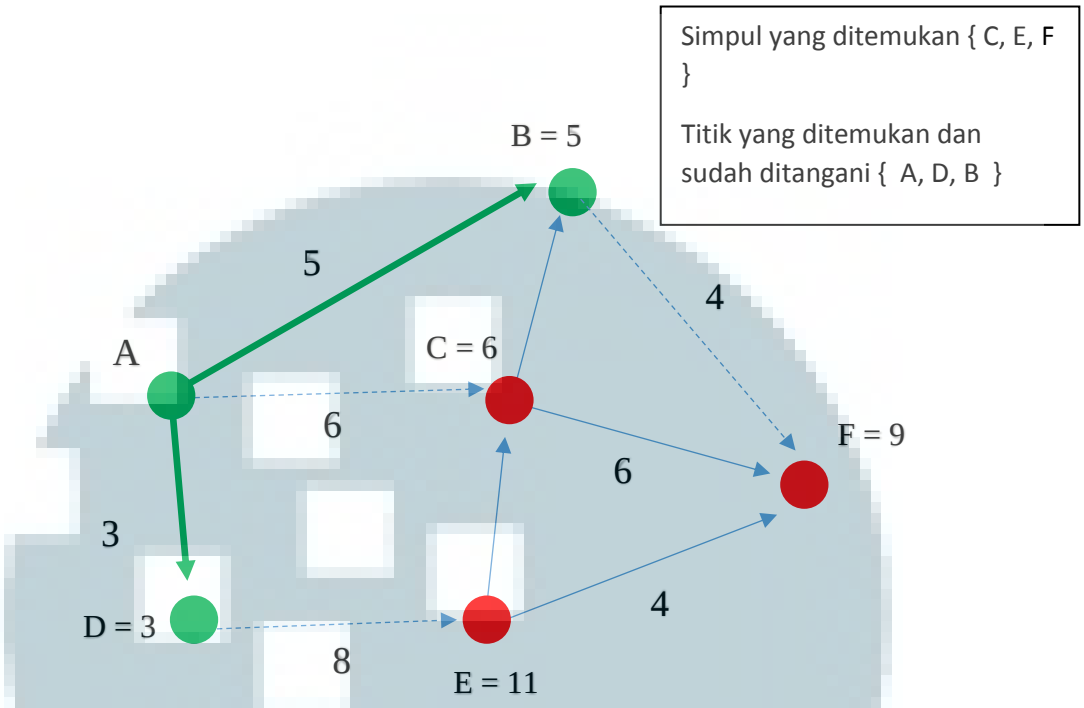
Gambar 2.3 Graf Algoritma Dijkstra Tahap 1



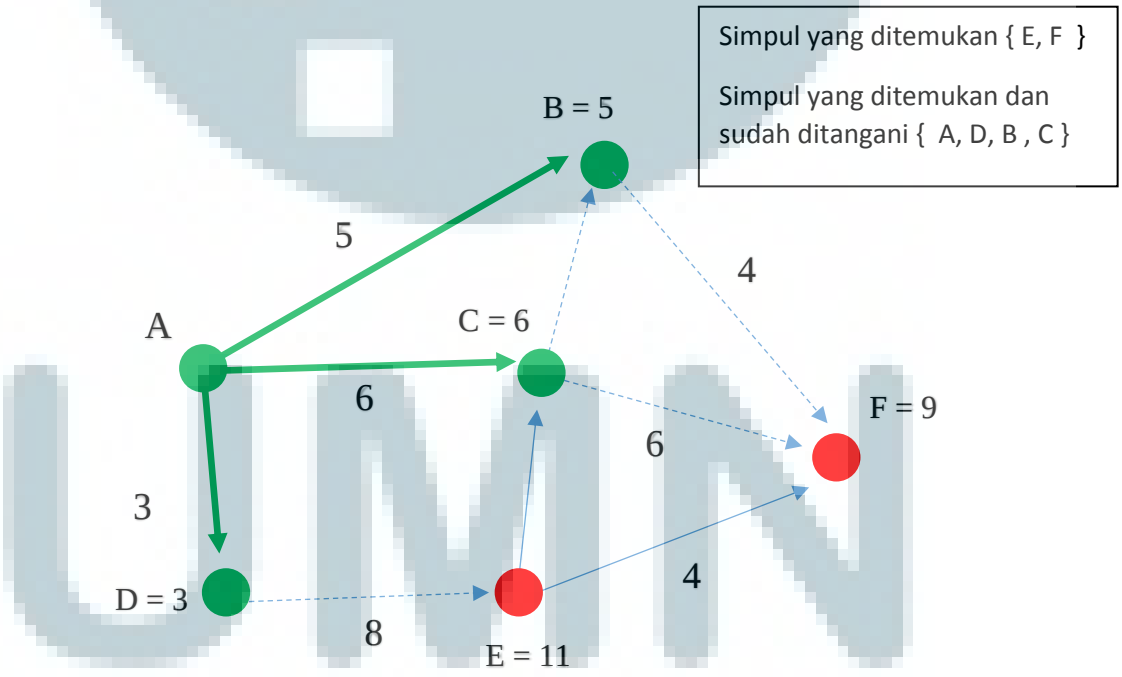
Gambar 2.4 Graf Algoritma Dijkstra Tahap 2



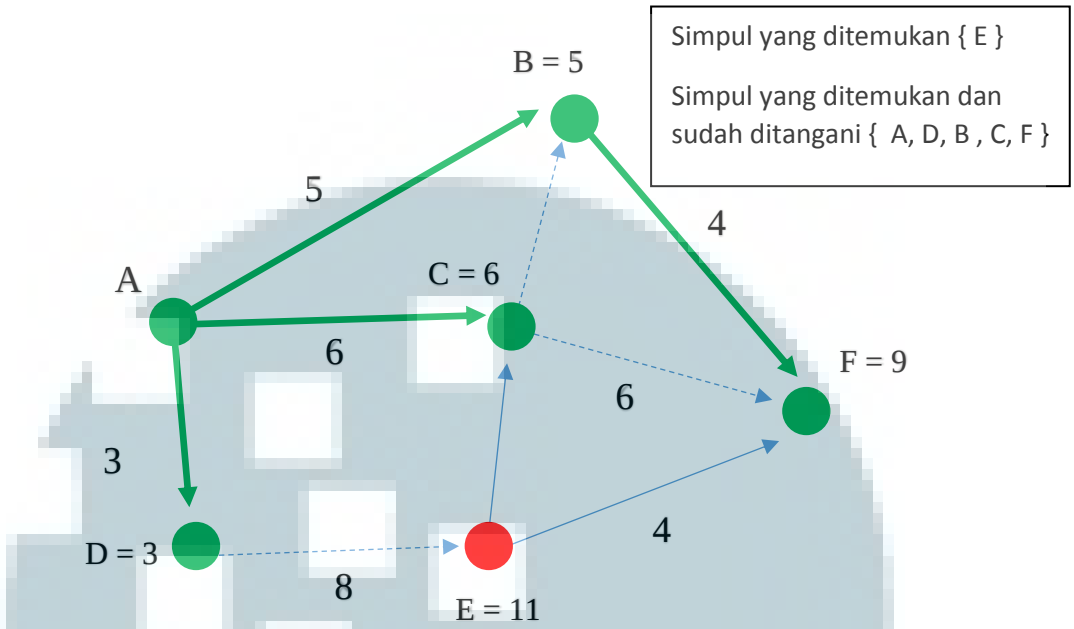
Gambar 2.5 Graf Algoritma Dijkstra Tahap 3



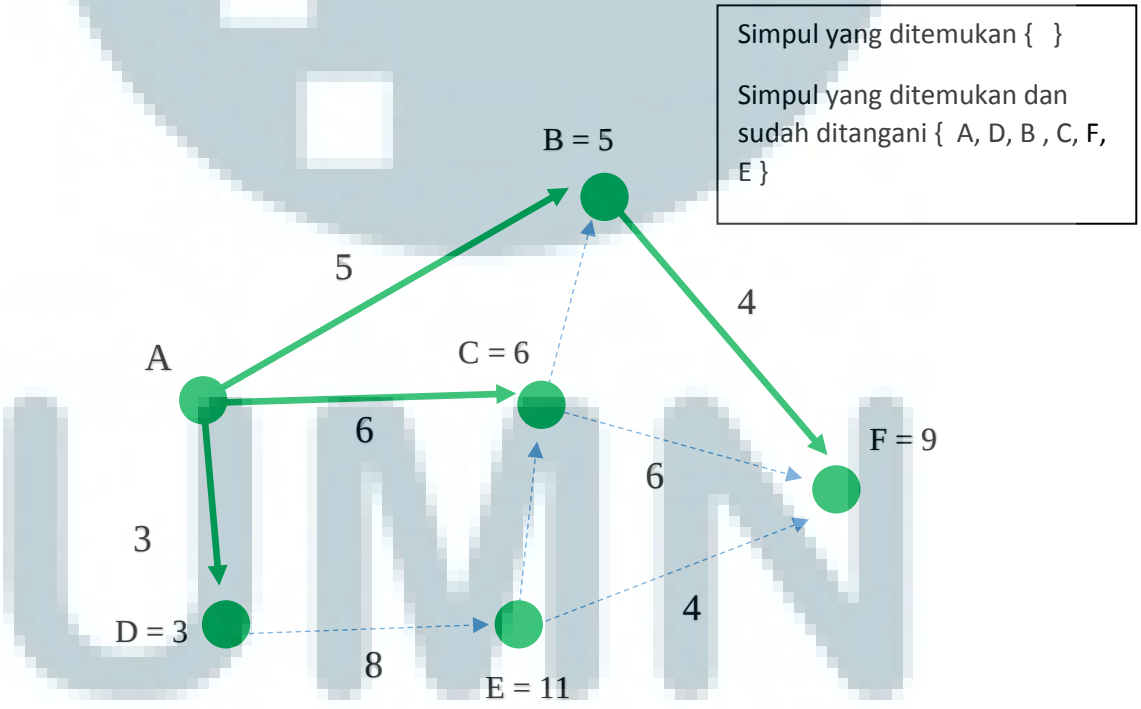
Gambar 2.6 Graf Algoritma Dijkstra Tahap 4



Gambar 2.7 Graf Algoritma Dijkstra Tahap 5



Gambar 2.8 Graf Algoritma Dijkstra Tahap 6



Gambar 2.9 Graf Algoritma Dijkstra Tahap 7

Berdasarkan dari urutan gambar 2.3 hingga gambar 2.9 dapat dijelaskan bahwa untuk simpul A yang merupakan simpul awal agar bisa sampai ke simpul tujuan yaitu simpul F dapat melalui jalur terpendek yang telah ditemukan dan telah ditangani berdasarkan nilainya masing-masing. Untuk sampai ke simpul F dengan total jarak 9 dapat melalui rute A, B dan F yang merupakan rute terpendek dengan asumsi terburuk semua simpul telah ditemukan dan ditangani berdasarkan perhitungan pada algoritma Dijkstra.

2.4. *Prototyping*

2.4.1. Pengertian *Prototyping*

Metode *prototyping* sangat baik digunakan untuk menyelesaikan masalah yang timbul akibat kesalahpahaman antara pengguna dan analis sistem yang timbul akibat pengguna tidak mampu mendefinisikan secara jelas kebutuhannya (Mulyanto, 2009).

Prototyping adalah pengembangan yang cepat dan pengujian terhadap model kerja (*prototype*) dari aplikasi baru melalui proses interaksi dan berulang-ulang yang biasa digunakan oleh ahli sistem informasi dan ahli bisnis. *Prototyping* disebut juga desain aplikasi cepat (*rapid application design*) karena menyederhanakan dan mempercepat desain sistemnya (O'Brien, 2005).

Menurut Kendal (2003), jenis-jenis informasi yang dicari saat melakukan *prototyping* adalah sebagai berikut :

a) Reaksi awal dari pengguna: Pada saat *system analyst* menampilkan sebuah prototipe sistem informasi, maka pada saat itu juga pengguna akan melakukan reaksi terhadap prototipe.
b) Saran-saran dari pengguna: Analis juga tertarik dengan saran-saran pengguna dan pihak manajemen perbaikan terhadap prototipe yang ditampilkan
c) Inovasi: Inovasi merupakan bagian dari informasi yang dicari oleh analis sistem. Inovasi adalah kemampuan-kemampuan sistem baru yang tidak dianggap berhubungan dengan waktu saat pengguna mulai berinteraksi dengan prototipe.
d) Rencana revisi: Berfungsi untuk membantu mengidentifikasikan prioritas apa yang akan diprototipekan

2.4.2. Langkah Pengembangan *Prototyping*

Menurut Kendal (2003), langkah-langkah pengembangan prototipe adalah sebagai berikut :

a) Mengidentifikasi kebutuhan pemakai: mewawancarai untuk mendapatkan gagasan dari apa yang diinginkan oleh pemakai.
b) Pengembangan prototipe: bekerjasama dengan spesialis informasi lain, menggunakan satu atau lebih pendekatan *prototyping* untuk mengembangkan sebuah prototipe. Contoh dari alat *prototyping* adalah *Integrated* A*pplication Generator* yang merupakan sistem perangkat

lunak yang mampu menghasilkan sebuah tampilan yang diinginkan dalam suatu sistem baru sperti menu, laporan, layer dan *database*.

c) Menentukan apakah prototipe dapat diterima dengan baik oleh pengguna dan juga memberi kesempatan kepada mereka untuk beradaptasi dengan sistem.
d) Menggunakan prototipe menjadi sistem operasional.
e) Melakukan perancangan sistem operasional.
f) Melakukan pengujian sistem operasional.
g) Menentukan jika sistem operasional dapat diterima.

2.5. *Unified Modeling Language* (UML)

2.5.1. Definisi UML

Berikut adalah definisi dari *Unified Modeling Language* (UML) menurut para ahli:

a) UML adalah bahasa pemodelan standar yang memiliki sintak dan semantic (Widodo, 2011).
b) UML (*Unified Modeling Language*) adalah bahasa pemodelan untuk sistem atau perangkat lunak yang berorientasi objek. Pemodelan (*modeling*) sesungguhnya digunakan untuk penyederhanaan permasalahan-permasalahan yang kompleks sedemikian rupa sehingga lebih mudah dipelajari dan dipahami (Nugroho, 2010).

Berdasarkan pendapat yang dikemukakan di atas dapat disimpulkan bahwa UML adalah sebuah bahasa yang berbentuk grafik atau gambar yang

berfungsi untuk melakukan visualisasi, menspesifikasikan, membangun dan melakukan dokumentasi dari sebuah sistem pengembangan perangkat lunak yang berbasis Objek (*Object Oriented programming*).

2.5.2. Jenis - Jenis UML

Menurut Henderi (2008), Berikut ini adalah definisi mengenai 5 diagram UML, antara lain:

a) *Use Case Diagram* secara grafis menggambarkan interaksi antara sistem, sistem *eksternal* dan pengguna. Dengan kata lain *use case* diagram secara grafis mendeskripsikan siapa yang akan menggunakan sistem dan dalam cara apa pengguna (*user*) mengharapkan interaksi dengan sistem itu. *Use case* secara naratif digunakan untuk secara tekstual menggambarkan sekuensi langkah-langkah dari setiap interaksi.

b) *Class Diagram* menggambarkan struktur obyek sistem. Diagram ini menunjukkan *class object* yang menyusun sistem dan juga hubungan antara *class object* tersebut.

c) *Sequence Diagram* secara grafis menggambarkan bagaimana objek berinteraksi dengan satu sama lain melalui pesan pada sekuensi sebuah *use case* atau operasi.

d) *State Chart Diagram* digunakan untuk memodelkan *behaviour* objek khusus yang dinamis. Diagram ini mengilustrasikan siklus hidup objek berbagai keadaan yang dapat diasumsikan oleh objek dan *event-*

event (kejadian) yang menyebabkan objek beralih dari satu tempat ke tempat yang lain.

e) *Activity Diagram* secara grafis digunakan untuk menggambarkan rangkaian aliran aktivitas baik proses bisnis maupun *use case*. *Activity diagram* dapat juga digunakan untuk memodelkan *action* yang akan dilakukan saat sebuah operasi dijalankan, dan memodelkan hasil dari *action* tersebut.

2.5.3. Relasi (*Relationship*)

Ada 4 (empat) macam *relationship* dalam *Unified Modeling Language* (UML) menurut Nugroho (2010) , yaitu:

a) Ketergantungan (*dependency*) merupakan hubungan dimana perubahan yang terjadi pada suatu elemen mandiri*(independent)* akan mempengaruhi elemen yang bergantung padanya elemen yang tidak mandiri*(dependent)*. Berikut adalah gambar dari relasi *dependency*.

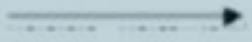
Gambar 2.10 ***Dependency***

b) Asosiasi (*Association)* merupakan apa yang menghubungkan antara objek satu dengan objek lainnya, bagaimana hubungan suatu objek dengan objek lainnya. Suatu bentuk asosiasi adalah agregasi yang menampilkan hubungan suatu objek dengan bagian-bagiannya. Umumnya asosiasi

digambarkan dengan sebuah garis yang dilengkapi dengan sebuah label, nama, dan status hubungannya seperti pada gambar berikut.

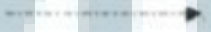
Gambar 2.11 *Association*

c) Generalisasi (*generalization)* merupakan hubungan dimana objek anak (*descendent)* berbagi perilaku dan struktur data dari objek yang ada diatasnya objek induk (*ancestor)*. Arah dari atas ke bawah dari objek induk ke objek anak dinamakan spesialisasi, sedangkan arah berlawanan sebaliknya dari arah bawah ke atas dinamakan generalisasi. Berikut adalah model dari *generalization*.



Gambar 2.12 *Generalization*

d) Realisasi (*realization)* merupakan operasi yang benar-benar dilakukan oleh suatu objek. Hubungan yang ada pada realisasi ini dapat diwujudkan diantara *interface* dan kelas atau *elements*, serta antara *use cases* dan *collaborations*. Berikut adalah model dari *realization.*



Gambar 2.13 *Realization*

2.5.4. Langkah Penggunaan UML

Menurut Henderi (2008), langkah-langkah penggunaan *Unified Modeling Language* (UML) adalah sebagai berikut:

a) Membuat daftar *business process* dari *level* tertinggi untuk mendefinisikan aktivitas dan proses yang mungkin muncul.

b) Melakukan pemetaan terhadap *use case* pada setiap *business process* untuk melakukan pendefinisian dengan tepat mengenai fungsional yang harus disediakan oleh sistem, kemudian perhalus *use case diagram* dan lengkapi dengan *requirement, constraints* dan catatan - catatan lain.

c) Membuat *deployment diagram* secara kasar untuk mendefinisikan arsitektur fisik sistem tersebut.

d) Mendefinisikan *requirement* lain, *security* dan sebagainya yang juga harus disediakan oleh sistem.

e) Membuat *activity diagram* berdasarkan *use case* diagram.

f) Melakukan pendefinisian terhadap obyek *level* dan buatlah *sequence* atau *collaboration* untuk tiap alur pekerjaannya, jika sebuah *use case* memiliki kemungkinan alur normal dan *error*, lakukan pembuatan ulang untuk satu diagram terhadapmasing-masing alur.

g) Melakukan perancangan *user interface model* yang menyediakan antar muka bagi pengguna untuk menjalankan skenario *use case.*

h) Membuat *class diagram* berdasarkan model-model yang sudah ada. Setiap *package* atau *domain* dipecah menjadi hirarki *class* lengkap

dengan atribut dan metodenya. Akan lebih baik jika untuk setiap *class* dibuat *unit test* untuk menguji fungsionalitas *class* dan interaksi dengan *class* lain.

i) Membuat *component diagram* dan definisikan *test* integrasi untuk setiap komponen.

j) Perhalus *deployment diagram* yang sudah dibuat. Detilkan kemampuan dan *requirement* perangkat lunak, sistem operasi, jaringan dan sebagainya. Petakan komponen ke dalam *node.*

k) Membangun sistem. Dalam membangun sistem, ada dua pendekatan yang tepat digunakan yaitu pendekatan *use case* dan pendekatan komponen. Pendekatan *use case* dilakukan dengan melakukan *assignment* terhadap setiap *use case* kepada tim pengembang tertentu untuk mengembangkan unit kode yang lengkap dengan *test.* Sedangkan pendekatan komponen dilakukan dengan cara melakukan *assignment* terhadap setiap komponen kepada tim pengembang tertentu.

2.6. *Waterfall*

Menurut Pressman (2010), model *waterfall* adalah model klasik yang bersifat sistematis, berurutan dalam membangun *software*. Nama model ini sebenarnya adalah "*Linear Sequential Model*". Model ini sering disebut dengan "*classic life cycle*" atau model waterfall. Model ini termasuk kedalam model generic pada rekayasa perangkat lunak dan pertama kali diperkenalkan oleh Winston Royce sekitar tahun 1970 sehingga sering dianggap kuno, tetapi merupakan model yang paling banyak dipakai didalam *Software Engineering* (SE). Model ini melakukan pendekatan secara sistematis dan berurutan. Disebut dengan *waterfall* karena tahap demi tahap yang dilalui harus menunggu selesainya tahap sebelumnya dan berjalan berurutan.

2.7. Definisi Teknik Pengumpulan Data

Berikut adalah definisi dari teknik pengumpulan data menurut para ahli:

a) Teknik pengumpulan data merupakan langkah yang paling strategis dalam penelitian, karena tujuan utama dari penelitian adalah mendapatkan data (Sugiyono, 2013).

b) Teknik pengumpulan data merupakan cara pengumpulan data yang dibutuhkan untuk menjawab rumusan masalah penelitian (Juliansyah Noor, 2011).