BAB II

TINJAUAN PUSTAKA

2.1 Algoritma Pencarian

Penerapan kecerdasan buatan *(Artificial intelligence)* untuk pemecahan masalah *(problem solving)* dalam bidang ilmu komputer telah mengalami perkembangan yang pesat dari tahun ke tahun seiring perkembangan kecerdasan buatan itu sendiri. Permasalahan yang melibatkan pencarian *(searching)* adalah salah satu contoh penggunaan kecerdasan buatan yang cukup populer untuk memecahkan berbagai macam permasalahan (Rian Putra Pratama, 2011).

Penerapannya bermacam-macam, mulai masalah dunia nyata, seperti penentuan rute pada suatu peta, *travelling salesman problem* (TSP), penentuan urutan perakitan *(assembly sequencing)* oleh robot, sampai penerapan dalam dunia *game*, seperti membuat komputer dapat bermain catur layaknya manusia ataupun penentuan pengambilan jalan karakter dalam sebuah *game* (Sunbara Lukito, 2011).

Dalam cara kerjanya, sistem menggunakan algoritma tertentu untuk mencapai kondisi yang diinginkan atau menemukan hasil yang dicari dari kondisi atau input yang ada sekarang. Dalam algoritma pencarian, dikenal istilah '*state'* yang berarti kondisi. Kondisi akhir yang hendak dituju dikenal dengan istilah *'goal state'.* Contoh *state* antara lain, dalam *game* catur misalnya, adalah letak tiap buah catur pada papan. *Goal state* dalam kasus ini biasanya kondisi raja ter-skak mati (Rian Putra Pratama, 2011).

Pada umumnya, algoritma pencarian bekerja dengan mengembangkan berbagai kemungkinan *state* yang mungkin dicapai dari *state* sekarang. *State* dalam proses pencarian biasa disebut dengan istilah *node*. Kumpulan *node* akan terus dikembangkan sampai ditemukan *node* yang merupakan *goal state* atau bila sudah tidak ada lagi *node* yang dikembangkan (berarti tidak ditemukan solusi) (Rian Putra Pratama, 2011).

Rangkaian kumpulan *node* dari *state* awal sampai *goal state* yang terbaiklah yang akhirnya diambil menjadi solusi. Kriteria terbaik di sini tergantung pada kasus yang dihadapi. Pada penentuan rute terbaik misalnya, biasanya adalah solusi yang memberikan jalan terpendek atau tercepat untuk mencapai tujuan (Rian Putra Pratama, 2011).

Berbagai algoritma untuk pencarian *(searching algorithm)* yang ada berbeda satu dengan yang lain dalam hal pengembangan kumpulan *node* untuk mencapai *goal state*. Perbedaan ini terutama dalam hal cara dan urutan pengembangan *node*, dan sangat berpengaruh pada kinerja masing-masing algoritma. Empat kriteria yang menjadi ukuran algoritma pencarian adalah (Rian Putra Pratama, 2011) :

1. *Completeness*, apakah algoritma pasti dapat menemukan solusi?
2. *Time Comlexity*, berapa lama waktu yang dibutuhkan untuk menemukan sebuah solusi?
3. *Space Complexity*, berapa memori atau *resource* yang diperlukan untuk melakukan pencarian?
4. *Optimality*, apakah algoritma tersebut dapat menemukan solusi yang terbaik jika terdapat beberapa solusi yang berbeda?

Menurut cara algoritma mengembangkan *node* dalam proses pencarian, terdapat dua golongan, yakni *Uninformed Search/Blind Search* dan *Informed Search/Heuristic Search* (Russell, Stuart J. & Peter Norvig, 2009).

2.1.1 Uninformed Search atau Blind Search

Sebuah algoritma pencarian *uninformed* adalah algoritma yang tidak mempertimbangkan sifat alami dari permasalahan. Oleh karena itu algoritma tersebut dapat diimplementasikan secara umum, sehingga dengan implementasi yang sama dapat digunakan pada lingkup permasalahan yang luas, hal ini berkat abstraksi. Kekurangannya adalah sebagian besar ruang pencarian adalah sangat besar, dan sebuah pencarian *uninformed* (khususnya untuk pohon) membutuhkan banyak waktu walaupun hanya untuk contoh yang kecil (Russell, Stuart J. & Peter Norvig, 2009).

Beberapa contoh algoritma yang termasuk *Uninformed Search* antara lain adalah: *Uniform Cost Search, Depth First Search, Depth Limited Search, Iterative Deepening Search, Dijkstra.*

2.1.2 Informed Search atau Heuristic Search

Pada pencarian *informed*, sebuah heuristik yang khusus untuk permasalahan tertentu digunakan sebagai pedoman. Sebuah heuristik yang baik dapat membuat sebuah pencarian *informed* bekerja secara dramatis melebihi pencarian *uninformed*. Sebagian besar algoritma *informed* adalah mengeksplor pohon. Sebagaimana algoritma *uninformed*, algoritma *informed* dapat dikembangkan untuk bekerja pada graf (Russell, Stuart J. & Peter Norvig, 2009).

Beberapa contoh algoritma pencarian yang menggunakan metode *Informed Search* adalah *Best First Search, Greedy Search, A* (A Star) Search,* dan *Hill Climbing Search.*

2.2 Algoritma A* (A Star)

Menurut Peter Hart, Nils Nilsson, dan Bertram Raphael yang memperkenalkan algoritma ini. Dalam ilmu komputer, A* (yang diucapkan dengan "*A Star*") merupakan salah satu algoritma pencarian *graph* terbaik yang mampu menemukan jalur dengan biaya pengeluaran paling sedikit atau jarak terdekat dari titik permulaan yang diberikan sampai ke titik tujuan yang diharapkan (dari satu atau lebih mungkin tujuan) (Hart, P. E., N. J. Nilsson & B. Raphael, 1968).

Algoritma ini menggunakan fungsi *distance - plus – cost* (biasanya di notasikan dengan f(x)) untuk menentukan urutan kunjungan pencarian *node* di dalam *tree.* Gabungan jarak - plus – biaya merupakan penjumlahan dari dua fungsi, yaitu fungsi *path - cost* (selalu dinotasikan dengan g(x), dimungkinkan bernilai *heuristic* ataupun tidak), dan sebuah kemungkinan penerimaan atas "perkiraan *heuristic*" jarak ke titik tujuan (dinotasikan dengan h(x)).

Fungsi *path - cost* g(x) adalah jumlah biaya yang harus dikeluarkan dari *node* awal menuju *node* yang akan dituju. Dengan h(x) bagian dari fungsi f(x) yang harus dapat *heuristic* yang mana tidak diperbolehkan untuk terlalu jauh memperkirakan jarak ke arah tujuan. Oleh karena itu untuk aplikasi seperti *routing,* h(x) mungkin mewakili garis lurus jarak ke titik tujuan, karena hal ini secara nyata dimungkinkan adanya jarak terpendek diantara dua titik. Jika

keadaan *heuristic* dari h(x) memenuhi kondisi h(x) lebih kecil sama dengan d(x,y) + h(y) untuk setiap *edge* x,y dari *graph* (dengan d mempresentasikan lebar dari *edge* tersebut), maka h tersebut dikatakan konsisten pada nilainya.

Sedangkan Menurut (Russell, Stuart J. & Peter Norvig, 2009) dalam bukunya, Algoritma A* adalah algoritma *best first search* yang paling banyak dikenal. Algoritma ini memeriksa *node* dengan menggabungkan g(n), yaitu *cost* yang dibutuhkan untuk mencapai sebuah *node* dan h(n) yaitu *cost* yang didapat dari *node* ke tujuan. Sehingga didapatkan rumus algoritma A* ini adalah :

$$\mathbf{F(n) = g(n) + h(n)}\text{……..rumus 2.1}$$

Dimana :

G(n) adalah biaya (*cost*) yang dibutuhkan oleh sebuah jalur (*path*) untuk mencapai *node* n dari *node* awal,

H(n) adalah estimasi biaya (*cost*) sebuah jalur (*path*), dan

F(n) adalah estimasi total biaya (*cost*) sebuah jalur (*path*) dari *node* awal ke *node* tujuan (*goal*) melalui *node* n.

Keoptimalan dari A* ini cukup langsung untuk dianalisa apabila digunakan dengan *tree search*. Pada kasus ini, A* dinilai optimal jika h(n) adalah *admissible heuristic* yaitu nilai h(n) tidak akan memberikan penilaian lebih pada *cost* untuk mencapai tujuan. Salah satu contoh dari *admissible heuristic* adalah jarak dengan menarik garis lurus karena jarak terdekat dari dua titik adalah dengan menarik garis lurus. Jarak dari garis lurus antara dua titik dapat dihitung dengan rumus (Russell, Stuart J. & Peter Norvig, 2009) :

$$h(n) = \sqrt{(x - x1)^2 + (y - y1)^2}$$ ……..rumus 2.2

Dimana :

x = Koordinat x dari *node* awal,

y = Koordinat y dari *node* awal,

x1 = Koordinat x dari *node* lokasi ke n, dan

y1 = Koordinat y dari *node* lokasi ke n.

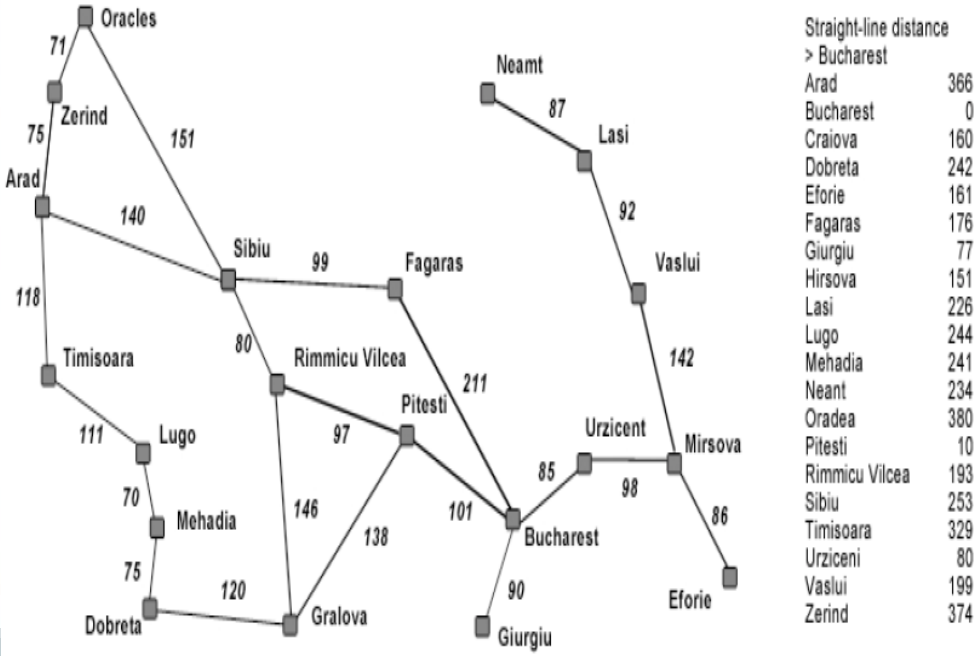
Pseudocode untuk algoritma A* dapat dilihat pada gambar 2.1

```
function A*(start,goal)
    closedset := the empty set    // The set of nodes already evaluated.
    openset := {start}    // The set of tentative nodes to be evaluated, initially containing the start node
    came_from := the empty map    // The map of navigated nodes.

    g_score[start] := 0    // Cost from start along best known path.
    // Estimated total cost from start to goal through y.
    f_score[start] := g_score[start] + heuristic_cost_estimate(start, goal)

    while openset is not empty
        current := the node in openset having the lowest f_score[] value
        if current = goal
            return reconstruct_path(came_from, goal)

        remove current from openset
        add current to closedset
        for each neighbor in neighbor_nodes(current)
            tentative_g_score := g_score[current] + dist_between(current,neighbor)
            if neighbor in closedset and tentative_g_score >= g_score[neighbor]
                    continue

            if neighbor not in openset or tentative_g_score < g_score[neighbor]
                came_from[neighbor] := current
                g_score[neighbor] := tentative_g_score
                f_score[neighbor] := g_score[neighbor] + heuristic_cost_estimate(neighbor, goal)
                if neighbor not in openset
                    add neighbor to openset

    return failure

function reconstruct_path(came_from, current_node)
    if current_node in came_from
        p := reconstruct_path(came_from, came_from[current_node])
        return (p + current_node)
    else
        return current_node
```

Gambar 2.1 *Pseudocode* Algoritma A*

2.2.1 Penerapan Pencarian Algoritma A*

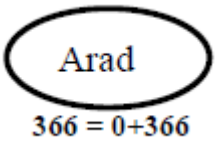
Dalam buku Russel & Norvig. yang berjudul *Artificial Intelligence: A Modern Approach* (1995) hal 63. Berikut adalah contoh penerapan metode A* *Search Algorithm* dalam menyelesaikan suatu permasalahan. Bagaimana cara mencari rute terpendek dari kota Arad menuju Bucharest menggunakan metode *A* Search Algorithm* berdasarkan peta gambar 2.2 dan berdasarkan rumus 2.1



Gambar 2.2 Contoh Penerapan Metode A* Search Algorithm
Sumber: *Artificial Intelligence: A Modern Approach*

Langkah 1 :

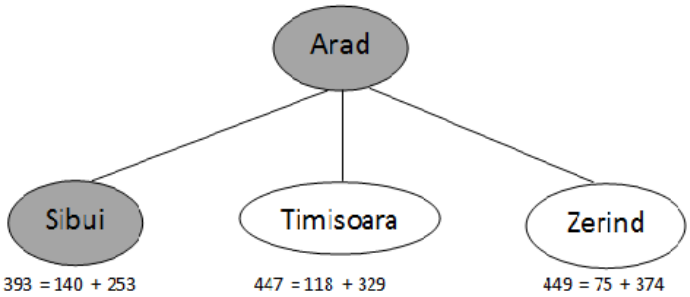
- Menentukan posisi awal, yaitu kota Arad



Gambar 2.3 Contoh Kasus Algoritma A* - Langkah 1
Sumber: *Artificial Intelligence: A Modern Approach*

Langkah 2 :

- Mencari kota-kota yang terhubung dengan kota Arad, yaitu Sibiu, Timisoara, dan Zerind.
- Menentukan *cost* untuk setiap kota yang didapat dengan menggunakan fungsi evaluasi metode *A* Search Algorithm.*
- Mengambil *cost* yang terkecil dari hasil perhitungan di atas. Kota yang diambil adalah Sibiu.

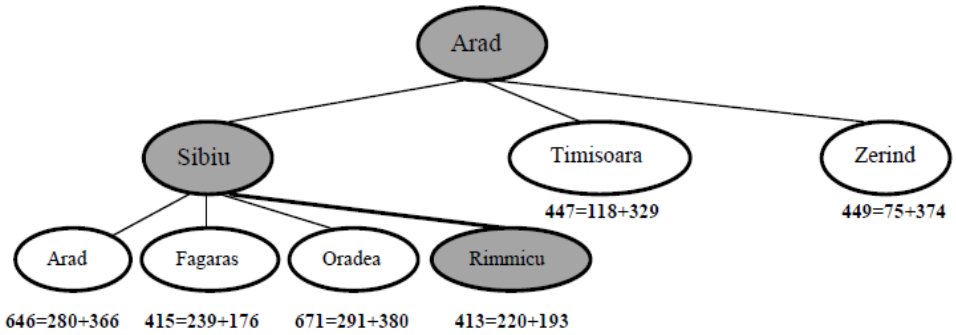


Gambar 2.4 Contoh Kasus Algoritma A* - Langkah 2
Sumber: *Artificial Intelligence: A Modern Approach*

Langkah 3 :

- Mencari kota-kota yang terhubung dengan kota Sibiu, yaitu Arad, Fagaras, Oradea, dan Rimnicu Vilcea.
- Menentukan *cost* untuk setiap kota yang didapat dengan menggunakan fungsi evaluasi metode A* *Search Algorthm.*

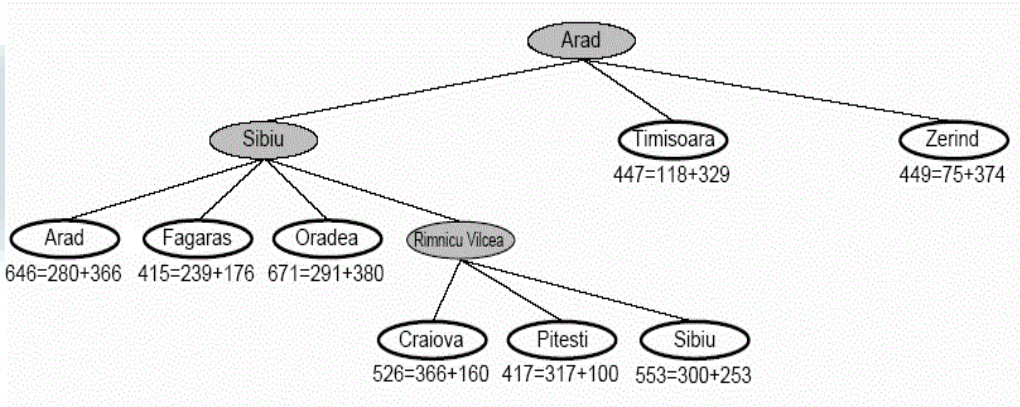
- Mengambil *cost* yang terkecil dari hasil perhitungan di atas. Kota yang diambil adalah Rimmicu Vilcea.



Gambar 2.5 Contoh Kasus Algoritma A* - Langkah 3
Sumber: *Artificial Intelligence: A Modern Approach*

Langkah 4 :

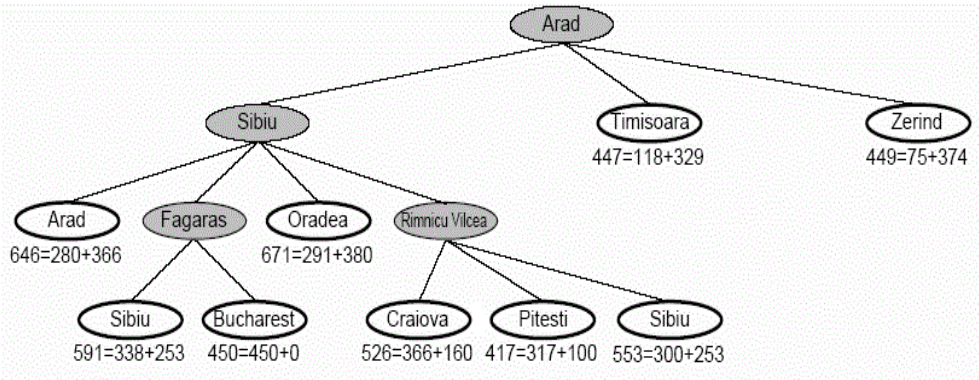
- Mencari kota-kota yang terhubung dengan kota Rimnicu Vilcea, yaitu Craiova, Pitesti, dan Sibiu.
- Menentukan *cost* untuk setiap kota yang didapat dengan menggunakan fungsi evaluasi metode A* *Search Algorithm.*
- Mengambil *cost* yang tekecil dari hasil perhitungan di atas. Kota yang diambil adalah Fagaras.



Gambar 2.6 Contoh Kasus Algoritma A* - Langkah 4
Sumber: *Artificial Intelligence: A Modern Approach*

Langkah 5 :

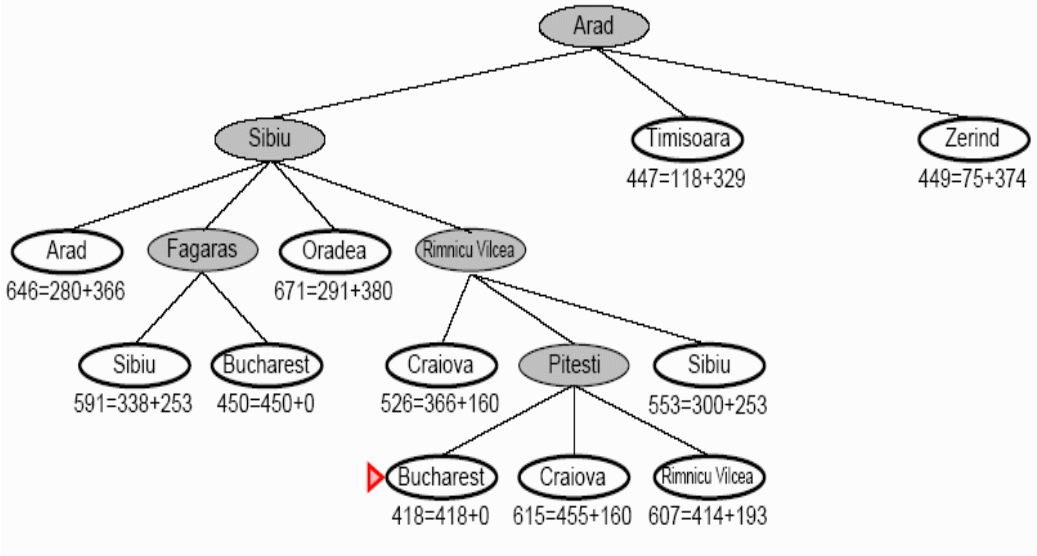
- Mencari kota-kota yang terhubung dengan kota Fagaras, yaitu Sibiu dan Bucharest.
- Menentukan *cost* untuk setiap kota yang didapat dengan menggunakan fungsi evaluasi metode A* *Search Algorithm.*
- Mengambil *cost* yang tekecil dari hasil perhitungan di atas. Kota yang diambil adalah Pitesti.



Gambar 2.7 Contoh Kasus Algoritma A* - Langkah 5
Sumber: *Artificial Intelligence: A Modern Approach*

Langkah 6:

- Mencari kota-kota yang terhubung dengan kota Pitesti, yaitu Bucharest, Craiova, dan Rimnicu Vilcea.
- Menentukan *cost* untuk setiap kota yang didapat dengan menggunakan fungsi evaluasi metode A* *Search Algorithm.*
- Mengambil *cost* yang tekecil dari hasil perhitungan di atas. Kota yang diambil adalah Bucharest. Karena *Goal* telah tercapai maka langkah berhenti sampai di sini.



Gambar 2.8 Contoh Kasus Algoritma A* - Langkah 6
Sumber: *Artificial Intelligence: A Modern Approach*

Jadi, hasil pencarian rute terpendek dari kota Arad menuju ke kota Bucharest dengan menggunakan metode A* *Search Algorithm* adalah Arad – Sibiu – Rimnicu Vilcea – Pitesti – Bucharest dengan total *cost* 418.

2.3 MySQL

MySQL adalah sebuah perangkat lunak sistem manajemen *database* SQL atau DBMS yang *multithread, multi-user*. MySQL mendukung operasi *database* transaksional maupun operasi *database* non-transaksional. Pada modus operasi non-transaksional, MySQL dapat dikatakan unggul dalam hal unjuk kerja dibandingkan perangkat lunak peladen *database* kompetitor lainnya. Namun demikian pada modus non-transaksional tidak ada jaminan atas reliabilitas terhadap data yang tersimpan, karenanya modus non-transaksional hanya cocok untuk jenis aplikasi yang tidak membutuhkan reliabilitas data seperti aplikasi

blogging berbasis *web* (wordpress), CMS, dan sejenisnya (Achmad Solichin, 2010).

2.3.1 Keunggulan MySQL

MySQL memiliki beberapa keistimewaan, antara lain :

1. Portabilitas. MySQL dapat berjalan stabil pada berbagai sistem operasi seperti Windows, Linux, FreeBSD, Mac Os X Server, Solaris, Amiga, dan masih banyak lagi.
2. Perangkat lunak sumber terbuka. MySQL didistribusikan sebagai perangkat lunak sumber terbuka, dibawah lisensi GPL sehingga dapat digunakan secara gratis.
3. *Multi-user*. MySQL dapat digunakan oleh beberapa pengguna dalam waktu yang bersamaan tanpa mengalami masalah atau konflik.
4. *Performance tuning*, MySQL memiliki kecepatan yang menakjubkan dalam menangani *query* sederhana, dengan kata lain dapat memproses lebih banyak SQL per satuan waktu.
5. Ragam tipe data. MySQL memiliki ragam tipe data yang sangat kaya, seperti *signed / unsigned integer, float, double, char, text, date, timestamp*, dan lain-lain.
6. Perintah dan fungsi. MySQL memiliki operator dan fungsi secara penuh yang mendukung perintah *Select* dan *Where* dalam perintah (*query*).
7. Keamanan. MySQL memiliki beberapa lapisan keamanan seperti *level subnetmask*, nama *host*, dan izin akses *user* dengan sistem perizinan yang mendetail serta sandi terenkripsi.

8. Skalabilitas dan pembatasan. MySQL mampu menangani *database* dalam skala besar, dengan jumlah rekaman (*records*) lebih dari 50 juta dan 60 ribu tabel serta 5 milyar baris. Selain itu batas indeks yang dapat ditampung mencapai 32 indeks pada tiap tabelnya.
9. Konektivitas. MySQL dapat melakukan koneksi dengan klien menggunakan protokol TCP/IP, Unix soket (UNIX), atau *Named Pipes* (NT).
10. Lokalisasi. MySQL dapat mendeteksi pesan kesalahan pada klien dengan menggunakan lebih dari dua puluh bahasa. Meski pun demikian, bahasa Indonesia belum termasuk di dalamnya.
11. Antar muka. MySQL memiliki antar muka (*interface*) terhadap berbagai aplikasi dan bahasa pemrograman dengan menggunakan fungsi API (*Application Programming Interface*).
12. Klien dan peralatan. MySQL dilengkapi dengan berbagai peralatan (*tool*) yang dapat digunakan untuk administrasi *database*, dan pada setiap peralatan yang ada disertakan petunjuk online.
13. Struktur tabel. MySQL memiliki struktur tabel yang lebih fleksibel dalam menangani *alter table*, dibandingkan basis data lainnya semacam PostgreSQL ataupun Oracle.

2.3.2 Syntak pada MySQL

Sama seperti *database* lainnya syntak SQLnya terbagi 2 kategori yaitu syntak untuk *Data Definition Language* (DDL) dan *Data Manipulation Language* (DML).

1. DDL - *Data Definition Language*

   DDL adalah kumpulan perintah SQL yang digunakan untuk membuat (*create*), mengubah (*alter*), dan menghapus (*drop*) struktur dan definisi tipe data dari objek-objek *database*

2. DML - *Data Manipulation Language*

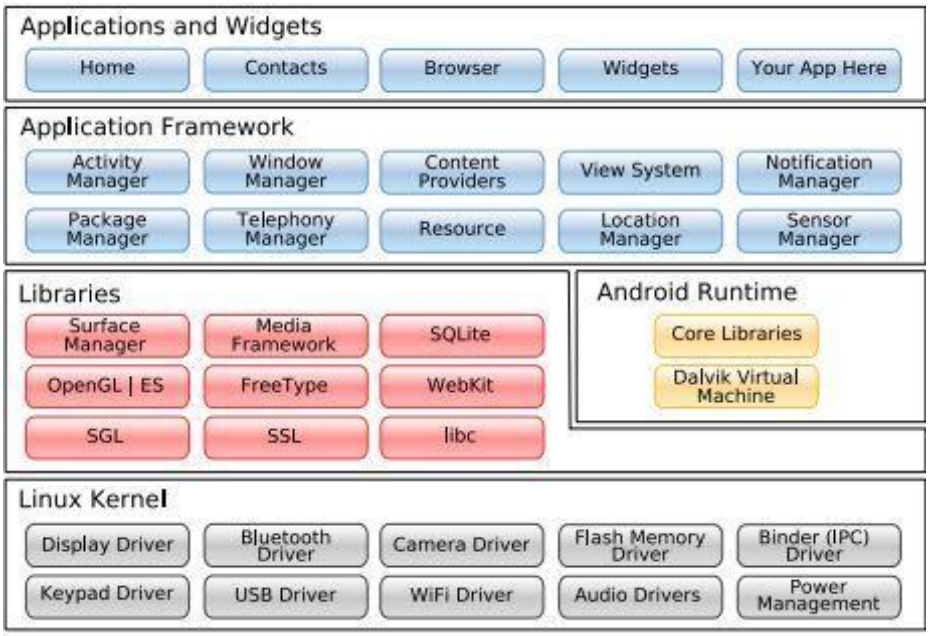
   DML sendiri adalah kumpulan perintah SQL yang berhubungan dengan pekerjaan mengolah data di dalam table dan tidak terkait dengan perubahan struktur dan definisi tipe data dari objek database seperti table, column, dan sebagainya.

2.4 Android

Android adalah sebuah sistem operasi untuk perangkat *mobile* berbasis linux yang mencakup sistem operasi, *middleware*, dan aplikasi. Android menyediakan platform terbuka bagi para pengembang untuk menciptakan aplikasi mereka. Android adalah *platform* yang sangat lengkap baik itu sistem operasinya, aplikasi dan *tool* pengembangan, *market* aplikasi Android *serta* dukungan yang sangat tinggi dari komunitas *open source* di dunia (H., Nazruddin Safaat, 2012).

2.4.1 Arsitektur Android

Secara sederhana arsitektur Android merupakan sebuah kernel Linux dan sekumpulan pustaka C / C++ dalam suatu framework yang menyediakan dan mengatur alur proses aplikasi, seperti yang ditunjukkan pada gambar 2.9 (H., Nazruddin Safaat, 2012).



Gambar 2.9 Arsitektur Android
Sumber: Android Pemrograman Aplikasi Mobile Smartphone dan Tablet PC Berbasis Android (edisi revisi)

1. Linux Kernel

   Android dibangun di atas kernel Linux 2.6. Linux merupakan sistem operasi terbuka yang handal dalam manajemen memori dan proses.

2. *Libraries*

   Android menggunakan beberapa paket pustaka yang terdapat pada C/C++ dengan standar *Berkeley Software Distribution* (BSD) hanya setengah dari yang aslinya untuk tertanam pada kernel Linux. Beberapa pustaka diantaranya:

   a. *Media Library* untuk memutar dan merekam berbagai macam format *audio* dan *video*.
   b. *Surface Manager* untuk mengatur hak akses *layer* dari berbagai aplikasi.
   c. *Graphic Library* termasuk didalamnya SGL dan OpenGL, untuk tampilan 2D dan 3D.

d. SQLite untuk mengatur relasi *database* yang digunakan pada aplikasi.

e. SSl dan WebKit untuk browser dan keamanan internet.

Pustaka-pustaka tersebut bukanlah aplikasi yang berjalan sendiri, namun hanya dapat digunakan oleh program yang berada di level atasnya. Sejak versi Android 1.5, pengembang dapat membuat dan menggunakan pustaka sendiri menggunakan Native Development Toolkit (NDK).

3. Android *Runtime*

a. *Core Libraries,* Android menyediakan hampir semua fungsi yang terdapat pada pustaka Java serta beberapa pustaka khusus Android.

b. *Dalvik Virtual Machine*, adalah *virtual machine* yang mengoptimalkan efisiensi penyimpanan dan pengalamatan memori pada file yang dieksekusi. Dalvik berjalan di atas kernel Linux 2.6, dengan fungsi dasar seperti *threading* dan manajemen memori yang terbatas.

4. Application Framework

Kerangka aplikasi menyediakan kelas-kelas yang dapat digunakan untuk mengembangkan aplikasi Android. Selain itu, juga menyediakan abstraksi generik untuk mengakses perangkat, serta mengatur tampilan *user interface* dan sumber daya aplikasi.

5. Application Layer

Lapisan aplikasi merupakan lapisan yang paling tampak pada pengguna ketika menjalankan program. Lapisan ini berjalan dalam Android *runtime* dengan menggunakan kelas dan *service* yang tersedia pada *framework* aplikasi.

Pada Android semua aplikasi, baik aplikasi inti (native) maupun aplikasi pihak ketiga berjalan di atas lapisan aplikasi dengan menggunakan pustaka API (Application Programming Interface) yang sama.

2.5 Global Positioning System

Global Positioning System atau GPS adalah sistem untuk menentukan letak di permukaan bumi dengan bantuan penyelarasan (*synchronization*) sinyal satelit. Sistem ini menggunakan 24 satelit yang mengirimkan sinyal gelombang mikro ke Bumi. Sinyal ini diterima oleh alat penerima di permukaan, dan digunakan untuk menentukan letak, kecepatan, arah, dan waktu. Sistem ini dikembangkan oleh departemen pertahanan Amerika Serikat, dengan nama lengkapnya adalah NAVSTAR GPS (Andi, Tanpa tahun)**.**

GPS Tracker atau sering disebut dengan GPS Tracking adalah teknologi AVL (*Automated Vehicle Locater*) yang memungkinkan pengguna untuk melacak posisi kendaraan, armada ataupun mobil dalam keadaan Real-Time. GPS Tracking memanfaatkan kombinasi teknologi GSM dan GPS untuk menentukan koordinat sebuah obyek (Andi, Tanpa tahun).