BAB II
KAJIAN PUSTAKA

2.1 Metode A* (A-Star)

Menurut (Russel & Norvig, 2003) 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 dasar dari algoritma A* ini adalah:

$f(n) = g(n) + h(n)$

Dalam notasi standar yang dipakai untuk algoritma A* di atas, digunakan g(n) untuk mewakili *cost* rute dari *node* awal ke *node n*. Lalu h(*n*) mewakili perkiraan *cost* dari *node* n ke *node goal*, yang dihitung dengan fungsi *heuristic*. A* "menyeimbangkan" kedua nilai ini dalam mencari jalan dari *node* awal ke *node goal*. Dalam menentukan *node* yang akan dikembangkan, algoritma ini akan memilih *node* dengan nilai $f(n) = g(n) + h(n)$ yang paling kecil.

Algorima A*, dapat juga disebut sebagai Algoritma A Star, merupakan salah satu contoh algoritma pencarian yang cukup popular di dunia. Jika kita mengetikkan Algoritma A* pada sebuah mesin pencari, seperti *google.com*, maka akan kita temukan lebih dari sepuluh ribu literatur mengenai algoritma A*. Beberapa terminologi dasar yang terdapat pada algoritma ini adalah *starting point*, simpul (*nodes*) A, *open list*, *closed list*, harga (*cost*), halangan (*unwalkable*). *Starting point* adalah sebuah terminologi untuk posisi awal sebuah benda. A adalah simpul yang sedang dijalankan dalam algortima pencarian jalan terpendek. Simpul adalah petak-petak kecil sebagai representasi dari area *pathfinding*. Bentuknya dapat berupa persegi, lingkaran, maupun segitiga. *Open list* adalah tempat menyimpan data simpul yang mungkin diakses dari *starting point* maupun simpul yang sedang dijalankan. *Closed list* adalah tempat menyimpan data simpul sebelum A yang juga merupakan bagian dari jalur terpendek yang telah berhasil didapatkan. Harga (F) adalah nilai yang diperoleh dari penjumlahan nilai G, jumlah nilai tiap simpul dalam jalur terpendek dari *starting point* ke A, dan H, jumlah nilai perkiraan dari sebuah simpul ke simpul tujuan. Simpul tujuan yaitu simpul yang dituju. Rintangan adalah sebuah atribut yang menyatakan bahwa sebuah simpul tidak dapat dilalui oleh A.

Prinsip algoritma ini adalah mencari jalur terpendek dari sebuah simpul awal (*starting point)* menuju simpul tujuan dengan memperhatikan harga (F) terkecil. Diawali dengan menempatkan A pada *starting point*, kemudian memasukkan seluruh simpul yang bertetangga dan tidak memilik atribut rintangan dengan A ke dalam *open list*. Kemudian mencari nilai H terkecil dari simpul-simpul dalam *open list* tersebut. Kemudian memindahkan A ke simpul yang memiliki nilai H terkecil. Simpul sebelum A disimpan sebagai *parent* dari A dan dimasukkan kedalam *closed list*. Jika terdapat simpul lain yang bertetangga dengan A (yang sudah berpindah) namun belum termasuk ke dalam anggota *open list*, maka masukkan simpul-simpul tersebut ke dalam *open list.* Setelah itu, bandingkan nilai G yang ada dengan nilai G sebelumnya (pada langkah awal, tidak perlu dilakukan perbandingan nilai G). Jika nilai G sebelumnya lebih kecil maka A kembali ke posisi awal. Simpul yang pernah dicoba dimasukkan ke dalam *closed list.* Hal terebut dilakukan berulang-ulang hingga terdapat solusi atau tidak ada lagi simpul lain yang berada pada *open list*. Kelebihan algoritma A* antara lain :

1. Waktu pencarian algoritma A* dalam menemukan rute lebih cepat dibanding algoritma pencarian yang lain.
2. Jumlah *Loop* A star lebih sedikit.
3. Rute yang ditemukan dapat berbeda tapi mempunyai biaya yang sama.

Prinsip kerja algoritma A* dapat dijelaskan dengan *pseudocode* dibawah ini :

1. Masukan node awal ke *openlist*.
2. *Loop* Langkah – langkah di bawah ini :
    a. Cari *node* (n) dengan nilai f(n) yang paling rendah dalam *openlist*. *Node* ini sekarang menjadi *current node*.
    b. Keluarkan *current node* dari *openlist* dan masukan ke *closelist*.
    c. Untuk setiap tetangga dari *current node* lakukan berikut :
        i. Jika tidak dapat dilalui atau sudah ada dalam *closelist*, abaikan.
        ii. Jika belum ada di *openlist* . Buat *current* node *parent* dari *node* tetangga ini. Simpan nilai f,g dan h dari *node* ini.

iii. Jika sudah ada di *openlist*, cek bila *node* tetangga ini lebih baik, menggunakan nilai g sebagai ukuran. Jika lebih baik ganti *parent* dari *node* ini di *openlist* menjadi *current node*, lalu kalkulasi ulang nilai g dan f dari *node* ini.

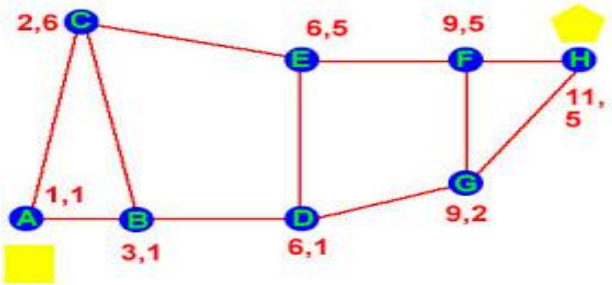
d. Hentikan *loop* jika :

i. *Node* tujuan telah ditambahkan ke *openlist*, yang berate rute telah ditemukan.

ii. Belum menemukan *node goal* sementara *openlist* kosong atau berarti tidak ada rute.

3. Simpan rute. Secara '*backward*', urut mulai dari *node goal* ke *parent*- nya terus sampai mencapai *node* awal sambil menyimpan *node* ke dalam sebuah *array*.

Pertama, kita misalkan ada persoalan menentukan rute seperti dibawah ini :



Gambar 2.1 Contoh Graph Rute.

Segi lima berwarna kuning adalah tempat tujuan dan kotak kuning adalah tempat asal, sedangkan titik biru adalah persimpangan setiap jalan. Setiap titik memiliki nilai X dan Y yang digunakan untuk perhitungan.

Langkah - langkah penentuan rute di atas adalah :

i. Karena A adalah persimpangan terdekat dari tempat asal maka A masuk ke *closelist*.

ii. Sedangkan H adalah persimpang terdekat dengan tempat tujuan, maka H masuk ke *closelist*. Kemudian dilakukan pengecekan percabangan dari A yaitu B dan C.B dan C masuk ke *openlist*

iii. Dilakukan perhintungan dengan algoritma A* diantara B dan C mana yang lebih dekat dengan H. Rumusnya :

$$d(x,y) = |(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_n - y_n)^2| = \sqrt{\sum_{i=1}^{n}}$$

Hasil pehitungan adalah :

B - H = √(11 - 5)2 + (3 - 1)2 + (11 - 3)2 + (5 - 1)2 = 15.26882723

C - H = √(11 - 5)2 + (2 - 6)2 + (11 - 2)2 + (5 - 6)2 = 16.26648769

Maka persimpangan yang dipilih adalah B, maka B masuk ke *closelist* dan mengkosongkan *openlist.*

4. Lalu dilakukan pengecekan apakah H sudah masuk ke *closelist*, jika belum maka dilakukan proses seperti no 2 - 6 hingga H masuk ke *closelist*.

5. Sehingga jalur yang ditemukan adalah A - B - D - G - F.

Kelebihan algoritma A* antara lain :

i. Waktu pencarian algoritma A* dalam menemukan rute lebih cepat dibanding algoritma pencarian yang lain.
ii. Jumlah *loop* A star lebih sedikit.
iii. Rute yang ditemukan dapat berbeda tapi mempunyai biaya yang sama.

2.2 Rute Pendek

Lintasan terpendek adalah lintasan minimum yang diperlukan untuk mencapai suatu tempat dari tempat tertentu. Lintasan minimum yang dimaksud dapat dicari dengan menggunakan *graf*. *Graf* yang digunakan adalah *graf* yang tidak berbobot, yaitu *graf* yang tidak memiliki suatu nilai atau bobot.

Ada beberapa macam persoalan lintasan terpendek, antara lain:

1. Lintasan terpendek antara dua buah simpul tertentu (*a pair shortets path*).
2. Lintasan terpendek antara semua pasangan simpul (*all pairs shortest path*).
3. Lintasan terpendek dari simpul tertentu ke semua simpul yang lain (*single source shortest path*).

4. Lintasan terpendek antara dua buah simpul yang melalui beberapa simpul tertentu (*intermediate shortest path*).

2.3 Java

Java menurut definisi dari *Sun* adalah nama untuk sekumpulan teknologi untuk membuat dan menjalankan perangkat lunak pada komputer *standalone* ataupun pada lingkungan jaringan. Java dikembangkan pada bulan Agustus 1991, dengan nama semula *Oak*. Pada Januari 1995, karena nama *Oak* dianggap kurang komersial, maka diganti menjadi Java. Pada Desember 1998, *Sun* memperkenalkan nama *"Java 2"* (J2) sebagai generasi kedua dari *java platform*. Konvensi nama baru ini diterapkan untuk semua edisi Java yaitu *Standard Edition* (J2SE), *Enterprise Edition* (J2EE), dan *Micro Edition* (J2ME). Ada tiga platform Java yang telah didefinisikan yang masing-masing diarahkan untuk tujuan tertentu dan untuk lingkungan komputasi yang berbeda-beda :

1. Standard Edition (J2SE),

   J2SE merupakan inti dari bahasa pemrograman Java. J2SE Didesain untuk jalan pada komputer desktop dan komputer *workstations*.

2. Enterprise Edition (J2EE),

   Dengan *built-in* mendukung untuk *servlets*. JSP, dan XML, edisi ini ditujukan untuk aplikasi berbasis *server*.

3. Micro Edition (J2ME).

   Didesain untuk piranti dengan memori terbatas, layar display terbatas dan power pemrosesan yang juga terbatas.

Beberapa kelebihan Java :

1. Sederhana dan Ampuh.

Java dirancang untuk mudah dipelajari, terutama bagi programmer yang telah mengenal C/C++ akan mudah sekali untuk berpindah ke Java Pemakai dapat belajar membuat program dengan Java secara cepat jika telah memahami konsep dasar

pemrograman berorientasi objek. Java tidak memiliki hal-hal yang mengejutkan dan aneh. Java memberi anda kemampuan untuk menuangkan semua ide, karena bahasa pemrograman ini bukan merupakan *scripting language* (bahasa naskah) yang menghilangkan kemampuan kita untuk berinovasi, tetapi dengan cara berorientasi objek yang mudah dan jelas.

2. Aman.

Java dirancang sebagai bahasa pemrograman yang handal dan aman. Aplikasi-aplikasi yang dibangun dengan bahasa Java sangat handal dengan manajemen memori yang bagus. Aplikasi Java juga dikenal sangat *secure*, yaitu kasus-kasus seperti *buffer everflow* yang umumnya menjadi lubang keamanan aplikasi-aplikasi berbasis C/C++ tidak terjadi di Java, karena pengaturan *security* nya yang bagus. Seperti yang kita ketahui ancaman virus dan penyusup ada dimana-mana, bahkan dokumen *word processor* dapat mengandung virus. Salah satu prinsip kunci perancangan Java adalah keselamatan dan keamanan. Java tidak pernah memiliki fasilitas dan kemampuan yang tidak aman sampai perlu ditangani secara khusus untuk pengamanannya. Maka karena program Java tidak dapat memanggil fungsi-fungsi global dan memperoleh akses ke berbagai sumber dalam sistem, terdapat sejumlah pengawasan yang dapat dilakukan oleh program Java yang tidak dapat dilakukan oleh sistem lain.

3. Berorientasi Objek.

Paradigma pemrograman berorientasi objek merupakan paradigma pemrograman masa depan. Java merupakan bahasa pemrograman berorientasi objek. Java bukan turunan langsung dari bahasa pemrograman manapun, juga sama sekali tidak kompetibel dengan semuanya. Java memiliki keseimbangan menyediakan mekanisme peng-*class*-an sederhana, dengan model antar muka dinamik yang intuitif hanya jika diperlukan

4. Kokoh.

Java membatasi anda dari beberapa hal kunci supaya anda dapat menemukan kesalahan lebih cepat saat mengembangkan program. Java langsung memeriksa program saat anda menuliskannya, dan sekali lagi ketika program di jalankan. Karena Java adalah bahasa yang sangat ketat dalam hal tipe data dan deklarasi, banyak kesalahan umum terjadi saat kompilasi. Hal ini akan lebih menghemat waktu jika

dibandingkan dengan keharusan menjalankan program terlebih dahulu dan memeriksa semua bagian program untuk melihat ketidakcocokan dinamis selama program berjalan. Ini adalah contoh di mana Java lebih luwes dan kokoh dari beberapa bahasa lain, tetapi dengan imbalan yang layak untuk kelebihan itu.

5. Interaktif

Java memiliki beberapa kemampuan yang memungkinkan program melakukan beberapa hal pada saat bersamaan, tanpa harus kesulitan menangani proses yang akan terjadi selanjutnya. Jalinan program-program Java yang mudah digunakan memungkinkan kita untuk memikirkan pembuatan perilaku khusus, tanpa harus mengintegrasikan perilaku tersebut dengan model pemrograman global yang mengatur perulangan kejadian.

6. Netral Terhadap Berbagai Arsitektur Java

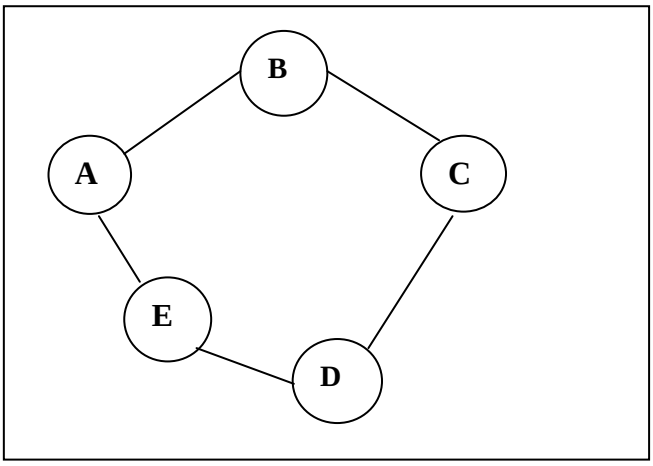
telah mengambil beberapa keputusan yang sulit dalam pembuatan bahasa Java dan bagaimana program dijalankan, jadi anda dapat sepenuhnya percaya "tulis sekali, jalan di mana saja, kapan saja, selamanya"

7. Terintepretasi Dan Berkinerja Tinggi

Java dilengkapi keajaiban lintas-*platform* yang luar biasa dengan kompilasi ke dalam representasi langsung yang disebut kode-*byte* Java (*Java byte-code*), yang dapat diterjemahkan oleh sistem manapun yang memilki program Java didalamnya. Java, bagaimanapun dirancang untuk tetap berkinerja baik pada CPU yang tidak terlalu kuat. Walaupun Java merupakan bahasa terinterpretasi, kode-kode Java telah dirancang dengan hati-hati sehingga mudah diterjemahkan ke dalam bahasa asli suatu mesin untuk menghasilkan kinerja yang tinggi. Sistem program Java yang melakukan optimasi tepat waktu tersebut tidak kehilangan keuntungan dari program yang netral terhadap platform. "lintas platform berkinerja tinggi" bukan sekedar omong-kosong. Dalam aplikasi Java *(*.class*) merupakan *Java bytecode* yang berjalan di atas jvm (*Java Virtual Machine*), yang kemudian jum-lah yang akan menginterpresentasikan kode-kode tersebut ke kode *native* atau kode mesin dari arsitektur yang bersangkutan. Hal ini sangat menarik karena urusan arsitektur mesin bukan jadi masalah bagi programmer tapi menjadi urusan kompiler pada bahasa pemrograman Java.

2.4 Graph

Teori graph adalah cabang utama dari matematika kombinasi dan sudah dipelajari ratusan tahun yang lalu. Menurut Sedgewik (1998, p416) graph adalah koleksi dari vertices and edges. Vertices adalah objek sederhana yang dapat memiliki nama dan property, edge adalah koneksi yang menghubungkan vertices. Sedangan menurut Wiitala (1987,p178) graph adalah pasangan berurutan (V,E) yang mana V disebut sebagai kumpulan vertex dan E adalah kumpulan yang berisi dua elemen subset dari V. E dapat dibilang sebagai edge set apabila memiliki garis yang menghubungkan antara yang satu dengan yang lain.

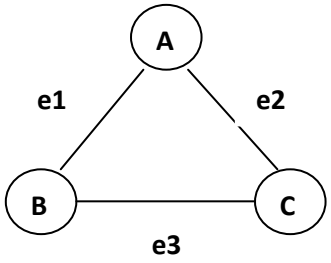


Gambar 2.2 Vertex dan edge pada Graph

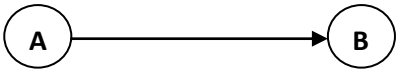
V = {A, B, C, D, E}

E = {(A,B), (A,E), (B,C), (C,D), (D,E)}

Pada gambar 2.5, edge (A,B) dapat direpresentasikan dengan e1, edge (B,C) dapat direpresentasikan dengan e2, edge (C,D) dapat direpresentasikan dengan e3, demikian seterusnya sampai dengan edge (E,A) sebagai e5.

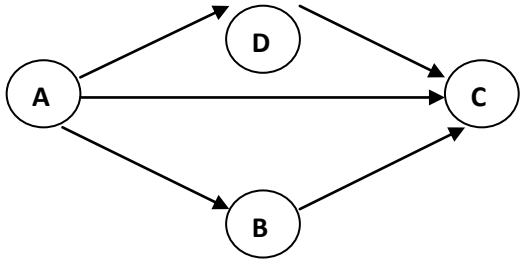


Gambar 2.3 Undirected Graph



Gambar 2.4 Directed Graph

`Directed graph dan undirected graph memiliki perbedaan dalam hal tanda `panah pada tiap edge yang mana menunjukkan arah dari edge. Seperti yang dilihat pada gambar directed graph memiliki tanda panah dari vertex A menuju vertex B.



Gambar 2.5 Outdegree

Dalam sebuah graph, dikatakan sebuah vertex adalah outdegree apabila sejumlah arah panah mengarah keluar dari vertex tersebut dan sebuah vertex dikatakan indegree apabila sejumlah arah panah mengacu ke vertex tersebut.

Pada contoh gambar 2.6, banyaknya outdgree dari node A adalah 3, dapat ditulis outdeg (3) atau outdeg (A) = 3. Kesimpulannya, arah panah yang keluar dari node A adalah sebanyak tiga. Sedangkan untuk indegree dari gambar 2.7 dari node C adalah 3 dan dapat ditulis indeg (C) = 3.