Hak cipta dan penggunaan kembali:

Lisensi ini mengizinkan setiap orang untuk menggubah, memperbaiki, dan membuat ciptaan turunan bukan untuk kepentingan komersial, selama anda mencantumkan nama penulis dan melisensikan ciptaan turunan dengan syarat yang serupa dengan ciptaan asli.

Copyright and reuse:

This license lets you remix, tweak, and build upon work non-commercially, as long as you credit the origin creator and license it on your new creations under the identical terms.

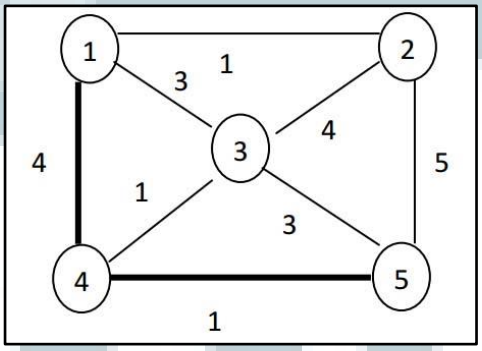
Team project ©2017
Dony Pratidana S. Hum | Bima Agus Setyawan S. IIP

BAB II

LANDASAN TEORI

2.1 Lintasan Terpendek (Shortest Path)

Lintasan terpendek adalah lintasan yang memiliki total bobot minimum untuk mencapai suatu tempat dari tempat tertentu. Lintasan terpendek merupakan salah satu masalah yang dapat diselesaikan dengan menggunakan algoritma *graph* (Putu Agus Indra Purnama, 2008). Jika diberikan sebuah *graph* berbobot, masalah lintasan terpendek adalah bagaimana mencari sebuah jalur pada *graph* yang meminimumkan jumlah biaya pembentuk jalur tersebut. Gambar di bawah ini menunjukkan lintasan terpendek (garis tebal) dari simpul 1 ke simpul 5 dengan jalur 1-4-5 dengan jumlah bobot 5 (lima).



Gambar 2.1 *Shortest Path* (Garis Tebal)
(sumber: Simulasi Pencarian Jalur Terpendek
Menggunakan Algoritma Dijkstra, Giri Mega Gentara, 2012)

Menurut Putu Agus Indra Purnama (2008) Ada beberapa macam persoalan lintasan terpendek, antara lain:

- lintasan terpendek antara dua buah simpul tertentu (*a pair shortets path*).
- lintasan terpendek antara semua pasangan simpul (*all pairs shortest path*).

- lintasan terpendek dari simpul tertentu ke semua simpul yang lain (*single-source shortest path*).
- lintasan terpendek antara dua buah simpul yang melalui beberapa simpul tertentu (*intermediate shortest path*).

Graph adalah kumpulan simpul (*nodes*) yang dihubungkan satu sama lain melalui sisi/busur (*edges*). Suatu *Graph* G terdiri dari dua himpunan yaitu himpunan V dan himpunan E. Keterangan:

- *Verteks* (simpul) : V = himpunan simpul yang terbatas dan tidak kosong.
- *Edge* (sisi/ruas) : E = himpunan busur yang menghubungkan sepasang simpul.

2.2 Pencari Lintasan (Pathfinding)

Menurut Russel, Stuart dan Peter Norvig dalam buku Putu Agus Indra Purnama (2008) pengertian *pathfinding* adalah menemukan jalur terbaik dari *vertex* awal ke *vertex* akhir. Pendekatan masalah *pathfinding* secara umum dimodelkan dalam bentuk kota-kotak persegi dan diimplementasikan ke dalam algoritma sebagai pemecahan masalah jalur terpendek. Secara umum algoritma *pathfinding* digolongkan menjadi dua jenis, yaitu:

- Algortima *Uniform Search*

  Adalah algoritma yang tidak memiliki keterangan tentang jarak atau biaya dari jalur dan tidak memiliki pertimbangan akan jalur mana yang lebih baik. Yang termasuk dalam algoritma ini *BFS* (*Breadth First Search*).
- Algortima *Informed Search*

  Adalah algoritma yang memiliki keterangan tentang jarak atau biaya dari jalur dan memililki pertimbangan berdasarkan pengatahuan akan jalur mana yang

lebih baik. Yang termasuk dalam algoritma ini adalah algoritma *Dijktstra* dan algoritma *A** (*A-Star*).

Penerapan yang dapat dilakukan dengan P*athfinding* antara lain pencarian rute dalam suatu aplikasi permainan dan pencarian jalan/rute pada suatu peta digital. Algoritma pencarian yang dipakai harus dapat mengenali jalan dan elemen peta yang tidak dapat dilewati.

2.3 Alogritma A* (A-Star)

Algoritma ini pertama kali diperkenalkan pada 1968 oleh Peter Hart, Nils Nilsson, dan Bertram Raphael. Dalam ilmu komputer, merupakan format pencarian heuristik untuk menghitung efisiensi solusi optimal. Algoritma *A** adalah algoritma *Best First Search* dimana *cost* yang terkait dengan node adalah $f(n) = g(n) + h(n)$ (Hapsari Tilawah, 2013), dan $g(n)$ adalah biaya jalur dari lokasi awal ke *node* n dan $h(n)$ adalah perkiraan heuristik atau perkiraan jalur dari *node* n ke tujuan. Jadi, $f(n)$ adalah perkiraan total biaya terendah dari setiap jalur yang akan dilalui *node* n ke *node* tujuan. Dengan kata lain, biaya adalah jarak yang telah ditempuh, dan panjang garis lurus antara *node* n dengan *node* akhir adalah perkiraan heuristiknya. Semakin rendah nilai $f(n)$, semakin tinggi prioritasnya. Atribut g, h, dan f didefinisikan sebagai berikut: (sumber: Kalsum Mustika Nasution, 2012)

- g adalah biaya yang didapatkan dari node awal ke node saat ini yaitu jumlah dari semua nilai di jalan antara awal dan node saat ini.
- h singkatan heuristik yang merupakan perkiraan biaya dari node saat ini ke tujuan node (biasanya jarak garis lurus dari node ini ke tujuan).

- f adalah jumlah dari g dan h dan merupakan estimasi terbaik dari biaya jalan akan melalui node saat ini. Intinya semakin rendah nilai f yang akan lebih efisien.

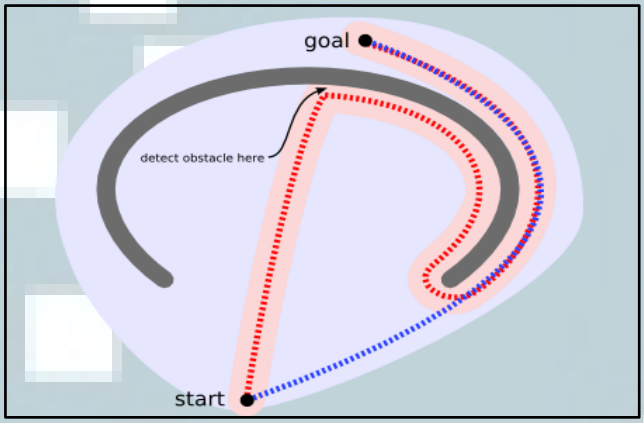
Prinsip dari algoritma ini adalah melakukan traversal satu per satu pada tiap simpul untuk memperoleh lintasan terpendek pada suatu *graph*. Algoritma A* akan menghitung jarak salah satu lintasan, lalu menyimpannya dan kemudian menghitung jarak lintasan lainnya. Ketika seluruh lintasan telah selesai dihitung, algoritma A* akan memilih lintasan yang paling pendek. Berikut adalah pseudocode dari algoritma *A**:

```
create the open list of nodes, initially containing only our starting node
   create the closed list of nodes, initially empty
   while (we have not reached our goal) {
       consider the best node in the open list (the node with the lowest f
value)
       if (this node is the goal) {
           then we're done
       }
       else {
           move the current node to the closed list and consider all of its
neighbors
           for (each neighbor) {
               if (this neighbor is in the closed list and our current g value
is lower) {
                   update the neighbor with the new, lower, g value
                   change the neighbor's parent to our current node
               }
               else if (this neighbor is in the open list and our current g
value is lower) {
                   update the neighbor with the new, lower, g value
                   change the neighbor's parent to our current node
               }
               else this neighbor is not in either the open or closed list {
                   add the neighbor to the open list and set its g value
               }
           }
       }
   }
```

Gambar 2.2 Contoh Pseudocode Algoritma *A** (*A-Star*)
(sumber: http://heyes-jones.com/pseudocode.php, diakses Juli 2014)

Pada gambar 2.3 di bawah ini, lokasi tujuan berada di atas dengan berjalan mulai dari titik bawah. Jalur yang dapat ditempuh (ditampilkan dalam warna merah muda), pertama kali algoritma ini menyusuri jalur tersebut (ditandai dengan titik-titik merah), kemudian menemukan rintangan seperti belokan dan

juga terjadi putar-balik *u-turn*. Lalu, algortima akan menentukan jalur tercepat (diperlihatkan dengan titik-titik biru), dengan melihat gambar besar dari jalur yang akan ditempuh dengan memindai seluruh area (ditandai dengan warna biru besar), sehingga jalur tidak melewati rintangan. *A** adalah pilihan yang paling populer untuk solusi menjari jalur, karena cukup fleksibel seperti, algoritma *Dijkstra*.



Gambar 2.3 Ilustrasi Solusi Algoritma *A** Dalam Mencari Jalur Terpendek (sumber: http://theory.stanford.edu/~amitp/GameProgramming/ AStarComparison.html, diakses Juli 2014)

2.4 Algoritma Dijkstra

Algortima *Dijkstra* ditemukan oleh Edsger W. Dijkstra yang merupakan salah satu varian bentuk algoritma popular dalam pemecahan persoalan yang terkait dengan masalah optimaisasi dan bersifat sederhna. Algortima ini menyelesaikan masalah mencari sebuah lintasan terpendek (sebuah lintasan memiliki panjang minimum) dari verteks a ke verteks z dalam graf berbobot, bobot tersebut meruapakan bilangan positif jadi tidak dapat dilalui oleh node negatif, namun jika terjadi demikian, maka penyelesaian yang diberikan adalah *infinity* (menurut Henny Syahriza Lubis, 2009), proses algoritma *Dijkstra* dalam mencari lintasan

terpendek dari suatu simpul ke semua pasangan simpul algoritma Dijkstra melalui sejumlah langkah yang menggunakan prinsip *greedy*.

(sumber: Hendro Ongkowijoyo, Hendra Saputra Libra, Andy Susanto, 2006, halaman 60) Cara kerja algortitma *Dijkstra* dengan menelusuri jalur dengan verteks awal S, ke semua verteks dalam graf (V-1). V adalah semua verteks yang terdapat dalam graf. Algortima *Dijkstra* membagi verteks-verteks yang pernah ditelusuri menjadi S dan F. S terdiri dari veteks-verteks yang jalur terpendeknya belum ditemukan. Sedangkan verteks yang tidak termasuk dalam S dan F adalah verteks yang belum ditelusuri.

Algoritma *Dijkstra* terus memperbarui D, yang berisi jarak terpendek yang terbaru dari S ke masing- masing verteks. Jika sebuah verteks v termasuk dalam S, maka d[V] sudah pasti merupakan jarak terpendek menuju ke verteks tersebut. Jika v termasuk F, maka untuk sejauh ini, d[V] masih merupakan jarak terpendek (masih bisa berubah). Selain itu, jika v tidak termasuk S maupun F, maka d[V] belum bernilai.

Dibawah ini adalah pseudocode dari algoritma *Dijkstra*. L(u, v) adalah panjang *edge* dari u ke v.

```
Procedur Dijkstra
    S = {S};
    F = OUT {S};
    For V in OUT {S}        {d[V] = length(S,V)};
    while F is not empty  {
        V = U such that d[U] is minimum among V in F;
        F = F - {V};
        S = S + {V};
        for W in OUT(V)  {
          if W is not in S    {
            new_dist = d[V] + L (V,W);
            if W is in F  { d[W] = min{d[W], new_dist} ; }
            else    {
              D[W] = new_dist;
              F = F + {W};
      }
    }
  }
}
;
```

Gambar 2.4 Contoh Pseudocode Algoritma *Dijkstra*
(sumber: http://library.binus.ac.id/eColls/eThesisdoc/Bab2/LBM2006-282 Bab%202.pdf, diakses Agustus 2014)

2.5 JavaScript

JavaScipt merupakan bahasa pemrograman yang terintegrasi dengan *web browser* untuk memberikan fleksibilitas tambahan bagi pengembang program untuk mengontrol elemen-elemen dalam halaman *web*, sering disebut dengan *client side programming*. *Client* di sini adalah *browser* pada pengguna komputer pribadi (*Personal Computer*, *PC*). Kode pemrograman *JavaScript* biasanya disisipkan diantara kode-kode pemrograman *HTML*. *JavaScript* juga termasuk dalam pemrograman berorientasi objek (*Object Oriented Programing*) yang memiliki *property*, *method*, dan *event handler* (Diar Puji Oktavian, 2010).

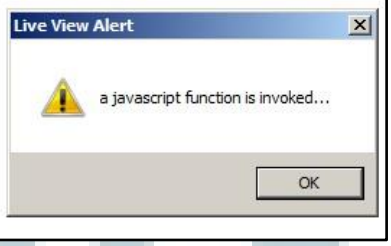
Property didefinisikan sebagai *setting* nilai suatu objek tertentu, contohnya adalah pengaturan warna suatu teks, *action* untuk suatu *form*, nama file untuk suatu gambar, dan masih banyak lagi.

Method adalah fungsi-fungsi yang dapat diterapkan dalam suatu objek, misalnya pengaturan untuk *window*.

Event handler menspesifikasikan bagaimana suatu objek merespon terhadap suatu kejadian, misalnya suatu *button* diklik, mengubah ukuran *window*, dan lain-lain. (Diar Puji Oktavian, 2010).

JavaScript bergantung kepada *web browser*, *Javascript* juga tidak memerlukan *Compiler* atau penerjemah khusus untuk menjalankannya, bersifat *Case Sensitive* untuk setiap perintah dan penamaan variabelnya, contoh variabel atau fungsi dengan nama TEST berbeda dengan variabel dengan nama test.

```
2   <html xmlns="http://www.w3.org/1999/xhtml">
3   <head>
4       <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
5
6       <title>JavaScript Function Example</title>
7
8       <script type="text/ecmascript">
9           function JSF_displaySomeInfo()
10          {
11              alert("a javascript function is invoked...");
12          }
13
14      </script>
15
16  </head>
17
18  <body>
19      <script type="text/ecmascript">
20          JSF_displaySomeInfo();
21      </script>
22
23  </body>
24  </html>
```



Gambar 2.5 Contoh Pemrograman *JavaScript*
(sumber: http://nana.hostei.com/source/materi/Script.pdf, diakses Juli 2014)