Implementasi Algoritma Dijkstra Dalam Menemukan Jarak Terdekat Dari Lokasi Pengguna Ke Tanaman Yang Di Tuju

Dokumen LaTeX ini dibuat oleh **Bostang Palaguna(13220055)**

Rafli F. Amanda* , Reynaldo A. A. Putra† , Muhammad Z. Fadhil‡ , Alifia Z. Ilmi§ , Astrid N. Hasanah¶
School of Electrical Engineering and Informatics
Institut Teknologi Bandung
Bandung, Indonesia
{*13219040, †13219071, ‡18319012, §18319013, ¶18319014}@std.stei.itb.ac.id

***Abstract*—Kebun Raya Purwodadi dengan luas area sekitar 85 hektar ternyata kekurangan papan informasi yang menyebabkan pengunjung kerap kali kebingungan dalam mencari lokasi tanaman tertentu. Paper ini bertujuan untuk membuat simulasi dari algoritma yang dapat menentukan jarak terdekat antara pengunjung (pengguna program) dengan lokasi tanaman yang dituju. Algoritma yang digunakan adalah algoritma Dijkstra yang beroperasi secara menyeluruh (*greedy*) untuk menguji seitap persimpangan (*Vertex*) dan jalan (*Edge*) pada Kebun Raya Purwodadi. Berdasarkan hasil simulasi dan pengujian, kompleksitas ruang dari program ini adalah $O(V)$ karena adanya pembentukan array yang berisi V nodes untuk mencari heap minimum. Sementara, kompleksitas waktu dari algoritma tersebut adalah $O(V^2)$.**

***Index Terms*—Dijkstra, Vertex, Edge, Tanaman.**

I. Introduction

Studi mengenai penggunaan algoritma Dijkstra dalam mencari jarak terdekat dapat diimplementasikan pada kasus pencarian tanaman pada Kebun Raya Purwodadi seperti yang telah dilakukan oleh Yusuf et al di tahun 2017 [1]. Paper ini bertujuan untuk melakukan simulasi kembali terhadap penelitian yang telah dilakukan dengan bahasa C serta mengevaluasi efisiensinya melalui perhitungan kompleksitas waktu dan ruang dengan analisis Big-O.

Di Kecamatan Purwodadi, Kabupaten Pasuruan, terdapat salah satu kebun raya di Indonesia yang bernama Kebun Raya Purwodadi yang memiliki luas area hingga 85 hektar. Kebun raya sebagai fasilitas rekreasi dan penelitian ini ternyata kekurangan papan informasi yang seharusnya disediakan oleh pihak pengelola. Hal ini menyebabkan banyaknya pengunjung yang merasa kebingungan untuk mencari lokasi dari tanaman tertentu. Oleh karena itu, Yusuf et al (2017) memutuskan untuk membuat suatu aplikasi dengan memanfaatkan algoritma Dijkstra untuk membantu pengunjung Kebun Raya Purwodadi dalam mencari lokasi tertentu.

Algoritma Dijkstra digunakan karena algoritma ini dapat beroperasi secara menyeluruh (algoritma greedy) terhadap semua alternatif fungsi serta durasi eksekusi yang lebih cepat jika dibandingkan dengan algoritma serupa, yaitu Bellman-Ford. Algoritma ini akan mencari jalur dengan 'biaya' atau cost terendah antara dua titik dengan membandingkan semua alternatif yang ada.

Pada kasus ini, masing-masing persimpangan di Kebun Raya Purwodadi direpresentasikan sebagai vertex dan setiap jalan direpresentasikan sebagai edge. Rute terdekat yang didapatkan akan diperoleh dari pembobotan setiap vertex dan edge berdasarkan jarak antara titik pengguna dengan titik tujuan atau tanaman.

II. Studi Pustaka

A. Algoritma Djikstra

Algorithm 1: Dijkstra's Algorithm `Dijkstra`

Result: Find the shortest path from a to z
procedure *Dijkstra*(G: weighted connected simple graph, with all weights positive)
{G has vertices $a = v_0, v_1, ..., v_n = z$ and lengths $w(v_i, v_j)$ where $w(v_i, v_j) = \infty$ if v_i, v_j is not an edge in G}
for $i := 1$ ***to*** n **do**
  $L(v_i) := \infty$
end
$L(a) := 0$
$S := \emptyset$
{the labels are now initialized so that the label of a is 0 and all other labels are ∞, and S is the empty set}
while $z \notin S$ **do**
  $u :=$ a vertex not in S with $L(u)$ minimal
  $S := S \cup \{u\}$
  for *all vertices v not in S* **do**
    if $L(u) + w(u, v) < L(v)$ **then**
      $L(v) := L(u) + w(u, v)$
      {this adds a vertex to S with minimal label and updates the labels of vertices not in S}
    end
  end
end
return $L(z) =$ *length of a shortest path from a to z*

Fig. 1. Pseudocode Algoritma Djikstra.

Algoritma Dijkstra adalah algoritma yang digunakan untuk menemukan jarak jalur terpendek antara dua *vertice* pada *graph* berbobot dan tidak berarah sederhana [2]. Berbobot berarti grafik memiliki *edge* dengan suatu 'bobot' atau harga. Bobot dapat merepresentasikan jarak, waktu, atau apapun yang memodelkan koneksi antara kedua *node*. Tidak berarah memiliki arti bahwa untuk setiap *node* yang terhubung, kita dapat mendekati suatu *node* dari kedua arah. Pendekatan Dijikstra juga memiliki asumsi bahwa bobot pada *edge* memiliki nilai yang tidak negatif. Hal ini karena nilai bobot akan terus dibandingkan dan diambil nilai yang paling kecil. Ada banyak varian pada algoritma ini, namun pada percobaan ini digunakan varian dimana suatu node ditetapkan menjadi *source node*. Dari *node* inilah akan dicari jarak terpendek diantara node lain. Algoritma ini dicetuskan oleh Edsger Wybe Dijkstra, salah seorang tokoh ternama di bidang computer science [3]. Kompleksitas dari algoritma dijkstra adalah $O(n^2)$, dengan n menyatakan jumlah *vertice* dari *graph* yang bersangkutan.

B. Kebun Raya Purwodadi

Kebun Raya Purwodadi adalah kebun penelitian di Kecamatan Purwodadi, Jawa Timur. Ia juga dikenal dengan nama Hortus Ilkim Kering Purwodadi dan didirikan tanggal 30 Januari 1941 oleh Dr. L.G.M. Baas Becking. Sebagai cabang dari Kebun Raya Bogor, ia memiliki fungsi mengkoleksi tumbuhan yang hidup di dataran rendah kering. Sebagai Balai Konservasi Tumbuhan di bawah Pusat Konservasi Tumbuhan Kebun Raya, Kedeputian Bidang Ilmu Pengetahuan Hayati LIPI, kebun raya ini memiliki banyak tumbuhan yang dinaunginya. Dengan menggunakan algoritma Dijkstra, diharapkan ia dapat membantu pengunjung mencari tanaman tertentu maupun jarak yang paling optimal.

III. Metode Penelitian

Peneliti menggunakan beberapa tahap dalam penyusunan paper ini. Pertama, dilakukan pengkajian dan studi literatur dengan membaca referensi paper yang berkaitan dan memilih paper yang dapat menjadi acuan dalam penelitian yang dilakukan, sehingga dari pilihan topik dan tema yang berkaitan secara luas dapat dikecilkan menjadi sebuah paper yang mencakup mayoritas dari topik yang dibahas. Setelah ditemukan beberapa paper, dilakukan perangkuman untuk menentukan paper yang sesuai sekaligus membahas poin-poin penting dari paper yang ingin dicapai. Setelah kedua tahap tersebut dilewati, penentuan paper yang dijadikan prototype penelitian merupakan hal yang mudah dan menjadi titik pencapaian dalam studi literatur dan pemilihan topik dari prototype penelitian yang dilakukan.

Setelah itu, tahap selanjutnya yang dilakukan oleh peneliti adalah pembuatan prototype berupa program yang ditulis dalam bahasa C. Pembuatan prototype berupa kode ini dilakukan terus-menerus dengan menggunakan metode trial and error sehingga perlu dilakukan revisi hingga protoype kode yang dibuat dapat mendapatkan output yang optimal dan sesuai dengan spesifikasi yang diharapakan. Tahap terakhir penelitian adalah pemaparan kode yang berhasil dijalankan tersebut ke dalam paper.

Fig. 2. Skema Metodologi Penelitian.

IV. Implementasi dan Pengujian

A. Implementasi Graph pada Array dalam Bahasa C

Program akan dimulai dengan pembacaan file bernama *listtanaman.txt*. File tersebut akan menyimpan informasi mengenai semua nama tanaman yang bersangkutan. Setelah pembacaan tersebut, akan dicari informasi mengenai bobot graph yang menghubungkan *node*. Informasi ini disimpan di dalam matriks segitiga bawah kiri didalam file *jarakantarpohon.txt* yang juga dibuka saat program dijalankan. Matriks menggambarkan bobot antara jarak dua *node* tanaman sekali saja karena pemodelan *undirected graph* yang memiliki jarak sama baik dari a ke b maupun b ke a. Nilai -1 akan menggambarkan bagian node yang tidak terhubung sama sekali dalam graph dan juga dinyatakan dalam suatu variabel bernama int_max (Jaraknya sebesar tak hingga). Nilai jarak terpendek akan disimpan dalam array tersebut selagi program berjalan.

B. Implementasi Algoritma Dijkstra dalam Bahasa C

alam implementasi algoritma, abstraksi dengan menggunakan pseudocode dapat dibagi menjadi dua buah fungsi dan satu program utama. Fungsi yang digunakan adalah fungsi printgraph (Fungsi Graph) untuk memunculkan graph berukuran $n \times n$ ke layar pengguna. Algoritma dari fungsi tersebut dapat dilihat pada bagian di bawah ini:

Fungsi kedua yang digunakan adalah fungsi pencari indeks pada array yang akan diproses dengan menggunakan pendekatan algoritma Dijkstra. Abstraksi fungsi yang digunakan dapat dilihat pada bagian berikut ini:

Program utama akan membaca file database tanaman beserta jarak masing-masing tanaman dan akan mencetak daftar tanaman yang berada di Kebun Raya Purwodadi. Kemudian, program akan menerima input salah satu tanaman terdekat dari

Algorithm 2: Fungsi Graph (`printgraph`)

Result: Memunculkan Graph $n \times n$ Ke Layar
procedure *printgraph(n, graph[n][n])*
while $i \leq n-1$ **do**
  $j \leftarrow 0$;
  while $j \leq n-1$ **do**
    if $graph[i][j] = int\_max$ **then**
      output (-1);
    else
      output $(graph[i][j])$;
    end
    $j \leftarrow j+1$;
  end
  $i \leftarrow i+1$;
end

Fig. 3. Pseudocode fungsi *printgraph*.

Algorithm 3: Fungsi Pencari Indeks `idx_process`

Result: Mencari indeks yang akan diproses dengan algoritma Dijkstra
Initialization:
$is\_found \leftarrow FALSE$;
$i \leftarrow 0$;
Algorithm:
while $i \leq n-1$ **do**
  $j \leftarrow 0$;
  if $!is\_final[i]$ ***and*** $!is\_found$ **then**
    $idx\_min \leftarrow i$;
    $val\_minimum \leftarrow jarak\_f[i]$;
    $is\_found \leftarrow true$;
  end
  if is_found ***and*** $!is\_final[i]$ ***and*** $(jarak\_f[i] < val\_minimum)$ **then**
    $idx\_min \leftarrow i$;
    $val\_minimum \leftarrow jarak\_f[i]$;
  end
end
if is_found **then**
  return (idx_min)
else
  return (int_max)
end

Fig. 4. Pseudocode fungsi *idxprocess*.

Algorithm 4: Program Utama Pencarian Rute Antara Dua Tanaman - Pembacaan Jumlah Tanaman

Result: Menyimpan nama tanaman pada sebuah array
Algorithm:
input(`namafile_tanaman`);
open(`namafile_tanaman`);
read(`namafile_tanaman`);
`n_tanaman` $\leftarrow 0$;
`baris` $\leftarrow 0$;
while `baris` $\leq$ `max_len` **do**
  `token` $\leftarrow$ `parse(baris)`;
  `token` $\leftarrow$ `nama_tanaman[n_tanaman]`;
  `n_tanaman` $\leftarrow$ `n_tanaman + 1`;
  `baris` $\leftarrow$ `baris+1`;
end

Fig. 5. Pseudocode pembacaan jumlah tanaman.

Algorithm 5: Program Utama Pencarian Rute Antara Dua Tanaman - Memasukkan Graph

Result: Menyimpan graph dalam sebuah matriks $n \times n$
input(`namafile_graph`);
open(`namafile_graph`);
read(`namafile_graph`);
$baris \leftarrow 0$;
while $baris \leq$ `max_len` **do**
  $k \leftarrow 0$;
  $token \leftarrow parse(baris)$;
  while $token! = NULL$ **do**
    `graph[j][k]` $\leftarrow token$;
    `graph[k][j]` $\leftarrow token$;
    if $token == -1$ **then**
      `graph[j][k]` $\leftarrow$ `int_max`;
      `graph[k][j]` $\leftarrow$ `int_max`;
    else
      $k \leftarrow k+1$;
      $token \leftarrow parse(NULL)$;
    end
  end
  $baris \leftarrow baris+1$;
end

Fig. 6. Pseudocode prosedur memasukkan graph.

pengguna sebagai penanda posisi awal pengguna. Setelah itu, program akan kembali menerima input posisi tanaman tujuan dan memproses pencarian rute terdekat dengan algoritma Dijkstra. Rute yang diperlukan akan ditampilkan dalam bentuk list nama tanaman yang harus dilalui pengguna dan menampilkan jarak antara kedua tanaman tersebut. Implementasi algoritma dalam abstraksi tersebut dapat dilihat pada gambar di bawah ini:

Setelah pembacaan jumlah tanaman dari file, maka diperlukan graph atau jarak antar tanaman yang akan menjadi dasar perhitungan dari pencarian rute terdekat. Proses memasukkan graph dapat dilihat pada algoritma berikut ini:

Setelah data yang dibutuhkan dimasukkan, implementasi dari algoritma Dijkstra untuk pencarian rute terdekat adalah sebagai berikut:

C. Implementasi Program dalam Bahasa C

Implementasi program dalam bahasa C dapat dilihat pada repository berikut.

D. Perhitungan Kompleksitas Waktu

Kompleksitas dari program ini dengan notasi kompleksitas Big O adalah $O(n^2)$. Hal tersebut disebabkan pada loop program bagian *for*, terdapat loop *for* lain yang berjumlah dua loop (Terletak pada bagian *assign* kondisi awal dan ketika program menjalankan algoritma Djikstra). Karena hal tersebut, akibatnya adalah kompleksitas waktu akan naik seiring dengan naiknya n program yang dijalankan, namun tidak bersifat linear sehingga kompleksitas waktunya adalah $O(n^2)$. Grafik kompleksitas waktu dapat direpresentasikan pada gambar berikut :

Algorithm 6: Program Utama Pencarian Rute Antara Dua Tanaman: Pencarian Jarak dengan Algoritma Djikstra

```
Algorithm:
input(idx_a);
idx_a ← idx_a-1;
input(idx_tujuan);
idx_tujuan ← idx_tujuan-1;
for i = 0 to n_tanaman do
    if i = idx_a then
        jarak_f[i] ← 0;
        is_final[i] ← FALSE;
    else
        jarak_f[i] ← int_max;
        is_final[i] ← FALSE;
    end
    for j = 0 to n_tanaman do
        list_dilalui[i][j] ← int_max;
    end
    idx_lalui[i] ← 0;
end
jarak_f[idx_a] ← 0;
list_dilalui[idx_a][0] ← idx_a;
idx_lalui[idx_a] ← idx_lalui[idx_a]+1;
idx_now ← idx_a;
while idx_now != int_max do
    is_final[idx_now] ← TRUE;
    for i = 0 to n_tanaman-1 do
        if (!is_final[i]) and
         graph[idx_now][i] != int_max and
         (jarak_f[idx_now] +
         graph[idx_now][i]) > jarak_f[i]
         then
            jarak_f[i] ←
             (jarak_f[idx_now] +
             graph[idx_now][i]);
            idx_lalui[i] ←
             idx_lalui[idx_now]+1;
        end
        for j=0 to idx_dilalui[i]-1 do
            if j=idx_dilalui[i]-1 then
                list_dilalui[i][j] ← i;
            else
                list_dilalui[i][j] ←
                 list_dilalui[idx_now][j];
            end
        end
    end
    idx_now ← idx_process(n_tanaman,
     jarak_f, is_final);
end
```

Fig. 7. Pseudocode pencarian jarak dengan algoritma Djikstra.

E. Perhitungan Kompleksitas Tempat

Matriks penyimpanan yang digunakan pada program ini memiliki ukuran terbesar $n \times n$, dengan nilai n merepresentasikan banyak tanaman dalam file *listtanaman.txt*. Program akan melalui grafik dan menyimpan nilai bobot antara *node* sebesar matriks di atas, mengakibatkan program dengan kompleksitas $O(n^2)$. Hal ini dapat dilihat pada grafik kompleksitas tempat pada gambar berikut:

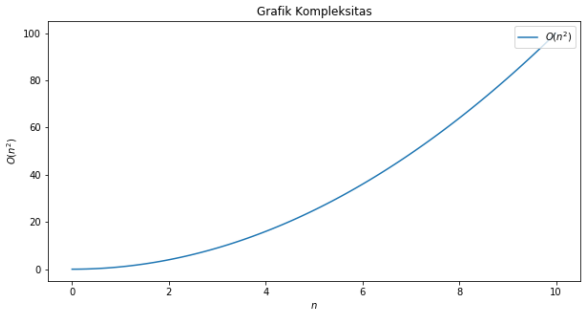

Fig. 8. Grafik kompleksitas waktu algoritma Djikstra.

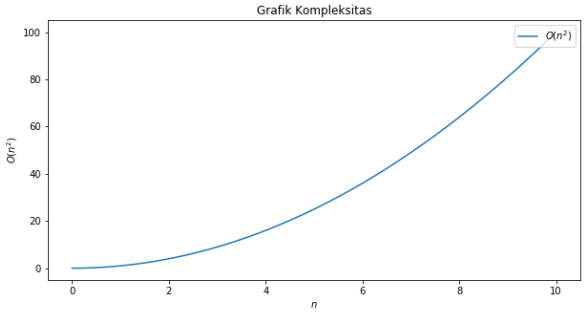

Fig. 9. Grafik kompleksitas ruang algoritma Djikstra.

V. KESIMPULAN

Pada perhitungan Jarak Terdekat dalam suatu lokasi atau ruang dapat diimplementasikan penggunaan Algoritma Djikstra dalam perhitungannya untuk mencapai suatu target pada ruang tersebut dari suatu titik. Terbukti dari penelitian Kebun Raya Purwodadi untuk menentukan Tanaman yang ingin dituju.

REFERENCES

REFERENCES

[1] M. S. Yusuf, H. M. Az-Zahra, and D. H. Apriyanti, "Implementasi algoritma dijkstra dalam menemukan jarak terdekat dari lokasi pengguna ke tanaman yang di tuju berbasis android (studi kasus di kebun raya purwodadi)," *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN*, vol. 2548, p. 964X, 2017.

[2] T. Koshy, *Discrete mathematics with applications*. Elsevier, 2004.

[3] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.