Penerapan Algoritma Breadth-First Search dan Exhaustive Search dalam Menentukan Lokasi Titik Kumpul dalam Ruang Publik

Muhammad Naufal Fakhrizal 13518115
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): mnaufalfakhrizal@gmail.com

***Abstrak*—Makalah ini membahas salah satu alternatif solusi untuk menentukan lokasi yang cocok untuk titik kumpul dalam suatu ruang dengan pendekatan graf dan dengan menggunakan algoritma *depth-first search* dan *exhaustive search*. Suatu peta atau denah dari ruang dapat direpresentasikan sebagai graf berbobot dengan simpul merepresentasikan fasilitas dan sisi merepresentasikan akses. Heuristik yang digunakan yaitu suatu simpul dalam graf berbobot yang merepresentasikan ruang sehingga rata-rata bobot perjalanan dari setiap simpul ke simpul tersebut minimal adalah kandidat yang cocok untuk menempatkan titik kumpul di ruang tersebut.**

Kata Kunci—algoritma; breadth-first search; exhaustive search; graf; keselamatan; titik kumpul

I. PENDAHULUAN

Dalam perencanaan tata ruang, baik tata ruang dalam bangunan, kompleks, wilayah, atau kota, beberapa aspek perlu diperhatikan. Salah satu dari aspek tersebut adalah keselamatan. Suatu ruang, terutama ruang publik, harus memiliki standar keamanan dan keselamatan yang memadai, untuk menjamin keselamatan pengguna ruang tersebut, terutama dalam keadaan darurat yang dapat mengancam nyawa.

Dalam kondisi darurat, diperlukan evakuasi orang yang berada di dalam fasilitas tersebut seefisien mungkin. Dalam banyak ruang publik, disediakan fasilitas berupa ruang yang cukup terbuka dan luas yang dapat berfungsi untuk mengumpulkan orang-orang dalam suatu titik saat keadaan darurat terjadi, seperti kebakaran atau gempa bumi. Ruang ini disebut titik kumpul atau *assembly point*.

Untuk menjamin keselamatan pengguna ruang tersebut, suatu titik kumpul harus memiliki beberapa kriteria, salah satunya adalah aksesibilitas dari semua tempat yang ada di ruang tersebut. Oleh karena itu, titik kumpul harus diletakkan sedemikian rupa sehingga semua pengguna ruang dapat melakukan evakuasi ke titik kumpul tersebut dari semua titik di ruang tersebut dengan mudah. Untuk menentukan lokasi yang cocok untuk titik kumpul tersebut, penulis menggunakan pendekatan graf dan algoritma *breadth-first search* dan *exhaustive search.*

II. DASAR TEORI

A. Titik Kumpul

Titik kumpul (*assembly point*) adalah sebuah area luas sehingga orang dengan jumlah besar dapat berkumpul di area tersebut. Dalam bidang keselamatan, titik kumpul merupakan fasilitas penting untuk disediakan pada perkantoran, kampus, ruang publik, dan sebagainya. Kegunaan dari titik kumpul adalah untuk mengarahkan orang-orang menuju satu tempat yang aman saat evakuasi berlangsung dalam keadaan darurat, serta untuk mencegah penyebaran orang-orang di segala tempat agar memudahkan tim penyelamat untuk penyelamat dalam menangani korban atau mengarahkan orang-orang ke tahap evakuasi selanjutnya.

Dalam Lampiran Peraturan Menteri Kesehatan Republik Indonesia Nomor 48 Tahun 2016 Tentang Standar Keselamatan dan Kesehatan Kerja Perkantoran, Tindakan awal dalam rencana tanggap darurat bencana meliputi merencanakan suatu titik kumpul yang merupakan suatu denah evakuasi yang menunjukkan kemana pekerja berkumpul bila terjadi kondisi darurat dan diperintahkan untuk evakuasi serta menyiapkan rambu-rambu arah yang menunjuk ke arah titik kumpul.

Sebuah titik kumpul ditandai dengan rambu yang mengandung simbol titik kumpul. Sesuai standar ISO 7010 - E007, Simbol titik kumpul adalah dua figur orang yang berdampingan, orang ketiga diperlihatkan kepalanya di belakang tengah kedua orang pertama, empat tanda panah yang mengarah ke ketiga orang dari setiap sudut, dan latar hijau.

Gambar 1. Simbol Penanda Titik Kumpul atau *Assembly Point*

Sumber: iso.org

Terdapat beberapa hal yang perlu diperhatikan dalam memilih tempat untuk dijadikan titik kumpul. Sebuah titik kumpul harus cukup besar untuk menampung semua orang yang ada dalam wilayah yang terkena keadaan darurat (Sebagai contoh, titik kumpul di area perkantoran harus bisa menampung seluruh staf kantor tersebut). Titik kumpul harus dapat diakses dengan mudah dari semua titik yang ada di ruang atau wilayah tersebut, terutama bagi orang yang memiliki kekurangan dalam mobilitas. Sebaiknya jumlah titik kumpul lebih dari satu, terutama jika ruang atau wilayah cukup luas. Selain itu, Titik kumpul harus jauh dari sumber bahaya, seperti bangunan dan objek tinggi, sumber listrik, lalu lintas, dan lain-lain.

B. Algoritma Exhaustive Search

Algoritma *brute force* adalah algoritma sederhana yang memandang persoalan algoritmik sesuai dengan deskripsi persoalan tersebut. Algoritma *exhaustive search* merupakan subset dari algoritma *brute force* yang menyelesaikan persoalan pencarian dan optimasi. Sebuah algoritma *exhaustive search* akan melakukan enumerasi semua kandidat solusi dari persoalan, dan dalam setiap enumerasi, memeriksa apakah kandidat tersebut benar-benar merupakan solusi dari persoalan tersebut (untuk persoalan pencarian, apakah kandidat memenuhi *constraint*, dan untuk persoalan optimasi, apakah kandidat memenuhi *constraint* dan memiliki nilai evaluasi maksimum atau minimum).

Karakteristik dari algoritma *exhaustive search* adalah sederhana dan mudah diimplementasi, sehingga algoritma pertama yang muncul untuk menyelesaikan suatu persoalan pada umumnya adalah algoritma *exhaustive search.*

Akan tetapi, algoritma *exhaustive search* memiliki kekurangan. yaitu masalah efisiensi. Untuk mendapatkan solusi, algoritma *exhaustive search* harus melakukan enumerasi untuk setiap anggota dari himpunan kandidat solusi dari persoalan, dan ukuran himpunan kandidat solusi berkontribusi terhadap kompleksitas algoritma *exhaustive search.* Sebagai contoh, untuk persoalan mencari pasangan bilangan dengan jumlah terkecil dari sebuah himpunan bilangan berukuran n, maka ukuran himpunan kandidat solusi adalah $(n - 1)n / 2$ dan kompleksitas algoritma *exhaustive search* untuk persoalan ini adalah $O(n^2)$. Jika nilai n sangat besar, maka ukuran himpunan kandidat solusi akan jauh lebih besar, terutama jika ukuran himpunan kandidat solusi meningkat terhadap n secara non-polinomial (Contohnya pada *Travelling Salesperson Problem* dengan ukuran graf n, ukuran himpunan kandidat solusi adalah $n!$, dengan anggotanya adalah semua kemungkinan permutasi dari semua simpul di graf).

Oleh karena itu, algoritma *exhaustive search* digunakan untuk persoalan dengan ukuran input yang kecil, atau untuk persoalan yang belum ditemukan algoritma yang lebih baik.

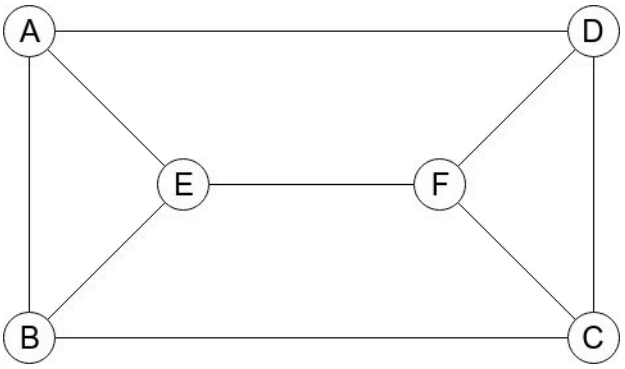
C. Algoritma Breadth-First Search

Algoritma *breadth-first search* adalah salah satu algoritma traversal graf yang mengunjungi semua simpul pada graf dengan urutan tertentu. Secara garis besar, algoritma *breadth-first search* adalah sebagai berikut.

a. Mulai dari sebuah simpul *A*. Tandai simpul *A* sudah dikunjungi.
b. Kunjungi semua simpul yang bertetangga dengan simpul *A*.
c. Kunjungi semua simpul yang belum dikunjungi dan bertetangga dengan semua simpul yang sudah dikunjungi sebelumnya. Ulangi langkah ini sampai semua simpul dikunjungi.

Secara umum, implementasi algoritma *breadth-first stack* menggunakan antrian atau *queue* yang menyimpan simpul yang akan atau telah dikunjungi. Saat sebuah simpul dikunjungi, semua simpul tetangga yang belum dikunjungi dimasukkan ke belakang *queue* tersebut. Setiap iterasi, algoritma mengambil simpul terdepan pada *queue* dan mengunjungi simpul tersebut.

Sebagai contoh, pada graf pada Gambar 2, algoritma *breadth-first search* dengan simpul awal A akan berjalan sebagai berikut.

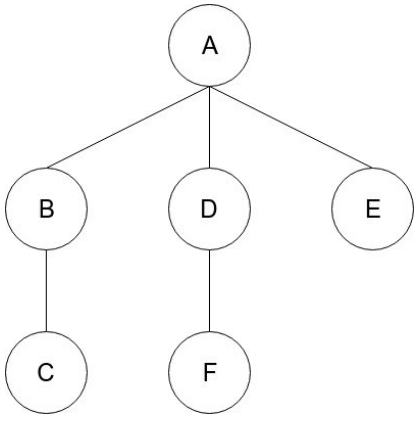


Gambar 2. Graf Contoh

Tabel 1. Hasil Algoritma *Breadth-First Search* pada Graf di Gambar 2 dengan Simpul Awal *A*.

Iter	Simpul yang Dikunjungi (Ekspan)	*Queue* (Simpul Hidup)	Simpul yang Sudah Dikunjungi					
			A	*B*	*C*	*D*	*E*	*F*
0		*A*	*T*	*F*	*F*	*F*	*F*	*F*
1	*A*	*B, D, E*	*T*	*T*	*F*	*T*	*T*	*F*
2	*B*	*D, E, C*	*T*	*T*	*T*	*T*	*T*	*F*
3	*D*	*E, C, F*	*T*	*T*	*T*	*T*	*T*	*T*
4	*E*	*C, F*	*T*	*T*	*T*	*T*	*T*	*T*
5	*C*	*F*	*T*	*T*	*T*	*T*	*T*	*T*
6	*F*	-	*T*	*T*	*T*	*T*	*T*	*T*
Urutan Kunjungan Simpul		*A > B > D > E > C > F*						

Selain itu, algoritma ini juga menghasilkan sebuah pohon status yang menyatakan tahap-tahap pengunjungan simpul. Kunjungan dilakukan dengan mengunjungi simpul status dengan level paling atas, kemudian mengunjungi simpul status pada level di bawahnya, dan seterusnya.



Gambar 3. Pohon Status untuk Tabel 1

Jika sebuah pohon status memiliki maksimal *d* tingkat dan setiap simpul status memiliki paling banyak *b* cabang, maka kompleksitas waktu algoritma *breadth-first search* adalah $O(b^d)$ dan kompleksitas ruangnya adalah $O(b^d)$. Untuk nilai *b* yang terbatas, algoritma bersifat *complete* (pasti menemukan solusi atau simpul tujuan).

D. Algoritma Depth-First Search

Algoritma traversal graf yang lain adalah *depth-first search.* Berbeda dengan *breadth-first search,* algoritma *depth-first search* mengunjungi satu simpul tetangga *B* dari simpul *A* yang sebelumnya dikunjungi, kemudian mengunjungi simpul tetangga dari simpul *B,* dan seterusnya, sampai tidak ditemukan lagi simpul tetangga yang belum dikunjungi. Jika tidak ditemukan lagi simpul tetangga yang belum dikunjungi dari suatu simpul, algoritma melakukan runut balik (backtracking) sesuai jalur kunjungan sebelumnya sampai simpul yang masih memiliki simpul tetangga yang belum dikunjungi, dan kemudian algoritma mengunjungi simpul tetangga tersebut. Jika runut balik kembali ke simpul awal, maka algoritma selesai dan semua simpul sudah dikunjungi dengan asumsi graf terhubung.

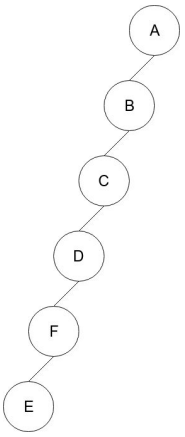
Secara umum, implementasi algoritma *breadth-first stack* menggunakan *stack* yang menyimpan simpul yang akan dikunjungi. Saat sebuah simpul dikunjungi, semua simpul tetangga yang belum dikunjungi di-*push* ke dalam *stack* tersebut. Pada setiap iterasi, algoritma mengambil simpul teratas pada *stack* dan mengunjungi simpul tersebut.

Sebagai contoh, pada graf di Gambar 2, algoritma *depth-first search* dengan simpul awal A akan berjalan sebagai berikut.

Tabel 2. Hasil Algoritma *Depth-First Search* pada Graf di Gambar 2 dengan Simpul Awal *A*.

Iter	Simpul yang Dikunjungi (Ekspan)	*Stack* (Simpul Hidup)	Simpul yang Sudah Dikunjungi					
			A	*B*	*C*	*D*	*E*	*F*
0		*A*	*F*	*F*	*F*	*F*	*F*	*F*
1	*A*	*B, D, E*	*T*	*F*	*F*	*F*	*F*	*F*
2	*B*	*C, E, D, E*	*T*	*T*	*F*	*F*	*F*	*F*
3	*C*	*D, F, E, D, E*	*T*	*T*	*T*	*F*	*F*	*F*
4	*D*	*F, F, E, D, E*	*T*	*T*	*T*	*T*	*F*	*F*
5	*F*	*E, F, E, D, E*	*T*	*T*	*T*	*T*	*F*	*T*
6	*E*	*F, E, D, E*	*T*	*T*	*T*	*T*	*T*	*T*
Urutan Kunjungan Simpul		*A > B > C > D > F > E*						

Algoritma menghasilkan pohon status sebagai berikut. Pada pohon tersebut, urutan pembangkitan simpul adalah dengan membangkitkan satu cabang dari simpul status, dan dari cabang tersebut dibangkitkan simpul cabang yang lain, dan seterusnya hingga mencapai simpul status daun, lalu dilakukan runut balik hingga mencapai simpul status yang masih bisa dibankitkan cabang dari simpul tersebut.



Gambar 4. Pohon Status untuk Tabel 2

Jika sebuah pohon status memiliki simpul daun dengan kedalaman terdalam *m* tingkat/level dan setiap simpul status memiliki paling banyak *b* cabang, maka kompleksitas waktu algoritma *breadth-first search* adalah $O(b^m)$ dan kompleksitas ruangnya adalah $O(bm)$. Algoritma *depth-first search* memiliki kompleksitas waktu yang lebih buruk dan kompleksitas ruang yang lebih baik dibandingkan dengan algoritma *breadth-first search.* Algoritma *depth-first search* bersifat *complete* jika nilai *m* terbatas.

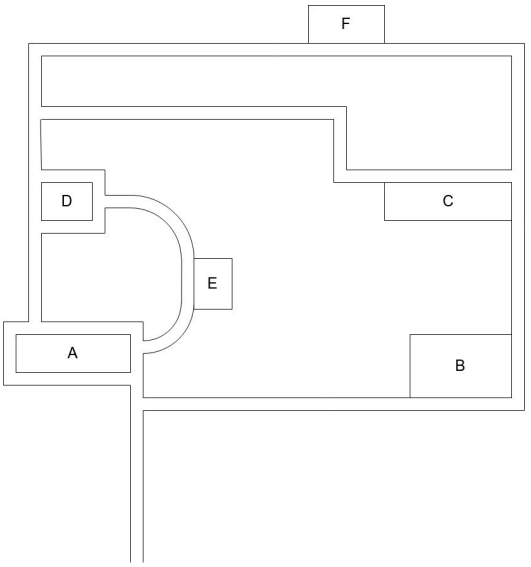
III. Penggunaan Algoritma dalam Menentukan Lokasi Titik Kumpul

E. Representasi Denah Ruang dalam Graf

Sebuah denah atau peta dari suatu ruang atau wilayah dapat direpresentasikan sebagai graf berbobot sebagai berikut:

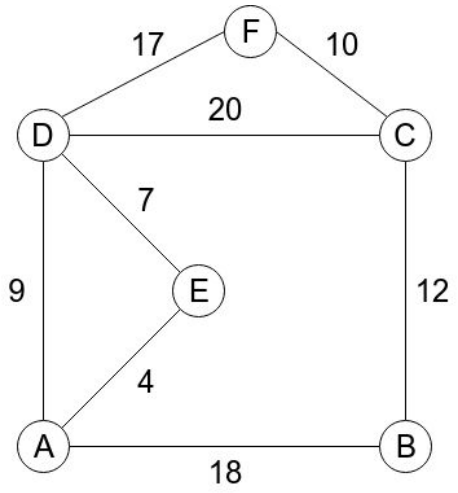
- Simpul (*node*) merepresentasikan suatu fasilitas dalam ruang tersebut. Simpul dapat berupa ruangan, bangunan, taman, lapangan, dan lain-lain.
- Sisi (*edge*) merepresentasikan adanya akses dari suatu fasilitas ke fasilitas lain. Sisi dapat merepresentasikan jalan, pintu, atau ketersediaan moda transportasi dari satu fasilitas ke fasilitas lain.
- Bobot dari sisi merepresentasikan kemudahan akses dari kedua fasilitas yang dihubungkan oleh sisi tersebut dengan bobot semakin kecil menandakan semakin mudah akses.. Bobot dapat merepresentasikan jarak atau waktu tempuh dari satu fasilitas ke fasilitas lain.

Sebagai contoh, berikut adalah peta dari sebuah kampus yang memiliki enam bangunan.



Gambar 5. Peta Kampus

Peta di atas dapat direpresentasikan dalam graf berbobot, dengan simpul merepresentasikan bangunan, sisi merepresentasikan jalan, dan bobot merepresentasikan jarak tempuh dalam satuan 100 meter.



Gambar 6. Representasi Peta pada Gambar . dalam Graf Berbobot

Heuristik yang akan digunakan adalah kandidat simpul terbaik untuk dijadikan titik kumpul adalah simpul yang rata-rata bobot tempuh dari semua simpul ke simpul kandidat memiliki nilai terkecil. Heuristik ini digunakan dengan asumsi bahwa hanya ada satu titik kumpul pada ruang tersebut, dan semua kriteria titik kumpul yang lain sudah dipenuhi, seperti jauh dari bangunan tinggi, area cukup luas, dan sebagainya.

F. Penggunaan Algoritma Breadth-First Search

Untuk mendapatkan rata-rata bobot tempuh dari semua simpul ke simpul kandidat, perlu didapatkan semua bobot tempuh untuk perjalanan dari setiap simpul ke simpul kandidat. Oleh karena itu, perlu dilakukan traversal di keseluruhan graf. Penulis menggunakan algoritma

breadth-first search dibanding algoritma depth-first search karena algoritma breadth-first search dapat melakukan pemeriksaan ke seluruh simpul dengan kedalaman pohon status minimal. Untuk keperluan penyelesaian masalah ini, algoritma breadth-first search adalah sebagai berikut.

a. Semua simpul pada graf memiliki sebuah nilai *f* yang menyatakan bobot minimal untuk mencapai simpul kandidat dari simpul ini yang sudah ditemukan. Sebelum algoritma dijalankan, untuk setiap simpul *S,* maka *f(S)* = ∞.
b. Untuk simpul asal *R*, yaitu simpul yang dijadikan kandidat sebagai lokasi titik kumpul, *f(R)* = 0.
c. Untuk semua sisi *E* yang keluar dari *R,* masukkan *E* ke dalam *queue* simpul status hidup.
d. Selama masih ada simpul status hidup, lakukan hal berikut:
    i. Ambil simpul status *Q* yang paling pertama masuk ke dalam *queue.*
    ii. simpul status *Q* merepresentasikan sisi *E* pada graf yang keluar dari simpul *A* menuju simpul *B*.
    iii. Jika pada *f(B) > f(A) + w(E),* maka nilai *f(B)* menjadi *f(A) + w(E)* dengan *w(E)* adalah bobot dari sisi *E,* dan semua sisi yang keluar dari simpul *B* dan tidak menuju simpul *A* masuk ke dalam *queue* simpul status hidup.
e. Untuk semua simpul *S* pada graf, *f(S)* adalah bobot minimum untuk menempuh perjalanan dari simpul *S* menuju simpul *R*.

G. Penggunaan Algoritma exhaustive search

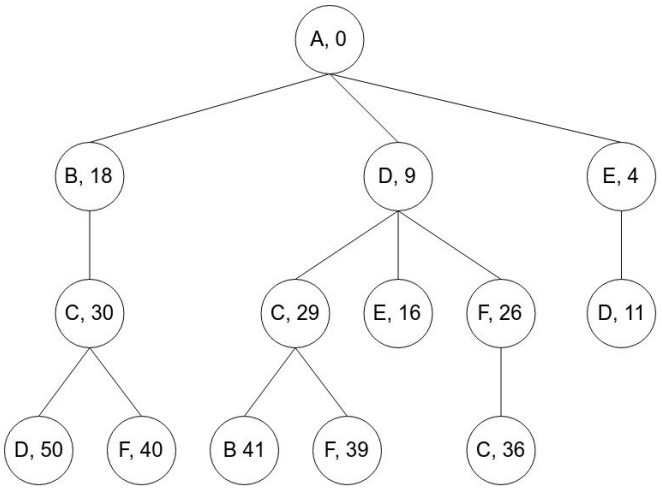
Algoritma *exhaustive search* digunakan untuk mencari simpul yang menghasilkan rata-rata bobot tempuh dari setiap simpul ke simpul tersebut. Hal ini dilakukan dengan melakukan algoritma *depth-first search* di atas terhadap setiap simpul yang ada pada graf sebagai simpul kandidat, kemudian mencari simpul mana yang menghasilkan nilai minimal.

H. Contoh Penggunaan Algoritma

Lihat gambar . Dengan menerapkan algoritma *depth-first search* di atas untuk simpul A sebagai simpul kandidat, didapatkan tabel iterasi sebagai berikut. Untuk setiap iterasi dengan simpul ekspan S_{Qi} , jika *f(S)* > i, maka *f(S)* = i dan sisi yang bersisian dengan simpul ekspan S_{Qi} dimasukkan ke dalam simpul hidup.

Tabel 3. Iterasi Algoritma untuk Simpul Kandidat *A*

Iter	Ekspan	Simpul Hidup	*f(S)*					
			A	B	C	D	E	F
1	A_0	B_{A18}, D_{A9}, E_{A4}	0	∞	∞	∞	∞	∞
2	B_{A18}	D_{A9}, E_{A4}, C_{B30}	0	18	∞	∞	∞	∞
3	D_{A9}	$E_{A4}, C_{B30}, C_{D29}, E_{D16}, F_{D26}$	0	18	∞	9	∞	∞
4	E_{A4}	$C_{B30}, C_{D29}, E_{D16}, F_{D26}, D_{E11}$	0	18	∞	9	4	∞
5	C_{B30}	$C_{D29}, E_{D16}, F_{D26}, D_{E11}, D_{C50}, F_{C40}$	0	18	30	9	4	∞
6	C_{D29}	$E_{D16}, F_{D26}, D_{E11}, D_{C50}, F_{C40}, B_{C41}, F_{C39}$	0	18	29	9	4	∞
7	E_{D16}	$F_{D26}, D_{E11}, D_{C50}, F_{C40}, B_{C41}, F_{C39}$	0	18	29	9	4	∞
8	F_{D26}	$D_{E11}, D_{C50}, F_{C40}, B_{C41}, F_{C39}, C_{F36}$	0	18	29	9	4	26
9	D_{E11}	$D_{C50}, F_{C40}, B_{C41}, F_{C39}, C_{F36}$	0	18	29	9	4	26
10	D_{C50}	$F_{C40}, B_{C41}, F_{C39}, C_{F36}$	0	18	29	9	4	26
11	F_{C40}	$B_{C41}, F_{C39}, C_{F36}$	0	18	29	9	4	26
12	B_{C41}	F_{C39}, C_{F36}	0	18	29	9	4	26
13	F_{C39}	C_{F36}	0	18	29	9	4	26
14	C_{F36}	-	0	18	29	9	4	26



Gambar 7. Pohon Status yang Dibangkitkan Algoritma untuk Simpul Kandidat *A*

Didapatkan bahwa total bobot perjalanan dari setiap simpul ke simpul *A* adalah 0 + 18 + 29 + 9 + 4 + 26 = 86.

Melakukan algoritma yang sama pada simpul *B, C, D, E,* dan *F* menghasilkan nilai sebagai berikut.

Tabel 4. Bobot tempuh dari setiap simpul ke simpul kandidat

Kandidat	Bobot Tempuh Dari						
	A	B	C	D	E	F	Total
A	0	18	29	9	4	26	86
B	18	0	12	27	22	22	101
C	29	12	0	20	27	10	98
D	9	27	20	0	7	17	80
E	4	22	27	7	0	24	84
F	26	22	10	17	24	0	99
Terpilih			D	Minimum			80

Didapatkan bahwa jumlah bobot tempuh perjalanan dari setiap simpul ke simpul kandidat terkecil saat simpul kandidat adalah simpul *D,* sehingga bangunan *D* dapat dijadikan titik kumpul pada saat terjadi keadaan darurat.

IV. Implementasi dalam Program

Untuk memudahkan dalam menjalankan algoritma, terutama untuk graf berukuran besar, sebuah program sederhana dibuat.

Dalam program ini, graf merupakan suatu struktur yang menampung senarai berisi simpul dan matriks ketetanggaan (*adjacency matrix*) dari setiap pasang simpul, dengan elemen matriks berupa bobot sisi yang menghubungkan kedua simpul berupa bilangan bulat positif atau tak hingga jika tidak ada sisi yang menghubungkan kedua simpul tersebut.

Sebuah simpul menyimpan informasi mengenai bobot perjalanan dari simpul tersebut ke simpul kandidat solusi. Pada awalnya, nilai yang disimpan adalah tak hingga, dan diubah selama berjalannya algoritma *breadth-first search.* Perubahan nilai hanya akan berhasil jika nilai baru lebih kecil dari nilai lama. Hal ini karena bobot perjalanan yang diinginkan adalah bobot perjalanan minimal.

```
class Node:

    def __init__(self):
        self.dist = INT_MAX

    def reset_dist(self):
        self.dist = INT_MAX

    def get_dist(self):
        return self.dist

    def set_dist(self, new_dist):
        if self.dist > new_dist:
            self.dist = new_dist
            return True
        return False
```

Gambar 8. Implementasi simpul dalam Python

Berikut adalah implementasi algoritma *breadth-first search* dalam Python.

```
def run_bfs(self, start_idx):
    queue = []
    for node in self.nodes:
        node.reset_dist()
    queue.append((start_idx, start_idx, 0))
    while len(queue) > 0:
        expanded_idx, from_idx, cur_dist = queue.pop(0)
        if self.nodes[expanded_idx].set_dist(cur_dist):
            for i in range(len(self.nodes)):
                new_dist = self.dists[expanded_idx][i]
                if new_dist < INT_MAX and i != from_idx:
                    queue.append((i, expanded_idx, cur_dist + new_dist))
```

Gambar 9. Implementasi algoritma *breadth-first search* dalam Python

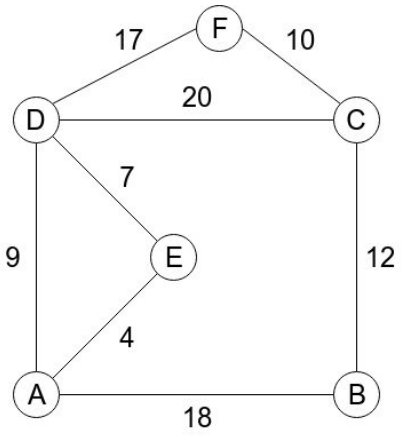
Untuk menentukan simpul kandidat dengan jumlah bobot minimum, dilakukan *exhaustive search* dengan mengenumerasi setiap simpul pada graf, dan untuk setiap simpul, dijalankan algoritma *breadth-first search* pada graf dengan simpul tersebut sebagai simpul kandidat. Jika ada dua atau lebih simpul kandidat yang memiliki jumlah bobot sama dan terkecil, algoritma akan menghasilkan himpunan solusi yang mengandung semua simpul tersebut.

```
def find_assembly_point_idx(self):
    min_idxs = list()
    min_dist = INT_MAX
    totals = list()
    dists = list()

    for i in range(len(self.nodes)):
        self.run_bfs(i)
        total_dist, dist_list = self.get_total_dist()
        totals.append(total_dist)
        dists.append(dist_list)
        if min_dist >= total_dist:
            if min_dist > total_dist:
                min_idxs = list()
                min_dist = total_dist
            min_idxs.append(i)
    return Result(dists, totals, min_dist, min_idxs)
```

Gambar 10. Implementasi Algoritma *Exhaustive Search* untuk Mencari Simpul Kandidat dengan Jumlah Bobot Terkecil Dengan Informasi Tambahan

I. Skenario Satu Simpul dengan Bobot Terkecil



Gambar 11. Graf Untuk Skenario Satu Simpul dengan Bobot Terkecil

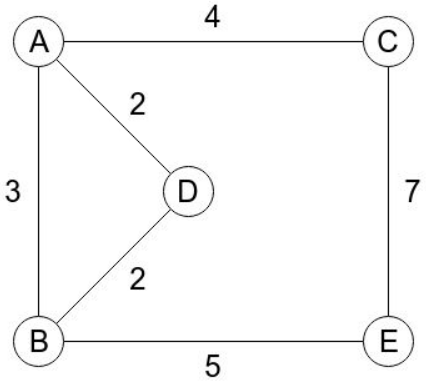
```
Adjacency Matrix:
   |   A   B   C   D   E   F
---+----------------------------
 A |   -  18   -   9   4   -
 B |  18   -  12   -   -   -
 C |   -  12   -  20   -  10
 D |   9   -  20   -   7  17
 E |   4   -   -   7   -   -
 F |   -   -  10  17   -   -

Result:
 From/To |   A   B   C   D   E   F Total
---------+--------------------------------
       A |   0  18  29   9   4  26    86
       B |  18   0  12  27  22  22   101
       C |  29  12   0  20  27  10    98
       D |   9  27  20   0   7  17    80
       E |   4  22  27   7   0  24    84
       F |  26  22  10  17  24   0    99
Suitable nodes for assembly point:
D
```

Gambar 12. Hasil Eksekusi Program untuk Skenario Satu Simpul dengan Bobot Terkecil

Menggunakan graf yang sama dengan sebelumnya, program mengeluarkan hasil yang sama dengan hasil dari bagian sebelumnya, yaitu titik kumpul cocok diletakkan pada simpul *D*, dengan jumlah bobot perjalanan dari setiap simpul terkecil.

J. Skenario Banyak Simpul dengan Bobot Terkecil



Gambar 13. Graf Untuk Skenario Banyak Simpul dengan Bobot Terkecil

```
Adjacency Matrix:
   |   A   B   C   D   E
---+----------------------
 A |   -   3   4   2   -
 B |   3   -   -   2   5
 C |   4   -   -   -   6
 D |   2   2   -   -   -
 E |   -   5   6   -   -

Result:
 From/To |   A   B   C   D   E Total
---------+----------------------------
       A |   0   3   4   2   8    17
       B |   3   0   7   2   5    17
       C |   4   7   0   6   6    23
       D |   2   2   6   0   7    17
       E |   8   5   6   7   0    26
Suitable nodes for assembly point:
A B D
```

Gambar ?. Hasil Eksekusi Program untuk Skenario Banyak Simpul dengan Bobot Terkecil

Pada graf ini, terdapat tiga kandidat lokasi titik kumpul yang sama kuatnya, yaitu simpul *A, B,* dan *D.*

V. Analisis

Dari kedua skenario di atas, terdapat kemiripan di antara keduanya, yaitu kandidat lokasi yang terpilih cenderung memiliki jumlah simpul tetangga terbanyak. Pada skenario pertama, kandidat yang terpilih adalah simpul *D*, dengan empat simpul tetangga, dan pada skenario kedua, dua dari tiga kandidat terpilih memiliki jumlah simpul tetangga terbanyak, yaitu simpul A dan B, dengan tiga simpul tetangga. Hal ini sesuai dengan kriteria titik kumpul yang baik, yaitu kemudahan akses dari semua titik menuju titik kumpul tersebut. Dalam keadaan ideal, dalam representasi graf, simpul yang menjadi kandidat terpilih untuk menjadi titik kumpul bertetangga dengan semua simpul lainnya. Dengan kata lain, semua tempat yang ada dalam ruang atau wilayah yang

direpresentasikan memiliki akses langsung menuju titik kumpul tersebut.

VI. Kesimpulan

Algoritma *breadth-first search* dan *exhaustive search* dapat digunakan untuk menyelesaikan persoalan menentukan lokasi yang cocok dalam suatu ruang publik untuk dijadikan titik kumpul dengan pendekatan graf dan beberapa asumsi. Berdasarkan hasil analisis, sebuah titik kumpul sebaiknya memiliki akses langsung dan mudah dari sebanyak mungkin tempat.

Link Video Youtube

https://youtu.be/fYk1d4i6cGI

Ucapan Terima Kasih

Penulis memanjatkan puji dan syukur ke hadirat Tuhan Yang Maha Esa atas terselesaikannya makalah ini. Selain itu, penulis juga mengucapkan terima kasih kepada seluruh dosen dan tim pengajar mata kuliah IF2211 Strategi Algoritma Tahun Akademik 2019/2020 atas kerja keras dalam menyampaikan ilmu sehingga penulis dapat menyelesaikan makalah ini.

Referensi

[1] Munir, Rinaldi. 2005. ”Diktat Kuliah IF2211 Strategi Algoritmik”. Bandung:Program Studi Teknik Informatika STEI ITB.

[2] ISO 7010 - E007, Evacuation Assembly Point. https://www.iso.org/obp/ui#iso:grs:7010:E007. Diakses 3 Mei 2020.

[3] Fire Evacuation Procedures: Choosing an Assembly Point. https://www.fireaction.co.uk/news/fire-evacuation-procedures-part-1-choosing-assembly-point/. Diakses 3 Mei 2020

[4] Assembly Point: Getting Tt Right. https://www.elitefire.co.uk/help-advice/assembly-points-getting-right. Diakses 3 Mei 2020

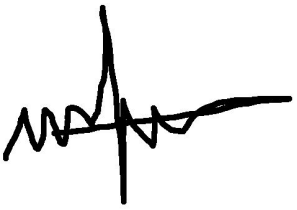
[5] Kementerian Kesehatan. 2016. Peraturan Menteri Kesehatan Republik Indonesia Nomor 48 Tahun 2016 Tentang Standar Keselamatan Dan Kesehatan Kerja Perkantoran.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 29 April 2020

Ttd



Muhammad Naufal Fakhrizal 13518115