Penerapan Teori Graf untuk Mencari Jalur Terpendek Menuju Turtle di Mobile Legends: Bang Bang

Brian Kheng - 13521049[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13521049@std.stei.itb.ac.id

***Abstract*—Penelitian ini bertujuan untuk mencari jalur terpendek yang harus dilalui untuk mencapai *Turtle* dari *Base*. Metode yang digunakan dalam penelitian ini adalah penerapan teori graf dan algoritma Dijkstra dalam bahasa C++. Hasil penelitian menunjukkan bahwa jalur terpendek untuk menuju *Turtle* dari *Base* ialah dengan mengambil jalur tengah lalu memotong *Jungle* langsung menuju *Turtle*.**

***Keywords*—C++, Dijkstra, Graf, Jalur Terpendek, Turtle.**

I. PENDAHULUAN

Mobile Legends: Bang Bang adalah *game multiplayer online battle arena* yang dikembangkan oleh *Moonton*. *Game* ini menawarkan *gameplay* yang sama dengan *game* MOBA lainnya, di mana dua tim bertarung melawan satu sama lain untuk mencapai dan menghancurkan *base* musuh. Setiap tim terdiri dari lima pemain yang saling bekerja sama untuk mengendalikan *hero* yang unik dengan kemampuan yang berbeda. *Game* ini menawarkan banyak mode permainan, seperti *Classic*, *Ranked*, dan *Brawl*, yang memungkinkan pemain untuk memilih cara bermain yang sesuai dengan preferensi mereka. *Mobile Legends* juga menawarkan sistem pembaruan *hero* dan *item* yang kaya, yang memungkinkan pemain untuk terus meningkatkan kemampuan *hero* mereka selama permainan.

Pada *game* ini, terdapat dua peta utama yang tersedia untuk dimainkan, yaitu *Classic Map* dan *Rank Map*. Kedua peta tersebut memiliki desain yang berbeda, tetapi tujuan utamanya sama, yaitu mencapai dan menghancurkan *base* musuh.

Classic Map adalah peta standar yang tersedia untuk semua pemain. Pada peta ini, terdapat dua *base* yang terletak di ujung masing-masing sisi peta. Di antara kedua *base* tersebut, terdapat jalur yang dikenal sebagai "lane" yang digunakan oleh *hero* untuk bergerak dan bertarung. *Lane* ini terdiri dari tiga jalur utama, yaitu *top lane*, *mid lane*, dan *bottom lane*. Selain itu, terdapat juga beberapa area di sekitar peta, seperti *jungle*, yang dapat dikunjungi oleh *hero* untuk mencari *item* dan mendapatkan keuntungan dalam permainan.

Rank Map adalah peta yang hanya tersedia untuk mode *Ranked*. Peta ini memiliki desain yang lebih kompleks dibandingkan *Classic Map*, dengan beberapa tambahan fitur seperti *turret* tambahan, *jungle* yang lebih luas, dan mekanisme yang berbeda untuk mengontrol "Lord" (boss yang kuat yang dapat membantu tim dalam menaklukkan musuh).

Kedua peta tersebut juga menyediakan berbagai macam fitur yang dapat membantu tim dalam mencapai kemenangan, seperti *tower* yang tersebar di sepanjang *lane*, "creep" yang memberikan *gold* dan *exp* ketika dikalahkan, dan "mines" yang dapat memberikan bonus *gold* jika dikuasai oleh tim. Selain itu, setiap peta juga memiliki beberapa monster kuat yang dapat dikalahkan untuk mendapatkan *item* dan keuntungan tambahan dalam permainan.

Turtle adalah salah satu monster yang ada di game *Mobile Legends: Bang Bang*. Monster ini terdapat di sekitar area *jungle* di kedua peta yang tersedia, yaitu *Classic Map* dan *Rank Map*.

Turtle adalah monster yang cukup kuat dan memiliki banyak *hit point*. Ketika dikalahkan, ia akan memberikan *gold* dan *exp* yang cukup banyak kepada tim yang berhasil mengalahkannya. Selain itu, *Turtle* juga akan memberikan *buff* kepada tim yang berhasil menguasainya, yang dapat meningkatkan kecepatan serangan dan kemampuan untuk menghasilkan *damage* lebih besar.

Menguasai *Turtle* adalah salah satu strategi yang dapat digunakan oleh tim untuk meningkatkan keunggulan dalam permainan. Dengan *buff* yang diberikan oleh *Turtle*, tim dapat lebih mudah mengalahkan monster lain di area *jungle*, menguasai area tersebut, dan mendapatkan keuntungan dalam permainan. Namun, strategi ini juga memiliki risiko, karena lawan juga dapat mencoba untuk mengalahkan *Turtle* dan mendapatkan *buff* tersebut. Oleh karena itu, tim harus berhati-hati dalam menggunakan strategi ini dan mempertimbangkan kondisi permainan yang ada sebelum memutuskan untuk menyerang *Turtle*.

Salah satu strategi yang dapat digunakan untuk menguasai *Turtle* ialah dengan sampai terlebih dahulu sebelum lawan. Agar dapat sampai lebih dulu, maka tim harus menempuh jalur terpendek dari posisi awal, yakni *Base* (posisi 1) menuju *Turtle* (posisi 19). Jalur dapat ditempuh melalui jalur-jalur valid yang ada (diberikan garis berwarna hijau yang menghubungkan satu titik ke titik lain).

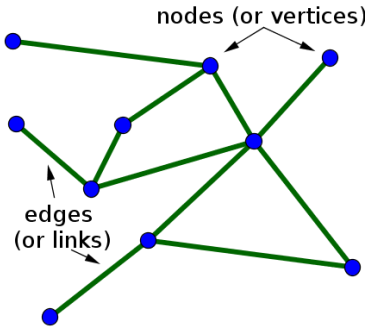
Gambar 1. Jalur valid yang dapat dilalui untuk menuju *Turtle* dari *Base* (ditunjukkan dengan garis merah)
(Sumber: Dokumen Pribadi)

Tujuan dalam penulisan makalah ini ialah untuk memperoleh strategi dalam game *Mobile Legends: Bang Bang*, yaitu mencari jalur terpendek yang harus dilalui agar dapat mencapai *Turtle* dari *Base* dalam waktu paling singkat. Langkahnya dengan menentukan bobot antar titik dan dengan algoritma Dijkstra menentukan jalur terpendek dari *Base* menuju *Turtle*.

II. Teori Dasar

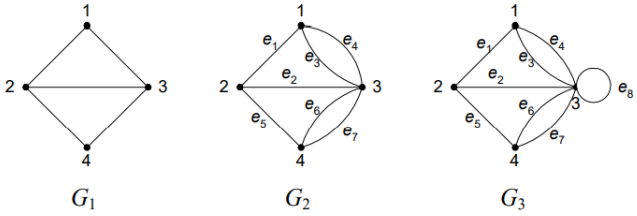
A. Definisi Graf

Graf adalah struktur data yang terdiri dari himpunan simpul (*node*) dan sisi (*edge*) yang menghubungkan simpul-simpul tersebut.



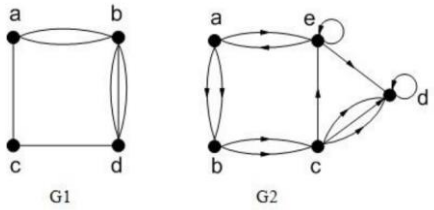
Gambar 2. Graf dengan 10 *nodes* dan 11 *edges*
(Sumber: https://mathinsight.org/network_introduction)

Graf dapat dibedakan menjadi dua jenis, yaitu graf sederhana dan graf tak-sederhana. Graf sederhana adalah graf yang tidak mengandung gelang maupun sisi ganda. Sebaliknya, graf tak-sederhana adalah graf yang mengandung *multiple edges* atau *self-loop*. Graf tak-sederhana dapat dibagi lagi menjadi dua jenis, yaitu graf ganda yang mengandung *multiple edges* dan graf semu yang mengandung *self-loop*. Graf sederhana lebih mudah diproses dibandingkan dengan graf tak-sederhana karena tidak memiliki *edge* yang terulang atau *edge* yang menghubungkan *node* dengan dirinya sendiri. Hal ini menyebabkan dalam banyak kasus penggunaan graf, graf sederhana lebih banyak digunakan dibandingkan dengan graf tak-sederhana.



Gambar 3. Dari kiri ke kanan, yakni graf sederhana, graf ganda, dan graf semu
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf)

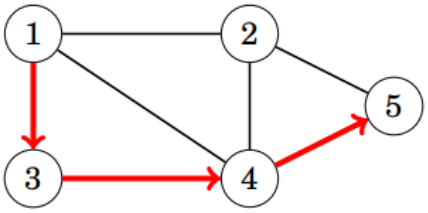
Graf dapat dibedakan menjadi dua jenis berdasarkan orientasinya, yakni graf berarah dan graf tak-berarah. Graf berarah adalah graf dengan *edges* yang menghubungkan *nodes* memiliki arah, sedangkan graf tak-berarah adalah graf dengan *edges* yang menghubungkan *nodes* tidak memiliki arah.



Gambar 4. Dari kiri ke kanan, yakni graf tak-berarah dan graf berarah
(sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf)

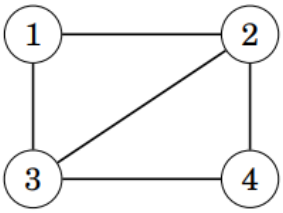
B. Terminologi Graf

Lintasan graf adalah jalur yang diambil dari *node* a menuju *node* b melalui *edges*, dengan panjang lintasan berupa jumlah *edges* yang dilalui. Sebagai contoh, jalur dari *node* 1 menuju *node* 5 ialah 1 → 3 → 4 → 5 pada gambar di bawah ini.



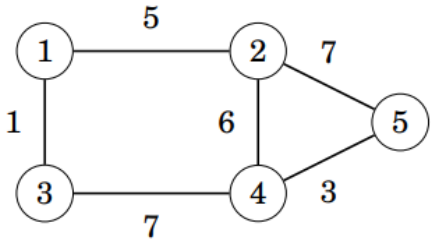
Gambar 5. Lintasan graf dari *node* 1 menuju *node* 5
(Sumber: *Competitive Programmer's Handbook*, Antti Laaksonen)

Suatu graf dapat dikatakan sebagai graf terhubung (*connected graph*) apabila untuk setiap *node* pada graf terdapat rangkaian *edge* yang menghubungkan antar *node*. Jika tidak, maka graf dikataan sebagai graf tak-terhubung (*disconnected graph*).



Gambar 6. Graf terhubung
(Sumber: *Competitive Programmer's Handbook*, Antti Laaksonen)

Graf berbobot (*weighted graph*) adalah salah satu jenis graf yang memiliki *edges* yang masing-masing memiliki sebuah bobot atau nilai tertentu. Bobot ini sering diinterpretasikan sebagai panjang *edge*, sehingga graf ini juga disebut dengan graf dengan panjang *edge*. Graf berbobot sering digunakan dalam menyelesaikan masalah pencarian jalur terpendek, seperti algoritma Dijkstra, karena dapat dengan mudah mengukur jarak antara *node* yang terhubung dengan *edge*.



Gambar 7. Graf berbobot
(Sumber: *Competitive Programmer's Handbook*, Antti Laaksonen)

C. Representasi Graf

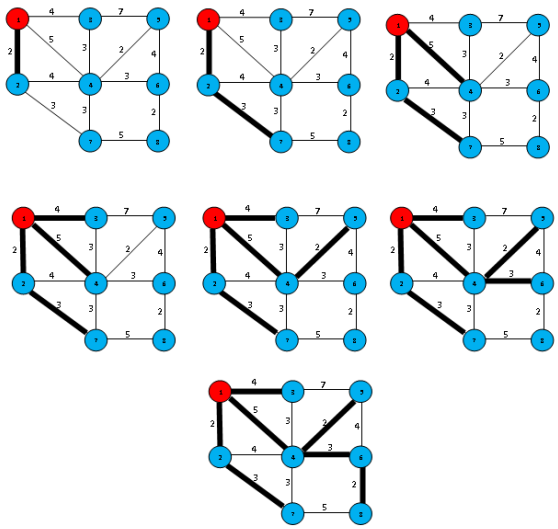
Terdapat beberapa cara dalam merepresentasikan sebuah graf, pemilihan struktur data tersebut bergantung kepada ukuran dan algoritma yang ingin digunakan dalam pemrosesan graf. Beberapa representasi yang umum, yakni:

1. Representasi *Adjacent List*
   Dalam representasi ini, setiap *node* dalam graf disimpan dalam sebuah *list* yang berisi semua *node* yang terhubung dengan *node* tersebut melalui *edge*. Setiap *node* akan memiliki *list* yang berisi semua *node* yang terhubung dengan *node* tersebut. Dengan menggunakan representasi ini, informasi tentang *edges* yang ada dalam graf dapat dengan mudah diakses dan diolah.
2. Representasi *Adjacent Matrix*
   Dalam representasi ini, sebuah matriks yang memiliki ukuran sesuai dengan jumlah *node* dalam graf digunakan untuk menyimpan informasi tentang *edges* yang ada dalam graf. Setiap elemen dalam matriks tersebut menunjukkan apakah ada *edge* yang menghubungkan dua *node* yang indeksnya sesuai dengan baris dan kolom dari elemen tersebut.
3. Representasi *Edge List*
   Dalam representasi ini, sebuah *list* yang berisi *pair* yang masing-masing *pair* menyimpan informasi tentang *nodes* yang terhubung oleh sebuah *edge* digunakan untuk menyimpan informasi tentang *edges* yang ada dalam graf.

D. Algoritma Dijkstra Untuk Mencari Jalur Terpendek

Algoritma Dijkstra adalah algoritma yang digunakan untuk menemukan jalur terpendek (*shortest path*) dari sebuah *node* asal ke semua *node* lainnya dalam sebuah graf berbobot. Langkah-langkah algoritma Dijkstra adalah sebagai berikut:

1. Tentukan *node* asal yang akan dicari jalur terpendeknya.
2. Buat sebuah set ***S*** yang berisi *nodes* yang sudah diproses. Buat sebuah array ***dist*** yang berisi jarak dari *node* asal ke semua *node* lainnya, diinisialisasi dengan nilai maksimal (*infinity*) kecuali jarak dari *node* asal ke dirinya sendiri, yang diinisialisasi dengan nilai 0.
3. Cari *node* dengan jarak terpendek dari *node* asal yang belum masuk ke dalam set ***S***, kemudian masukkan *node* tersebut ke dalam set ***S***.
4. Untuk setiap *node* yang terhubung dengan *node* yang baru saja dimasukkan ke dalam set ***S***, perbarui jaraknya jika terdapat jalur yang lebih pendek dari *node* asal ke *node* tersebut melalui *node* yang baru saja dimasukkan ke dalam set ***S***.
5. Ulangi langkah 3 dan 4 sampai semua *node* sudah dimasukkan ke dalam set ***S***.
6. Setelah semua *node* dimasukkan ke dalam set ***S***, jalur terpendek dari *node* asal ke setiap *node* lainnya dapat dicari dengan mengikuti urutan *node* yang dimasukkan ke dalam set ***S*** dan mengambil *node* tetangga dari *node* sebelumnya yang memiliki jarak terpendek.



Gambar 8. Algoritma Dijkstra dalam pencarian jalur terpendek dari *node* ke-1 ke semua *node* lainnya
(Sumber: https://www.researchgate.net/publication/271518595_A_New_Hardware_Architecture_for_Parallel_Shortest_Path_Searching_Processor_Based-on_FPGA_Technology)

III. ANALISIS

A. Mengubah Map Menjadi Graf Berbobot

Map ditambahkan titik-titik hijau yang mewakili *node* yang dapat dilalui oleh *hero*. Titik-titik dimulai dari *Base* (ujung kiri bawah) dan diakhiri di *Turtle* (tengah kanan). Dapat dilihat bahwa tidak semua bagian pada *map* tercakupi oleh *node*, karena penempatan *node* di arah kiri maupun belakang hanya akan memperpanjang jalur dan menjadi tidak efektif & efisien. Sehingga, penempatan *node* lebih condong ke arah kanan hingga tengah.

Gambar 9. Penempatan titik-titik hijau (*node*) dari *Base* menuju *Turtle*
(Sumber: Dokumen Pribadi)

Map yang telah ditambahkan *node* kemudian dicari panjang *edge* dari dua *node* yang dapat dilalui oleh *hero*. Range panjang edge dituliskan dalam bentuk desimal dan ditentukan berdasarkan edge ke-1 sebagai pivot. Dengan bantuan https://eleif.net/photomeasure dapat dicari *edge* yang nantinya akan membentuk suatu *connected graph*.

Gambar 10. Penempatan garis antar titik yang dilalui (*edge*) dari *Base* menuju *Turtle*
(Sumber: Dokumen Pribadi)

B. Penulisan Kode Algoritma Dijkstra Dalam Bahasa C++

Langkah 1: Melakukan inisiasi *header library*, *constant*, dan *variable* yang akan digunakan menggunakan struktur data yang efektif.

```
#include <bits/stdc++.h>
using namespace std;

const int MXN = 30; // Maximum node
const int INF = 1000000007;
int n, m, u, v;
double w, dist[MXN];
bool visited[MXN];
vector<pair<int, double>> adj[MXN];
vector<int> from(MXN, -1), ans;
priority_queue<pair<double, int>> pq;
```

Langkah 2: Menerima *input* berupa banyak *node* (*n*), banyak *edge* (*m*), dan untuk setiap *edge* ke-*i* ada titik u_i dan v_i yang saling terhubung dengan panjang w_i dan tiap elemen <*node*, *weight*> disimpan dalam suatu struktur data *adjacent list*.

```
cin >> n >> m; // Jumlah node dan edge

for(int i = 0; i < m; i++){
    cin >> u >> v >> w;
    --u; --v;
    // Mengisi adjacent list
    adj[u].push_back({v, w});
    adj[v].push_back({u, w});
}
```

Langkah 3: Mengubah jarak yang ditempuh dari *node* awal ke *node* awal menjadi nol dan *node* awal ke *node* lainnya menjadi INF.

```
for(int i = 0; i < n; i++){
    if(i == 0){
        dist[i] = 0;
    }
    else{
        dist[i] = INF;
    }
}
```

Langkah 4: Menginisiasi *priority_queue* dengan <0, 0> yang berarti dibutuhkan jarak sebesar 0 untuk mencapai *node* ke-1 dari *node* awal (*node* ke-1). Kemudian, *priority_queue* diurutkan meningkat berdasarkan total *weight* yang diperlukan untuk mencapai suatu *node* tertentu dari *node* awal. Elemen yang diproses merupakan elemen terdepan dari *priority_queue*, kemudian elemen akan dikeluarkan dari *priority_queue*. Lalu dilakukan pengecekan apakah *node* sudah pernah diproses sebelumnya atau belum, jika sudah maka *node* akan dilewati, namun jika belum, maka akan diiterasi untuk setiap *node* tetangganya dan mencari jarak minimal yang dibutuhkan ke *node* tetangganya dengan mencari *minimum(dist[v], dist[u] +*

w). Proses ini dilakukan terus menerus hingga *priority_queue* kosong (semua *node* telah dikunjungi).

```
pq.push({0, 0});

while(!pq.empty()){
    int idx = pq.top().second;
    pq.pop();
    if(visited[idx]){
        continue;
    }
    visited[idx] = true;
    for(auto it:adj[idx]){
        if(!visited[it.first]){
            if(dist[idx] + it.second <
               dist[it.first])
            {
                from[it.first] = idx;
                dist[it.first] = dist[idx] +
                                 it.second;
            }
            pq.push({-dist[it.first],
                     it.first});
        }
    }
}
```

Langkah 4: Lakukan *backtrack* dari *node* akhir hingga *node* awal untuk menemukan urutan *node* yang ditempuh untuk menghasilkan jalur terpendek dari posisi *Base* menuju *Turtle*.

```
int cur = n - 1;
while(cur != -1){
    ans.push_back(cur + 1);
    cur = from[cur];
}
reverse(ans.begin(), ans.end());
```

Langkah 5: *Output* jarak terpendek yang dibutuhkan untuk mencapai *Turtle* (*node* ke-19) dari *Base* (*node* ke-1) dan jalur yang diambil.

```
cout << dist[n - 1] << endl; // Min dist
for(auto it:ans){
    cout << it << ' ';
}
```

C. Hasil Uji Coba Program

Data masukan:
Terdapat 19 *nodes*, 30 *edges*, dan tabel *edges* (*node* 1 saling terhubung dengan *node* 2) sebagai berikut:

***node* 1**	***node* 2**	***weight***
1	2	1.00
1	4	0.95
1	6	1.01
2	3	0.49
3	4	0.28
4	5	0.28
5	6	0.45
6	9	0.53
9	5	0,34
9	8	0.23
8	7	0.27
4	7	0.41
6	10	0.65
10	11	0.31
9	11	0.51
11	12	0.39
12	13	0.29
8	13	0.50
11	18	0.46
17	18	0.27
10	17	0.44
18	19	0..45
12	19	0.42
13	19	0.59
7	14	0.49
14	15	0.27
15	16	0.30
16	19	0.69
14	16	0.42
13	14	0.43

Tabel 1. Daftar *node* yang saling terhubung beserta bobotnya
(Sumber: Dokumen Pribadi)

Hasil keluaran:

```
2.72
1 4 7 8 13 19
```

Jarak terpendek yang harus ditempuh untuk menuju *Turtle* dari *Base* ialah 2.72. Jalur yang diambil ialah *node* ke-1 (*Base*) → *node* ke-4 → *node* ke-7 → *node* ke-8 → *node* ke-13 → *node* ke-19 (*Turtle*).

Gambar 11. Jalur terpendek dari *Base* menuju *Turtle* (garis berwarna pink tua)
(Sumber: Dokumen Pribadi)

IV. KESIMPULAN

Algoritma Dijkstra dapat digunakan untuk mencari jalur terpendek yang harus ditempuh dari *Base* menuju *Turtle*. Algoritma ini dapat memberikan informasi berupa panjang jalur terpendek yang harus ditempuh berserta *node-node* yang perlu dilalui untuk mencapai *Turtle*. Jalur yang harus diambil ialah jalur tengah kemudian memotong dari *Jungle* menuju *Turtle*, yakni *node* ke-1 (*Base*) → *node* ke-4 → *node* ke-7 → *node* ke-8 → *node* ke-13 → *node* ke-19 (*Turtle*).

V. UCAPAN TERIMA KASIH

Terima kasih kepada semua pihak yang telah membantu dalam pembuatan makalah ini. Penulis menghargai semua masukan dan dukungan yang telah diberikan selama proses penyusunan makalah ini. Terima kasih juga kepada Ibu Dr. Nur Ulfa Maulidevi, S.T., M.Sc. selaku dosen pengampu mata kuliah IF2120. Penulis juga mengucapkan terima kasih kepada semua yang telah membaca makalah ini dan memberikan saran dan masukan yang berguna. Tanpa bantuan dan dukungan dari Anda semua, makalah ini tidak akan dapat terselesaikan dengan baik. Terima kasih atas semua bantuan dan dukungan Anda.

REFERENSI

[1] Nykamp DQ. (n.d.). *An Introduction to Networks*. https://mathinsight.org/network_introduction. Diakses pada 10 Desember 2022, pukul 08.16 WIB.

[2] Munir, R. (2020). *Graf (Bagian 1)*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf. Diakses pada 10 Desember 2022, pukul 13.23 WIB.

[3] Laaksonen, Antti. (2019). *Competitive Programmer's Handbook*.

[4] Al-Ibadi, M. (2012). *A New Hardware Architecture for Parallel Shortest Path Searching Processor Based-on FPGA Technology*. https://www.researchgate.net/figure/a-Network-topology-b-Steps-of-Dijkstra-algorithm_fig1_271518595. Diakses pada 10 Desember 2022, pukul 17.39 WIB.

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, 10 Desember 2022

Brian Kheng 13521049