# Perbandingan Penggunaan BFS dengan Multi-Source BFS dalam Strategi Permainan Platinum Rift 2

Faris Rizki Ekananda 13518125
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13518125@std.stei.itb.ac.id

***Abstrak*—Penggunaan algoritma Breadth First Search (BFS) dalam mengolah graf tanpa beban adalah hal yang biasa digunakan dalam pengolahan data, namun hal ini cukup memakan waktu apalagi jika terdapat batasan waktu yang ditentukan, sehingga kita harus menggunakan algoritma alternatif lain dalam pengolahannya. Pada makalah ini akan dibahas penggunaan BFS dan Multi-source BFS yang diterapkan pada permainan Platinum Rift 2.**

***Kata kunci—BFS, MS-BFS, Pathfinding, Shortest path***

## I. PENDAHULUAN

*Platinum Rift 2* adalah sebuah permainan pada situs *codingame.com* yang pada tahun 2015 lalu dikonteskan dan sejak saat itu dibuka untuk umum bagi yang ingin menerima tantangannya hingga sekarang. *Codingame* adalah sebuah situs yang menyelenggarakan berbagai macam kontes dan permainan yang dimainkan dengan membuat kodenya.

Pada permainan Platinum Rift 2, pemain dapat memprogram bot untuk mengeksekusi strateginya dalam meraih kemenangan. Bot ini berupa unit-unit POD (multi-agent) yang tiap unitnya harus dikendalikan pergerakannya. Dibutuhkan manajemen sumber daya, pengendalian waktu yang tepat, serta antisipasi Langkah yang lawan ambil untuk dapat memenangkan permainan ini. Perlu diperhatikan juga bahwa informasi yang didapat cukup terbatas karena adanya *fog of war* sehingga kita hanya bisa tahu lokasi markas kita, bot kita, serta markas musuh.


Gambar 1. Tampilan default Platinum Rift 2
(Sumber: dokumentasi penulis)

Permainan ini memiliki peta yang terdiri dari petak-petak heksagonal tanpa koordinat (hanya terdapat id) yang mengandung informasi keterhubungan, markas tiap pemain, platinum sebagai sumber daya yang dibutuhkan untuk membuat POD (dengan jumlah yang berbeda-beda pada tiap petak), serta POD yang terdapat pada arena permainan.

Permainan dimulai dengan kondisi semua petak adalah netral (kecuali markas). Pemain dapat mengambil alih petak (netral maupun kepunyaan lawan) dengan cara menempatkan POD pada petak tersebut. Apabila terdapat POD dari kedua pemain dalam satu petak, maka akan terjadi pertarungan. Pemain akan menang apabila dapat POD-nya dapat mencapai markas lawannya atau pemain yang menaklukkan petak terbanyak apabila permainan telah melewati 250 giliran. Apabila kedua pemain menaklukkan markas lawan secara bersamaan, atau setelah 250 giliran jumlah petak yang ditaklukkan sama, maka kedua pemain dinyatakan sebagai pemenang. Perlu diperhatikan untuk setiap turnnya, eksekusi kode strategi pemain tidak boleh melebihi 100 ms (namun pada praktiknya, tresholdnya mencapai 150 ms). Peraturan lebih lengkapnya dapat dilihat pada situs:
www.codingame.com/ide/puzzle/platinum-rift-episode-2 .

Untuk dapat melakukan penyerangan pada markas musuh, pemain harus menggunakan algoritma untuk menentukan jalur yang diperlukan oleh tiap POD-nya untuk mencapai markas musuh. Dengan mempertimbangkan peta yang tidak memiliki koordinat serta biaya memindahkan POD hanyalah langkahnya pada petak lain, penggunaan algoritma *Breadth First Search* (BFS) dirasa tepat untuk menentukan jalur terpendek dari POD menuju markas musuh. Makalah ini akan membahas kelebihan dan kekurangan dari pemilihan algoritma tersebut serta modifikasi yang diperlukan untuk memenuhi syarat-syarat pada permainan.

## II. TEORI DASAR

### A. *Graf*

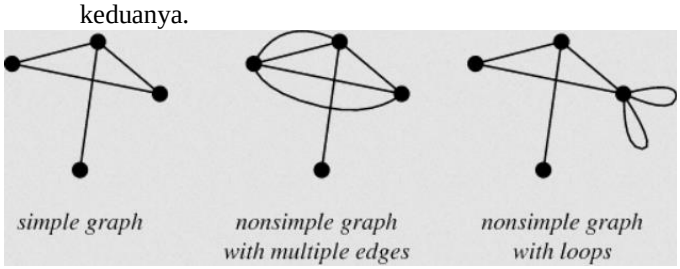
Graf secara sederhana adalah sebuah struktur yang terdiri dari objek-objek diskrit yang terkait. Secara matematis, objek-objek tersebut dinamakan simpu-simpul (*vertices*) dan tiap kaitan/hubungan antar simpul disebut sebagai sisi-sisi (*edges*). Secara formal, graf didefinisikan sebagai G = (V,E) dengan G adalah graf, V adalah himpunan tidak kosong dari simpul-simpul (*vertices*) $\{v_1, v_2, v_3, \ldots, v_n\}$, dan E adalah himpunan sisi

(*edges*) yang menghubungkan sepasang simpul $\{e_1, e_2, e_3, \ldots, e_n\}$.

Berdasarkan ada tidaknya gelang atau sisi-ganda pada suatu graf, graf dapat dibagi menjadi dua jenis:

1. Graf sederhana (*simple graph*)
   Graf yang tidak memiliki gelang maupun sisi-ganda.

2. Graf tak-sederhana (*unsimple graph*)
   Graf yang memiliki gelang, sisi-ganda, ataupun keduanya.



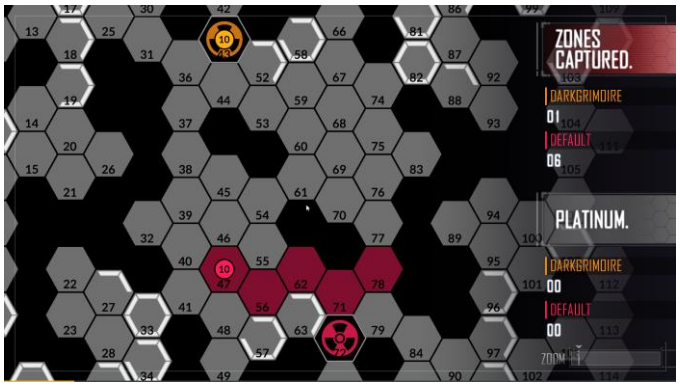

Gambar 2. Graf sederhana, graf tak sederhana dengan sisi ganda, serta graf tak sederhana dengan gelang.
(Sumber: mathworld.worlfram.com diakses 3 April 2020)

Berdasarkan ada tidaknya arah pada sisi, graf dapat dibedakan menjadi:

1. Graf tak-berarah (*undirected graph*)
   Graf yang setiap sisinya tidak mempunyai orientasi arah. Sebuah simpul dapat diakses dari simpul yang berhubungan secara dua arah.

2. Graf berarah (*directed graph* atau *digraph*)
   Graf yang setiap sisinya mempunyai arah. Dengan kata lain, simpul hanya bisa menuju simpul lain yang diarahkannya, atau dikunjungi simpul lain yang memiliki arah ke simpul tersebut.

Dalam konteks ini, graf yang akan digunakan adalah graf sederhana tak-berarah dan tidak berbeban.




Gambar 3. Struktur graf pada permainan Platinum Rift 2
(Sumber: dokumentasi penulis)

Terdapat berbagai macam cara (struktur data) untuk merepresentasikan sebuah graf pada implementasinya, yaitu:

1. Matriks ketetanggaan
   Pada matriks ketetanggaan, setiap baris adalah simpul dan setiap kolom adalah simpul yang dapat diakses dari simpul tersebut. Biasanya sel akan diisi dengan beban pergerakannya serta 0 apabila simpul tidak terkoneksi. Pada graf tak berbeban, beban pergerakan semua simpul ke simpul lain adalah 1. Matriks ketetanggaan memiliki implementasi yang mudah, namun dapat menggunakan ruang yang cukup besar untuk graf yang memiliki banyak sekali simpul, sehingga kurang ideal untuk matriks yang berukuran besar.

   Contoh matriks ketetanggaan:

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

2. List ketetanggaan
   Keterhubungan antar simpul didefinisikan dengan menggunakan linked list. Setiap simpul akan memiliki listnya tersendiri yang berisi simpul-simpul yang dapat diakses oleh simpul tersebut. Perlu diperhatikan bahwa hal ini bersifat satu arah, sehingga apabila simpul-simpul yang terhubung dapat mengakses simpul awalnya juga, maka akan didefinisikan pada list ketetanggaan simpul tersebut.

   Contoh list ketetanggaan:
   A → B, C
   C → A, B
   B → A, C

## *B. Algoritma Pencarian Graf*

Algoritma pencarian adalah algoritma yang menerima masukan berupa sebuah masalah dan menghasilkan solusi untuk masalah tersebut yang didapat dari evaluasi solusi-solusi yang mungkin. Dalam hal ini, masalahnya merupakan suatu graf yang ingin dicari solusi dengan kondisi tertentu. Algoritma pencarian dapat dikategorikan menjadi dua kategori yaitu pencarian tanpa informasi (uninformed/blind search) dan pencarian dengan informasi (informed/heuristic search).

1. Pencarian tanpa informasi

   Pencarian ini dilakukan dengan tidak memakai informasi mengenai sifat alami dari permasalahan yang dihadapi, sehingga dapat diimplementasikan secara umum dan dapat digunakan pada lingkup permasalahan yang luas. Akan tetapi, pencarian ini biasanya memiliki ruang lingkup solusi yang luas sehingga dapat membutuhkan waktu yang cukup lama untuk menemukan solusinya.

   Contoh pencarian tanpa informasi adalah BFS, DFS, UCS, DLS, IDS.

2. Pencarian dengan informasi

Pencarian ini dilakukan dengan memakai informasi berupa heuristik yang khusus digunakan pada permasalahan tertentu. Dengan menggunakan heuristik, pencarian menjadi jauh lebih cepat dibanding pencarian tanpa informasi karena pencarian memiliki pedoman yang dapat diikuti menuju tujuan sehingga tidak menyianyiakan waktu di jalur yang salah.

Contoh pencarian dengan informasi adalah Greedy Best First Search, A*.

Dalam proses pencarian solusi, terdapat dua pendekatan dalam representasi graf:

1. Graf statis

   Graf sudah terbentuk sebelum proses pencarian dilakukan.

2. Graf dinamis

   Graf tidak tersedia sebelum pencarian, melainkan dibangun selama pencarian solusi.

Karena kita memanfaatkan algoritma ini untuk mencari jalur terpendek, maka representasi graf yang dipakai adalah graf statis.

### C. *Algoritma Breadth First Search (BFS)*

Algoritma BFS adalah algoritma pencarian atau traversal struktur graf dan pohon. Pencarian dilakukan dari titik awal graf, lalu meluas ke cabang-cabangnya secara per level. BFS merupakan kebalikan dari *Depth First Search* (DFS) yang pencariannya menelusuri kedalaman suatu cabang terlebih dahulu sebelum berpindah ke cabang lainnya. Algoritma ini termasuk dalam kategori pencarian tanpa informasi.

Properti dari BFS adalah *complete* (selama nilai b terbatas), *optimal* (pada graf tanpa beban), serta memiliki kompleksitas waktu dan kompleksitas ruang $O(b^d)$, atau dapat diekspresikan juga sebagai $O(|V| + |E|)$ untuk kompleksitas ruangnya dan $O(|V|)$ untuk kompleksitas ruangnya.

BFS memiliki banyak kegunaan, seperti:

1. Mencari jalur terpendek untuk graf tak berbeban

   Dengan menggunakan BFS, akan selalu dijamin bahwa jalur yang ditemukan adalah jalur terpendek karena BFS menelusuri graf secara per level sehingga ketika ia mencapai solusi pertamanya, solusi tersebut adalah jalur terpendek (lintasan dengan level terkecil) dari titik asalnya.

2. Jaringan Peer-to-Peer

   Pada jaringan peer-to-peer, kita bisa menggunakan BFS untuk dapat mengetahui seluruh simpul jaringan yang ada.

3. Web Crawler

   Algoritma ini biasa digunakan pada situs-situs pencarian untuk melakukan *indexing* terhadap situs yang akan ditelusuri.

Berikut adalah pseudocode algoritma BFS untuk menemukan jalur terpendek dari titik awal menuju titik tujuannya.

```
1  procedure BFS(G, start_v) is
2      let Q be a queue
3      label start_v as discovered
4      Q.enqueue(start_v)
5      while Q is not empty do
6          v := Q.dequeue()
7          if v is the goal then
8              return v
9          for all edges from v to w in G.adjacentEdges(v) do
10             if w is not labeled as discovered then
11                 label w as discovered
12                 w.parent := v
13                 Q.enqueue(w)
```

Gambar 4. Pseudocode algoritma BFS
(Sumber: Wikipedia diakses pada 3 April 2020)

BFS adalah algoritma yang sangat cocok untuk digunakan apabila graf yang akan ditelusuri tidak memiliki beban. BFS dapat dengan mudah dimodifikasi menjadi persoalan-persoalan lainnya karena algoritma ini termasuk algoritma yang cukup umum untuk banyak permasalahan.

## III. Penggunaan Algoritma BFS dalam Strategi Platinum Rift 2

### A. *Struktur Data dan Persiapan*

Bahasa yang digunakan adalah Python 3.x.

Representasi graf yang digunakan adalah list ketetanggaan dengan implementasi menggunakan map of lists. Pada Platinum Rift 2, akan diberikan list of edges yang dapat kita manipulasi menjadi list ketetanggaan.

```
adj_map = {}
for i in range(LINK_COUNT):
    ZONE_1, ZONE_2 = [int(j) for j in input().split()]
    adj_map.setdefault(ZONE_1, []).append(ZONE_2)
    adj_map.setdefault(ZONE_2, []).append(ZONE_1)
```

Pada tahap persiapan ini juga, akan disiapkan beberapa variabel global seperti id pemain (`MY_ID`), total sisi (`LINK_COUNT`), total simpul (`ZONE_COUNT`), markas pemain (`MY_BASE`), serta markas lawan (`ENEMY_BASE`).

Terdapat dua kelas yang tersedia, yaitu kelas Node yang digunakan untuk mendapatkan jalur terpendek, dan kelas PriorityQueue yang menggunakan implementasi maxheap dari pustaka heapq. PriorityQueue sendiri tidak berhubungan dengan strategi BFS (BFS menggunakan queue biasa yaitu deque), kelas ini hanya digunakan agar POD dapat memilih jalur dengan platinum tertinggi pada strategi `findResources`.

## B. *Strategi Permainan Platinum Rift 2*

Terdapat 2 strategi yang digunakan pada makalah ini, yaitu:

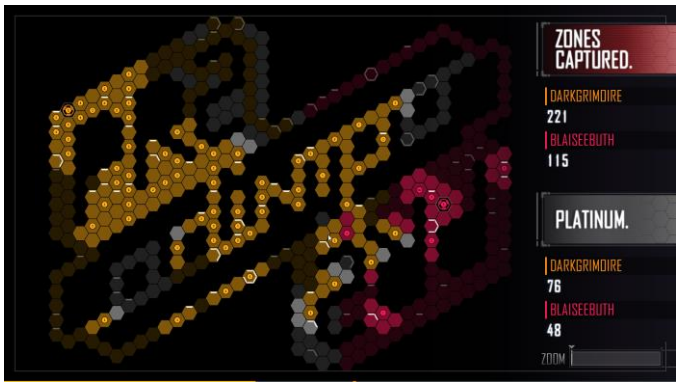
1. invadeEnemyBaseGuerilla (invasi)

   Strategi menginvasi markas lawan dengan cara menggerakkan seluruh POD pada arena menuju markas lawan. Strategi ini yang akan kita telaah mengenai kecepatan dan ruang memorinya. Strategi ini memiliki 2 implementasi yaitu yang menggunakan jalur algoritma BFS (`calculatePathBFS`) dan yang menggunakan jalur algoritma MS-BFS (`calculatePathMSBFS`). Terdapat optimisasi pada algoritma BFS yaitu apabila POD terdapat pada jalur hasil kalkulasi BFS sebelumnya, maka POD akan mengikuti jalur tersebut.


Gambar 5. Kondisi PODs ketika menggunakan strategi invasi dengan warna pemain kuning.
(Sumber: dokumentasi pribadi penulis)

2. findResources (pencarian sumber daya)

   Strategi agar POD dapat mencari platinum sebagai bahan sumber daya, menaklukkan daerah/petak lawan, dan melakukan penyebaran dari jalur invasi. Strategi ini digunakan hanya untuk keberlangsungan permainan dan tidak berpengaruh terhadap apa yang ingin dicari pada makalah ini.




Gambar 6. Kondisi PODs ketika menggunakan strategi findResources dengan warna pemain kuning.
(Sumber: dokumentasi pribadi penulis)

Strategi invasi akan dilakukan apabila platinum sudah mencukupi (50), area yang ditaklukkan sudah 1/3 arena, dan akan dilakukan rutin pada interval 25 turn dengan 20 turn invasi dan 5 turn mencari platinum dan akan dilakukan hingga kondisi sudah tidak terpenuhi.

Strategi pencarian sumber daya akan dilakukan apabila kondisi untuk menginvasi belum terpenuhi. Namun, apabila 20 turn telah lewat dan kondisi belum terpenuhi juga maka invasi akan dilakukan rutin pada interval 20 turn yaitu 10 turn invasi dan 10 turn mencari platinum dan akan dilakukan hingga kondisi untuk strategi invasi telah terpenuhi.

Pada setiap ronde permainan, kita akan mendapatkan informasi berupa petak-petak yang terlihat oleh kita (tidak terhalang oleh *fog of war*) dan mengandung informasi berupa id, platinum, kepunyaan, serta jumlah POD pemain yang terdapat pada petak tersebut.

## C. *Implementasi Penggunaan Algoritma BFS dan MS-BFS*

1. Implementasi BFS

   Implementasi BFS menggunakan algoritma yang sedikit berbeda dengan pengecekan simpul terletak pada ekspan simpul anak. Hal ini bertujuan untuk mempercepat penemuan simpul tujuan tanpa menunggu antrian yang panjang dan ekspansi dari simpul-simpul yang lebih dulu keluar dari antrian, mengingat batas waktu per ronde dari permainan ini adalah 100~150ms. Sehingga, simpul-simpul pada antrian sebenarnya sudah dicek apakah merupakan simpul tujuan atau tidak, tetapi tetap masuk ke dalam antrian untuk melakukan ekspansi. Hal ini menyebabkan tidak dilakukannya pengecekan pada simpul *source* sehingga akan menimbulkan masalah apabila *source sama dengan target*. Meskipun begitu, jarang sekali kondisi tersebut terjadi karena biasanya target yang dituju adalah markas lawan sementara jika POD sudah berada di markas lawan dia tidak perlu bergerak kemana-mana lagi sehingga pada POD tersebut tidak dilakukan algoritma ini.

   Algoritma ini memiliki kompleksitas waktu dan ruang yang sama dengan algoritma BFS pada umumnya yaitu $O(|V| + |E|)$ atau $O(b^d)$ untuk kompleksitas waktu dan $O(|V|)$ atau $O(b^d)$ untuk kompleksitas ruang. Karena BFS dilakukan untuk setiap POD pada arena, maka algoritma ini akan dieksekusi berkali-kali untuk tiap POD sehingga kompleksitas waktunya menjadi $O(V.E)$. Hal ini menyebabkan kompleksitas waktu dari algoritma ini meningkat untuk strategi invasi.

   Berikut adalah implementasi dari algoritma BFS.

```
def calculatePathBFS(source: int, target: int) -> list:
    """ Mendapatkan jalur terpendek dari source dengan menggunaka
n algoritma BFS """
    branch_raised = 0
    queue = deque()
    visited = []
```

```
    start_node = Node(None, source)
    finish_node = Node(None, target)
    queue.append(start_node)
    while queue:
        cur_node = queue.popleft()
        visited.append(cur_node.position)
        for child_pos in adj_map[cur_node.position]:
            if child_pos in visited:
                continue
            visited.append(child_pos)
            child_node = Node(cur_node, child_pos)
            child_node.g = cur_node.g+1
            child_node.f = child_node.g + child_node.h
            queue.append(child_node)
            branch_raised += 1
            if child_node == finish_node:
                result = []
                reconstruct_path(child_node, branch_raised, resul
t)
                result.append(source)
                return result[::-1]
```

2. Implementasi Multi-source BFS (MS-BFS)

Implementasi MS-BFS memiliki algoritma yang mirip sekali dengan implementasi BFS. Bedanya, algoritma ini menerima banyak simpul yang dijadikan *source*. Cara algoritma ini bekerja adalah dengan memutarbalikkan *source* dengan *target*. Karena *target* pada algoritma ini hanya 1 simpul, maka BFS sebenarnya dilakukan dari simpul *target* lalu akan dievaluasi jalurnya ketika mencapai simpul pada *source*. Algoritma akan berhenti ekspansi apabila seluruh simpul *source* telah ditemukan. Hasil dari algoritma ini berupa map dengan kunci yang merupakan setiap *source* dan nilai yang berisi jalur menuju *target* dari *source* tersebut.

Berbeda halnya dengan algoritma BFS, algoritma ini hanya dieksekusi sekali setiap dilakukan strategi invasi sehingga kompleksitas waktunya tetap sama yaitu O(|V| + |E|).

Berikut adalah implementasi dari algoritma MS-BFS.

```
def calculatePathMSBFS(source: list, target: int) -> dict:
    """ Mendapatkan jalur terpendek dari source dengan menggunaka
n algoritma Multi-source BFS (MS-BFS) """
    branch_raised = 0
    results = {}
    queue = deque()
    visited = []
    start_nodes = [Node(None, x) for x in source]
    finish_node = Node(None, target)
    queue.append(finish_node)
    while queue:
        cur_node = queue.popleft()
        visited.append(cur_node.position)
        for child_pos in adj_map[cur_node.position]:
            if child_pos in visited:
                continue
            visited.append(child_pos)
            child_node = Node(cur_node, child_pos)
            child_node.g = cur_node.g+1
            child_node.f = child_node.g + child_node.h
            queue.append(child_node)
            branch_raised += 1
            if child_node in start_nodes:
                src = next(x.position for x in start_nodes if x =
= child_node)
                start_nodes.remove(child_node)
                path = []
                reconstruct_path(child_node, branch_raised, path)
                path.append(target)
                results.setdefault(src, []).extend(path)
        if start_nodes == []:
            break
    print(f"Branch raised: {branch_raised}", end="",file=sys.stde
rr) # LOGGING PURPOSE
    return results
```

## IV. Pembahasan

Hasil ranking pada permainan Platinum Rift 2 untuk algoritma BFS adalah 568$^{th}$/1.611, sementara algoritma MS-BFS adalah 458$^{th}$/1.611. Mengapa ranking tersebut berbeda padahal strategi yang digunakan tetap sama? Hal ini disebabkan karena pada algoritma BFS, terdapat beberapa jenis map yang terlalu besar sehingga pada pertengahan permainan ketika cabang yang diekspan rata-rata mencapai 300-400 cabang, dan POD tersebar menjadi lebih dari 60 grup, algoritma BFS akan mengalami *timeout* karena sudah melewati batas maksimal eksekusi untuk 1 ronde yaitu 150ms. Apabila *timeout* terjadi, maka pemain dikenakan penalti yaitu tidak dapat menggerakkan POD-nya lagi sehingga lawan dapat dengan mudah mengalahkan pemain.

Data untuk keperluan statistik dikumpulkan dengan cara memainkan berkali-kali pada tingkat percabangan yang berbeda-beda pada tiap permainannya, yang diatur dengan mengatur jarak letak markas pemain dengan markas musuh dengan menggunakan pengaturan mode manual pada peta dengan petak terbanyak.

Statistik dari kedua algoritma (dalam milisekon [ms]):

a. Algoritma BFS

Waktu terburuk: *timeout* (lebih dari 150 ms)

Waktu maksimal: 147 ms dengan percabangan rata-rata 400 cabang dan terdapat sekitar 50 grup POD yang tersebar pada peta.

Waktu pada awal permainan (grup PODs < 20)

- Waktu untuk ±100 percabangan: 10~30 ms
- Waktu untuk ±200 percabangan: 20~45 ms
- Waktu untuk ±300 percabangan: 30~50 ms
- Waktu untuk ±400 percabangan: 55~84 ms

Waktu rata-rata permainan: 50 ms (dengan banyak sentakan di awal pergantian strategi mencari platinum ke strategi invasi yang rata-rata mencapai 100~120 ms)

b. Algoritma MS-BFS

Waktu terburuk: 14 ms (percabangan 405 cabang dan lebih dari 100 grup PODs yang tersebar pada peta. Tidak pernah terjadi *timeout*)

Waktu maksimal:

Waktu pada awal permainan (grup PODs < 20)

- Waktu untuk ±100 percabangan: 1 ms
- Waktu untuk ±200 percabangan: 3 ms
- Waktu untuk ±300 percabangan: 6 ms
- Waktu untuk ±400 percabangan: 8 ms

Waktu rata-rata permainan: 9 ms (tidak ada sentakan)

Dari data tersebut, kita dapat mengetahui bahwa hasil dari modifikasi algoritma BFS menjadi algoritma Multi-source BFS (MS-BFS) telah membuat kenaikan yang pesat terhadap performa dari pencarian jalur terpendek pada banyak simpul awal (*many sources to one destination*). Meskipun tidak terlihat adanya perbedaan dari segi penggunaan memori (terlepas dari penggunaan map untuk menyimpan hasil dari MS-BFS), tetapi efisiensi waktu jauh meningkat pada penggunaan algoritma MS-BFS hingga permainan tidak pernah lagi mencapai kondisi *timeout*.

## V. Konklusi

Dari hasil pembahasan tersebut, dapat disimpulkan bahwa penggunaan Multi-source BFS (MS-BFS) akan dapat meningkatkan efisiensi waktu BFS biasa secara drastis. Kondisi yang tepat untuk memakai algoritma MS-BFS adalah ketika algoritma BFS biasa harus dilakukan berkali-kali dalam satu waktu yang sama (atau dalam hal ini dalam satu ronde yang sama) untuk mencari jarak terpendek antara dua simpul pada graf tak berbeban dan tidak memiliki koordinat (tidak memiliki heuristik), dengan konfigurasi pencarian *many sources to one destination* atau sebaliknya.

Pada Platinum Rift 2, algoritma ini tidak hanya bisa diterapkan untuk target markas lawan saja, melainkan bisa untuk pencarian platinum, atau perintah-perintah lainnya. Algoritma MS-BFS lebih disarankan karena pada permainan ini pemain harus mengontrol banyak unit sekaligus dalam satu ronde permainan dengan waktu yang terbatas.

Berikut adalah link menuju *source code* dari program uji implementasi:

https://github.com/darkGrimoire/PlatinumRift2_Makalah

## Video Link di Youtube

https://youtu.be/k5aaiwiaKKM


## Ucapan Terima Kasih

Pertama, penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa atas berkah dan rahmatnya makalah ini dapat diselesaikan dan dikerjakan dengan sebenar-benarnya. Terima kasih juga kepada dosen pengajar mata kuliah IF2211 Strategi Algoritma K-02, Dr. Nur Ulfa Maulidevi, S.T., M.Sc., serta dosen-dosen pada matkul terkait Dr. Ir. Rinaldi Munir, M.T., Masayu Leylia Khodra, S.T., M.T., yang telah membagikan ilmu kepada seluruh peserta kuliah walaupun dalam kondisi tidak tatap muka. Terakhir, penulis ingin berterima kasih kepada teman-teman dan pihak-pihak lainnya yang telah membantu memotivasi dan memberikan pendapatnya dalam penulisan makalah ini.


## Daftar Pustaka


[1] Munir, Rinaldi; Maulidevi, Nur Ulfa; Khodra, Masayu Leylia. 2020. Diktat Kuliah Strategi Algoritma, Institut Teknologi Bandung: Bandung.

[2] GeeksforGeeks - https://www.geeksforgeeks.org/multi-source-shortest-path-in-unweighted-graph/ - Multi Source Shortest Path in Unweighted Graph (visited 1 April 2020).

[3] M. Then, M. Kaufmann, F. Chirigati, T.-A. Hoang-Vu, K. Pham, A. Kemper, T. Neumann, and H. T. Vo. The more the merrier: Efficient multi-source graph traversal. *Proceedings of the VLDB Endowment*, 8(4), 2014.

[4] *Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001) [1990]. "22.2 Breadth-first search". Introduction to Algorithms (2nd ed.). MIT Press and McGraw-Hill. pp. 531–539. ISBN 0-262-03293-7.*


## 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, 4 Mei 2020

Faris Rizki Ekananda 13518125