Aplikasi Algoritma Breadth First Search dalam Sistem Pencarian File dengan Metode BitTorrent

Ryan Daniel 13518130
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13518130@std.stei.itb.ac.id

Abstrak—Tidak jarang kita menemukan pemakaian aplikasi BitTorrent di sekitar kita. Beberapa orang menyukai aplikasi ini karena kecepatan unduhnya yang luar biasa. Pemanfaatan sistem peer-to-peer yang dapat mempercepat kecepatan unduh hingga puluhan bahkan ratusan kali. Ada juga yang mengklaim, file bajakan dapat diakses dengan mudah menggunakan BitTorrent. Namun, tidak sedikit juga yang kontra terhadap BitTorrent karena memiliki berbagai macam bahaya, seperti Malware, Adware dan kemanan data. Seberapa efisienkah BitTorrent yang diciptakan oleh BramCohen ini sehingga banyak orang yang meletakkan perangkatnya dalam risiko yang bisa dibilang tidak kecil? Penulis berharap dengan makalah ini, penulis dapat menjawab pertanyaan di atas.

Keywords—BitTorrent, optimasi, Breadth First Search

I. PENDAHULUAN

Pada abad ke-21, internet merupakan hal yang sudah tidak asing hamper bagi seluruh manusia di bumi ini. Ada salah satu aktivitas yang cukup krusial, yakni pengunduhan. Penguduhan memiliki arti memindahkan dari tempat yang lebih tinggi.

Faktor yang paling mempengaruhi kecepatan aktivitas ini tentu saja kecepatan internet itu sendiri. Sehingga, semakin cepat koneksi internet yang digunakan, semakin cepat juga kedua proses tersebut terjadi. Tentu saja, semakin cepat prosesnya selesai, semakin baik. Sayangnya, kecepatan internet sangat bergantung dengan provider dan demografi. Ditambah lagi, Indonesia yang menurut referensi[1], berada di peringkat 72 dari 77, sangat sulit bagi kita sebagai individu untuk mengoptimalkan kecepatan aktivitas tersebut karena hal tersebut berada di luar jangkauan kita. Oleh karena itu, tidak mungkin kita melakukan pengoptimalan pada faktor ini.

Untungnya, masih ada satu faktor lagi yang tidak kalah penting dengan kecepatan internet, yaitu algoritma pengunduhan itu sendiri. Ada dua algoritma penguduhan yaitu langsung dan peer-to-peer. Pengunduhan langsung yang berarti pengunduhan yang dilakukan secara langsung dari server yang menyediakan file yang ingin diunduh. Sedangkan penguduhan peer-to-peer yang berarti penguduhan dilakukan antarkomputer tanpa melalui suatu server.

Menurut referensi[2], kedua sistem ini tentu saja memiliki kelebihan dan kekurangannya masing-masing. Kelebihan dari pengunduhan langsung adalah keamanan yang lebih baik dan backup data yang harus secara manual dilakukan per computer. Sedangkan kelemahannya adalah memerlukan server yang relatif mahal. Untuk pengunduhan peer-to-peer keuntungannya adalah keberlangsungan pengunduhan tidak berlangsung pada server karena setiap komputer berperan sebagai server. Sedangkan kelemahannya adalah lebih rentan dalam sisi keamanan.

Dalam makalah ini, akan dibahas mengenai pengunduhan *peer-to-peer* karena kecepatannya yang lebih cepat dibandingkan pengunduhan langsung. Salah satu kakas yang memanfaatkan *peer-to-peer* adalah BitTorrent. Kita dapat menganggap suatu jaringan komputer yang menggunakan BitTorrent sebagai Graf. Anggap seorang pengguna ingin mengunduh suatu file. Tentu saja komputer pengguna tersebut tidak mengetahui komputer mana yang memiliki file tersebut.

BitTorrent memanfaatkan algoritma traversal graf untuk mencari lokasi file terdekat yang suatu pengguna ingin unduh. BitTorrent memanfaatkan algoritma *BFS* bukan *DFS.* Mengapa? Karena semakin sedikit langkah menuju simpul solusi, semakin optimal solusi tersebut. Ini merupakan karakteristik dari hasil solusi pencarian *BFS* bukan *DFS.* Oleh karena itu, penulis akan membahas pemanfaatan algoritma *BFS* sistem pengunduhan pada BitTorrent.

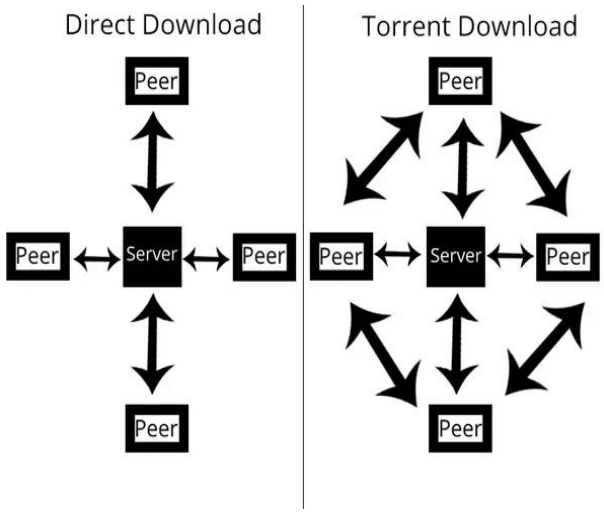
II. LANDASAN TEORI

A. *BitTorrent*

Menurut referensi[3], BitTorrent memiliki adalah sebuah protokol transfer internet untuk melakukan pengunduhan file. Namun, berbeda dengan *http* dan *ftp*, BitTorrent adalah protokol *peer-to-peer.* Karena *peer-to-peer,* BitTorrent mencari *user* yang memiliki file yang ingin diunduh lalu melakukan pengunduhan tersebut dari *PC* tersebut. Konsekuensinya, pengunduhan lebih cepat dibandingkan dengan *http* dan *ftp*, yang keduanya menggunakan server sebagai sumbernya.

BitTorrent memiliki suatu protokol dimana sebuah *user* yang pernah melakukan *request* penguduhan, harus bersedia diunduh file yang dimilikinya, hal ini karena dalam sistem BitTorrent, file tidak diunduh hanya dari satu komputer aja. File yang akan diunduh oleh seorang *user* akan dipecah menjadi bagian-bagian kecil. Sehingga, bahkan saat kita melakukan pengunduhan suatu

file (belum selesai), mungkin dapat terjadi juga sebuah pengunduhan dengan kita sebagai sumbernya di saat yang bersamaan.



Gambar 1 Ilustrasi Sistem Langsung dan BitTorrent

(Sumber: https://www.quora.com/What-is-the-difference-between-a-normal-download-and-a-torrent-download)

Istilah-istilah umum dalam BitTorrent:

1. *Availability*

   Angka dari ketersediaan file yang ingin diunduh, nilainya bisa kurang dari satu atau lebih. Semakin besar maka semakin baik dan lebih besar kemungkinan tingkat keberhasilan pengunduhan.

2. *Seeder*

   Suatu komputer yang sudah selesai melakukan pengunduhan dan siap melakukan pengunggahan. Secara umum semakin banyak *seeder* maka semakin cepat pengunduhannya, dan file tersebut secara utuh sudah tersebar dengan baik.

3. *Leecher*

   Orang yang sedang mengunduh torrent dan masih belum selesai.

4. *Swarm*

   Sebuah grup kolektif yang berisi *leecher* dan *seeder* yang dihubungkan oleh suatu file Torrent.

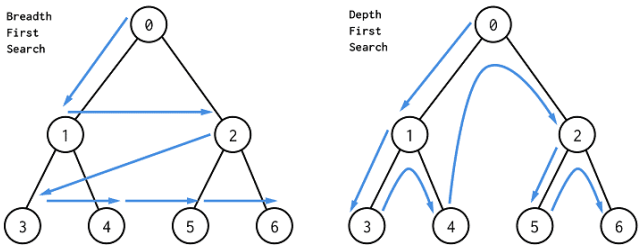
B. *Algoritma BFS dan DFS*

Breadth First Search (BFS) merupakan salah satu algoritma yang melakukan traversal pada graf. Algoritma ini melakukan kunjungan pada simpul dengan cara yang sistematik. Selain algoritma *BFS*, ada algoritma traversal graf lain yang mirip *BFS* yaitu *Depth First Search (DFS)*. Perbedaan dari keduanya adalah *BFS* melakukan pencarian secara melebar, sedangkan *DFS* melakukan pencarian secara mendalam. *BFS* biasanya menggunakan struktur data berupa *queue*, sedangkan *DFS* menggunakan struktur data *stack*.

```
Define : Graph G, Queue Q, List Inputs, List OutFlows, Parameter
MinFraction

Initialize: G for system with nodes containing tuples defined
(child = node, weight = fraction of emergy flowing to child)
Require: child weights sum to 1.0

Initialize: Inputs for system with tuples (node, flow)

Initialize: MinFraction as minimum fraction of each input to pass
to child process nodes

Initialize: Q as empty FIFO queue, Outflows as empty List

For input (node, flow)  in Inputs:
  Define : Set P
  Add: node to P
  Define : MinFlow = MinFraction * flow
  Define : Task T = (node, flow, P, MinFlow)
  Push: T to Q

While Q has elements:
  Pop: Task U = (node, flow, path, minflow)
  If node has no children:
    Define : Output O = (node, flow)
    Add: O to OutFlows

  For child in node (child, weight):
    Define : childflow = weight * flow
    If childflow >= minflow   AND child NOT IN path:
      Add: child to path
      Add: Task (child, childflow, path, minflow)   to Q

Collect: Sum outputs (node, flow) in Outflows by node
```

Gambar 2 *Pseudocode* algoritma *BFS*

(Sumber: https://www.researchgate.net/figure/Pseudo-code-of-the-BFS-algorithm-from-22_fig2_239526174)



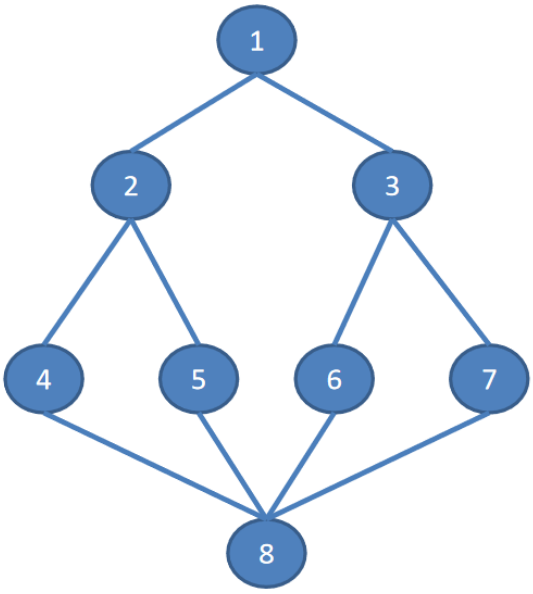
Gambar 3 Perbedaan pencarian solusi antara *BFS* dan *DFS*

(Sumber:https://www.freelancinggig.com/blog/2019/02/06/what-is-the-difference-between-bfs-and-dfs-algorithms/)

Dalam kedua algoritma ini, graf merupakan representasi persoalan yang akan diselesaikan sedangkan traversal yang dilakukan merupakan proses dari pemacahan masalah tersebut.

Algoritma pencarian traversal graf berdasarkan informasi yang dimiliki, dibagi menjadi dua, yaitu tanpa informasi (*uninformed search)* dan dengan informasi (*informed search*). *Uniformed search* merupakah pencarian traversal secara buta yang artinya tidak memiliki informasi tambahan, contohnya seperti *BFS*¸ *DFS, Depth Limited Search, Iterative Deepening Search,* dan *Uniform Cost Search.* Sedangkan *Informed search,* merupakan pencarian traversal dengan informasi tambahan yang berupa heuristik, contohnya *Best First Search* dan *A** atau *A-Star.*

Algoritma pencarian traversal graf berdasarkan proses pencarian solusi, dibadi menjadi dua, yaitu pendekatan graf statis dan pendekatan graf dinamis. Pendekatan graf statis adalah graf yang sudah terbentuk sebelum proses pencarian dilakukan yang menjadikan graf sebagai struktur data, contohnya pada permasalahan penelusuran direktori. Sedangkan pendekatan graf dinamis adalah graf yang terbentuk saat proses pencarian dilakukan. Sehingga, graf tidak tersedia saat pencarian belum dimulai, contohnya pada persoalan *N-Puzzle*.



Gambar 4 Ilustrasi algoritma *BFS*

(Sumber: Referensi[5])

Urutan algoritma pencarian solusi dengan *BFS*:

1. Traversal dimulai dari suatu simpul *v* dan jadikan simpul tersebut menjadi simpul ekspan.
2. Periksa apakah simpul tersebut solusi atau bukan. Jika iya, terdapat dua pilihan. Jika hanya butuh satu solusi saja, maka hentikan pencarian. Jika butuh lebih dari satu solusi, simpan simpul solusi tersebut dan lanjutkan ke langkah selanjutnya.
3. Bangkitkan semua simpul yang berhubungan dengan simpul ekspan lalu masukan semua simpul tersebut ke dalam *queue q*. Tandai semua simpul yang sudah pernah diekspan sehingga tidak perlu melakukan pengekspanan pada simpul yang sama.
4. Ambil simpul dari *head* pada *queue q*. Lalu, aplikasikan simpul tersebut pada langkah 2. Jika *head* kosong dan solusi belum ditemukan, maka pencarian solusi dinyatakan gagal.

Iterasi	V	Q	dikunjungi							
			1	2	3	4	5	6	7	8
Inisialisasi	1	{1}	T	F	F	F	F	F	F	F
Iterasi 1	1	{2,3}	T	T	T	F	F	F	F	F
Iterasi 2	2	{3,4,5}	T	T	T	T	T	F	F	F
Iterasi 3	3	{4,5,6,7}	T	T	T	T	T	T	T	F
Iterasi 4	4	{5,6,7,8}	T	T	T	T	T	T	T	T
Iterasi 5	5	{6,7,8}	T	T	T	T	T	T	T	T
Iterasi 6	6	{7,8}	T	T	T	T	T	T	T	T
Iterasi 7	7	{8}	T	T	T	T	T	T	T	T
Iterasi 8	8	{}	T	T	T	T	T	T	T	T

Urutan simpul2 yang dikunjungi: 1, 2, 3, 4, 5, 6, 7, 8

Gambar 5 Ilustrasi penyelesaian gambar 4 menggunakan algoritma *BFS*

(Sumber: Referensi[5])

Urutan algoritma pencarian solusi dengan *DFS:*

1. Traversal dimulai dari suatu simpul *v* dan jadikan simpul tersebut menjadi simpul ekspan.
2. Periksa apakah simpul tersebut solusi atau bukan. Jika iya, terdapat dua pilihan. Jika hanya butuh satu solusi saja, maka hentikan pencarian. Jika butuh lebih dari satu solusi, simpan simpul solusi tersebut dan lanjutkan ke langkah selanjutnya.
3. Bangkitkan semua simpul yang berhubungan dengan simpul ekspan lalu masukan semua simpul tersebut ke dalam *stack s*. Tandai semua simpul yang sudah pernah disekspan sehingga tidak perlu melakukan pengekspanan pada simpul yang sama.
4. Ambil simpul dari *top* pada *stack s*. Lalu, aplikasikan simpul tersebut pada langkah 2. Jika *top* kosong dan solusi belum ditemukan, maka pencarian solusi dinyatakan gagal.

III. Analisis dan Pembahasan

A. *Model Permasalahan*

Untuk menyamakan pandangan, penulis akan memodelkan masalah sebagai berikut:

1. *Seeder*, *leecher*, dan *PC* lainnya dianggap sebagai simpul.
2. Hubungan antarkomponen dalam *swarm* dianggap sebagai sisi.
3. *Swarm* dianggap sebagai graf.
4. *Latency* BUKAN heuristik karena *latency* diketahui setelah proses pengunduhan dimulai. Pemberian *latency* pada setiap simpul hanya untuk memudahkan pembaca untuk membayangkan pengoptimalan.

Sehingga, saat *leecher* melakukan pencarian, *leecher* akan mendapatkan *seeder* terdekat yang memiliki file tersebut dengan algoritma traversal graf. Hal ini dilakukan karena *latency* saat pengunduhan berbanding lurus dengan jarak.

Beberapa batasan dalam memodelkan masalah:

1. *Seeder* dalam suatu *swarm* dianggap memiliki kapasitas yang cukup sehingga kondisi pengunduhan dalam kondisi ideal.
2. Pengunduhan akan dimulai setelah pencarian sudah menemukan *seeder* dengan jumlah yang telah ditentukan sebelum pencarian dimulai.
3. Karena semakin banyak *seeder*, semakin cepat kecepatan pengunduhan, jumlah *seeder* lebih dari satu. Namun, jumlah *seeder* dalam suatu *swarm* dianggap tidak dinamis sehingga jumlahnya tidak pernah berubah.
4. Jumlah *leecher* hanya ada satu. Sehingga, tidak perlu dilakukan penghitungan pembagian *bandwith* antar-*leecher*.

5. Setiap langkah dari *leecher* menuju *seeder* dianggap sebagai *latency* yang diasumsikan setiap 10ms/ langkah.
6. *Swarm* akan dibentuk dari matriks ketetanggan sebagai model permasalahan.

B. *Implementasi Algoritma BFS*

Berikut adalah algoritma *BFS* yang sudah disesuaikan dengan kebutuhan BitTorrent itu sendiri(lihat gambar 6). Dalam implementasi ini, hanya digunakan *library* random untuk mengacak kepemilikan file.

```
#Preparasi
res = [] #Himpunan hasil
q = [] #Himpunan simpul hidup
dikunjungi = [False for i in range(20)]
latency = 0

#Algoritma BFS
q.append((0, latency))
while(len(res)<hasil and len(q)!=0):
    ekspan = q.pop(0)
    print('Ekspan:', ekspan[0])
    if(owned[ekspan[0]] == 1):
        res.append(ekspan)
    latency += 10
    for i in range(20):
        if(adjacency[ekspan[0]][i] == 1 and
 dikunjungi[i]==False):
            dikunjungi[i] = True
            q.append((i, latency))
    print('Hidup:', q)

if(len(res)!=0):
    print('Himpunan seeder: ')
    sums = 0
    count = 0
    for i in range(len(res)):
        sums += res[i][1]
        count += 1
        print('Seeder ke', i, 'Node', res[i
][0], 'Latency', res[i][1], 'ms')
    print('Latency rata-rata:', sums/count,
'ms')
    if(len(res)<hasil):
        print('Kekurangan seeder sebanyak',
 hasil-
len(res), 'karena tidak terhubung dengan no
de awal')
else:
    print('Gagal menemukan seeder.')
```

Gambar 6 Implementasi algoritma *BFS*

(Sumber: Penulis)

C. *Hasil Analisis*

Untuk melakukan analisis hasil *seeder* yang didapatkan, penulis akan membuat dua contoh persoalan dengan variabel bebas berupa jumlah *PC* pada *swarm* dan jumlah *seeder* yang dibutuhkan sebelum melakukan pengunduhan.

1. Percobaan ke-1

   Percobaan ini menggunakan jumlah *PC* pada *swarm* sebanyak 20 dengan *calon seeder* (pemilik file) sebanyak 7 *PC*.

```
0 0 0 0 0 0 1 1 0 1 0 0 1 1 0 1 1 0 1 1
0 0 0 1 0 0 0 0 1 0 1 1 0 0 0 1 0 1 1 0
1 1 0 0 0 0 1 0 0 0 0 0 1 1 0 1 0 1 0 1
0 0 0 0 0 1 0 0 1 1 1 0 0 0 1 0 0 0 0 1
0 0 0 0 0 1 1 0 0 1 0 0 1 0 0 0 1 0 1 1
0 1 0 1 0 0 0 0 0 1 1 1 0 0 1 1 1 1 0 1
0 0 0 0 0 1 0 0 0 1 0 0 1 0 0 1 1 0 1 0
0 0 0 1 1 0 0 0 1 1 1 0 1 0 0 0 1 0 0 1
1 1 1 0 1 1 1 1 0 1 1 1 0 0 1 0 0 0 1 0
1 1 1 1 0 0 0 1 0 0 1 0 1 0 0 0 1 0 1 1
0 0 0 0 1 0 1 0 1 0 0 0 1 0 1 1 0 1 1 0
1 0 1 0 0 1 1 0 0 0 1 0 0 0 0 0 1 1 1 1
1 0 1 1 1 0 1 0 1 1 1 0 0 0 1 0 1 0 1 0
1 0 0 0 1 0 1 1 0 1 1 1 1 0 1 1 0 0 0 1
1 0 1 0 0 1 0 1 0 0 1 1 0 0 0 1 0 0 0 1
0 0 0 1 1 0 0 1 0 1 0 1 0 1 1 0 1 0 0 1
0 0 0 0 0 1 1 1 1 0 0 1 0 1 0 0 0 0 0 1
1 1 1 1 1 0 0 0 1 0 1 1 1 1 1 1 0 0 1 1
1 1 0 1 0 1 0 0 1 0 1 1 0 1 1 1 1 0 0 0
0 1 1 0 0 0 0 0 1 1 0 0 1 1 1 1 0 1 0 0
```

Gambar 7 Matriks ketetanggaan persoalan 1

(Sumber: Penulis)

Dengan *array* kepemilikan sebagai berikut, [0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0] dan jumlah *seeder* yang dibutuhkan sebanyak 2 buah maka output yang diberikan program dapat dilihat pada gambar 8.

```
Ekspan: 0
Hidup: [(6, 10), (7, 10), (9, 10), (12, 10), (13, 10), (15, 10), (16, 10), (18, 10), (19, 10)]
Ekspan: 6
Hidup: [(7, 10), (9, 10), (12, 10), (13, 10), (15, 10), (16, 10), (18, 10), (19, 10), (5, 20)]
Ekspan: 7
Hidup: [(9, 10), (12, 10), (13, 10), (15, 10), (16, 10), (18, 10), (19, 10), (5, 20), (3, 30), (4, 30), (8, 30), (10, 30)]
Ekspan: 9
```

Hidup: [(12, 10), (13, 10), (15, 10), (16, 10), (18, 10), (19, 10), (5, 20), (3, 30), (4, 30), (8, 30), (10, 30), (0, 40), (1, 40), (2, 40)]

Ekspan: 12

Hidup: [(13, 10), (15, 10), (16, 10), (18, 10), (19, 10), (5, 20), (3, 30), (4, 30), (8, 30), (10, 30), (0, 40), (1, 40), (2, 40), (14, 50)]

Ekspan: 13

Hidup: [(15, 10), (16, 10), (18, 10), (19, 10), (5, 20), (3, 30), (4, 30), (8, 30), (10, 30), (0, 40), (1, 40), (2, 40), (14, 50), (11, 60)]

Ekspan: 15

Hidup: [(16, 10), (18, 10), (19, 10), (5, 20), (3, 30), (4, 30), (8, 30), (10, 30), (0, 40), (1, 40), (2, 40), (14, 50), (11, 60)]

Ekspan: 16Hidup: [(18, 10), (19, 10), (5, 20), (3, 30), (4, 30), (8, 30), (10, 30), (0, 40), (1, 40), (2, 40), (14, 50), (11, 60)]

Ekspan: 18

Hidup: [(19, 10), (5, 20), (3, 30), (4, 30), (8, 30), (10, 30), (0, 40), (1, 40), (2, 40), (14, 50), (11, 60)]

Ekspan: 19

Hidup: [(5, 20), (3, 30), (4, 30), (8, 30), (10, 30), (0, 40), (1, 40), (2, 40), (14, 50), (11, 60), (17, 100)]

Ekspan: 5

Hidup: [(3, 30), (4, 30), (8, 30), (10, 30), (0, 40), (1, 40), (2, 40), (14, 50), (11, 60), (17, 100)]

Ekspan: 3

Hidup: [(4, 30), (8, 30), (10, 30), (0, 40), (1, 40), (2, 40), (14, 50), (11, 60), (17, 100)]

Ekspan: 4

Hidup: [(8, 30), (10, 30), (0, 40), (1, 40), (2, 40), (14, 50), (11, 60), (17, 100)]

Ekspan: 8

Hidup: [(10, 30), (0, 40), (1, 40), (2, 40), (14, 50), (11, 60), (17, 100)]

Himpunan seeder:

Seeder ke 0 Node 5 Latency 20 ms

Seeder ke 1 Node 8 Latency 30 ms

Latency rata-rata: 25.0 ms

Gambar 8 Output persoalan 1

(Sumber: Penulis)

2. Percobaan ke-2

Percobaan ini menggunakan jumlah *PC* pada *swarm* sebanyak 30 dengan *calon seeder* (pemilik file) sebanyak 9 *PC*.

Gambar 9 Matriks ketetanggaan persoalan 2

(Sumber: Penulis)

Dengan *array* kepemilikan sebagai berikut, [0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1] dan jumlah *seeder* yang dibutuhkan sebanyak 3 buah maka output yang diberikan program dapat dilihat pada gambar 10.

Ekspan: 0

Hidup: [(1, 10), (2, 10), (4, 10), (9, 10), (10, 10), (11, 10), (15, 10), (16, 10), (18, 10), (19, 10), (20, 10), (21, 10), (22, 10), (24, 10), (25, 10)]

Ekspan: 1

Hidup: [(2, 10), (4, 10), (9, 10), (10, 10), (11, 10), (15, 10), (16, 10), (18, 10), (19, 10), (20, 10), (21, 10), (22, 10), (24, 10), (25, 10), (3, 20), (5, 20), (6, 20), (7, 20), (8, 20), (13, 20), (14, 20), (17, 20), (23, 20), (26, 20), (27, 20), (28, 20)]

Ekspan: 2

Hidup: [(4, 10), (9, 10), (10, 10), (11, 10), (15, 10), (16, 10), (18, 10), (19, 10), (20, 10), (21, 10), (22, 10), (24, 10), (25, 10), (3, 20), (5, 20), (6, 20), (7, 20), (8, 20), (13, 20), (14, 20), (17, 20), (23, 20), (26, 20), (27, 20), (28, 20), (0, 30), (12, 30)]

Ekspan: 4

Hidup: [(9, 10), (10, 10), (11, 10), (15, 10), (16, 10), (18, 10), (19, 10), (20, 10), (21, 10), (22, 10), (24, 10), (25, 10), (3, 20), (5, 20), (6, 20), (7, 20), (8, 20), (13, 20), (14, 20), (17, 20), (23, 20), (26, 20), (27, 20), (28, 20), (0, 30), (12, 30), (29, 40)]

Ekspan: 9

Hidup: [(10, 10), (11, 10), (15, 10), (16, 10), (18, 10), (19, 10), (20, 10), (21, 10), (22, 10), (24, 10), (25, 10), (3, 20), (5, 20), (6, 20), (7, 20), (8, 20), (13, 20), (14, 20), (17, 20), (23, 20), (26, 20), (27, 20), (28, 20), (0, 30), (12, 30), (29, 40)]

Ekspan: 10

Hidup: [(11, 10), (15, 10), (16, 10), (18, 10), (19, 10), (20, 10), (21, 10), (22, 10), (24, 10), (25, 10), (3, 20), (5, 20), (6, 20), (7, 20), (8, 20), (13, 20), (14, 20), (17, 20), (23, 20), (26, 20), (27, 20), (28, 20), (0, 30), (12, 30), (29, 40)]

Ekspan: 11

Hidup: [(15, 10), (16, 10), (18, 10), (19, 10), (20, 10), (21, 10), (22, 10), (24, 10), (25, 10), (3, 20), (5, 20), (6, 20), (7, 20), (8, 20), (13, 20), (14, 20), (17, 20), (23, 20), (26, 20), (27, 20), (28, 20), (0, 30), (12, 30), (29, 40)]

Ekspan: 15

Hidup: [(16, 10), (18, 10), (19, 10), (20, 10), (21, 10), (22, 10), (24, 10), (25, 10), (3, 20), (5, 20), (6, 20), (7, 20), (8, 20), (13, 20), (14, 20), (17, 20), (23, 20), (26, 20), (27, 20), (28, 20), (0, 30), (12, 30), (29, 40)]

Ekspan: 16

Hidup: [(18, 10), (19, 10), (20, 10), (21, 10), (22, 10), (24, 10), (25, 10), (3, 20), (5, 20), (6, 20), (7, 20), (8, 20), (13, 20), (14, 20), (17, 20), (23, 20), (26, 20), (27, 20), (28, 20), (0, 30), (12, 30), (29, 40)]

Ekspan: 18Hidup: [(19, 10), (20, 10), (21, 10), (22, 10), (24, 10), (25, 10), (3, 20), (5, 20), (6, 20), (7, 20), (8, 20), (13, 20), (14, 20), (17, 20), (23, 20), (26, 20), (27, 20), (28, 20), (0, 30), (12, 30), (29, 40)]

Ekspan: 19

Hidup: [(20, 10), (21, 10), (22, 10), (24, 10), (25, 10), (3, 20), (5, 20), (6, 20), (7, 20), (8, 20), (13, 20), (14, 20), (17, 20), (23, 20), (26, 20), (27, 20), (28, 20), (0, 30), (12, 30), (29, 40)]

Ekspan: 20

Hidup: [(21, 10), (22, 10), (24, 10), (25, 10), (3, 20), (5, 20), (6, 20), (7, 20), (8, 20), (13, 20), (14, 20), (17, 20), (23, 20), (26, 20), (27, 20), (28, 20), (0, 30), (12, 30), (29, 40)]

Ekspan: 21

Hidup: [(22, 10), (24, 10), (25, 10), (3, 20), (5, 20), (6, 20), (7, 20), (8, 20), (13, 20), (14, 20), (17, 20), (23, 20), (26, 20), (27, 20), (28, 20), (0, 30), (12, 30), (29, 40)]

Ekspan: 22

Hidup: [(24, 10), (25, 10), (3, 20), (5, 20), (6, 20), (7, 20), (8, 20), (13, 20), (14, 20), (17, 20), (23, 20), (26, 20), (27, 20), (28, 20), (0, 30), (12, 30), (29, 40)]

Ekspan: 24

Hidup: [(25, 10), (3, 20), (5, 20), (6, 20), (7, 20), (8, 20), (13, 20), (14, 20), (17, 20), (23, 20), (26, 20), (27, 20), (28, 20), (0, 30), (12, 30), (29, 40)]

Ekspan: 25

Hidup: [(3, 20), (5, 20), (6, 20), (7, 20), (8, 20), (13, 20), (14, 20), (17, 20), (23, 20), (26, 20), (27, 20), (28, 20), (0, 30), (12, 30), (29, 40)]

Ekspan: 3

Hidup: [(5, 20), (6, 20), (7, 20), (8, 20), (13, 20), (14, 20), (17, 20), (23, 20), (26, 20), (27, 20), (28, 20), (0, 30), (12, 30), (29, 40)]

Ekspan: 5

Hidup: [(6, 20), (7, 20), (8, 20), (13, 20), (14, 20), (17, 20), (23, 20), (26, 20), (27, 20), (28, 20), (0, 30), (12, 30), (29, 40)]

Ekspan: 6Hidup: [(7, 20), (8, 20), (13, 20), (14, 20), (17, 20), (23, 20), (26, 20), (27, 20), (28, 20), (0, 30), (12, 30), (29, 40)]

Himpunan seeder:

Seeder ke 0 Node 3 Latency 20 ms

Seeder ke 1 Node 5 Latency 20 ms

Seeder ke 2 Node 6 Latency 20 ms

Latency rata-rata: 20.0 ms

Gambar 10 Output persoalan 2

(Sumber: Penulis)

Dapat dilihat bahwa algoritma *BFS* sangat efektif dalam menemukan *seeder* dengan *latency* terkecil. Hal ini disebabkan karena *seeder* yang kita cari adalah *seeder* yang paling dekat dengan *leecher*. Kekurangan algoritma *BFS* dibandingkan *DFS* adalah saat *seeder* berada jauh dari leecher dalam suatu swarm karena iterasi pada *BFS* menghabiskan dahulu yang berada dekat dengan kita. Akan tetapi, *DFS* tidak aplikatif karena hal tersebut sangat jarang kemungkinannya terjadi, bahkan algoritma *DFS* dapat menyebabkan *leecher* melakukan pengunduhan dari tempat yang demografinya jauh, padahal bisa saja *PC* di sekitar *leecher* tersebut memiliki file yang ingin diunduh.

Selain itu, dapat dilihat juga bahwa semakin banyak *seeder*, semakin kecil *latency* yang didapatkan. Hal ini diakibatkan oleh sifat BitTorrent itu sendiri, yaitu semakin banyak *seeder*, semakin banyak pengunduhan paralel yang terjadi. Kekurangannya adalah kecepatan pengunduhan sangat bergantung dengan kecepatan pengunggahan *seeder* yang belum tentu selalu dalam keadaan optimal sesuai asumsi penulis. Berbeda dengan *server* yang tidak secepat BitTorrent namun karena spesifikasinya yang sudah canggih, mengakibatkan keadaan selalu stabil.

IV. PENUTUPAN

A. *Kesimpulan*

1. Saat melakukan pencarian *seeder*, algoritma *BFS* terbukti memberikan solusi yang lebih efisien dibandingkan *DFS* jika jumlah langkah menuju solusi dianggap sebagai *cost.* Namun, pada kenyataannya, sistem BitTorrent lebih kompleks dibandingkan model di atas. Misalnya, jumlah *seeder* yang dinamis dan tidak selalu dalam keadaan ideal, jumlah *leecer* yang bisa lebih dari satu, serta proses pengunduhan dapat dimulai saat jumlah *seeder* hanya satu saja.
2. Kecepatan unduh BitTorrent akan lebih cepat jika jumlah *seeder* semakin banyak dan jika hubungan *seeder* dan *leecher* pada *swarm* lebih dekat.

B. *Saran*

1. Dengan adanya kelebihan dan kekurangan pada BitTorrent, pengguna BitTorrent diharapkan dapat bijak dan sadar akan setiap tindakan yang dilakukan, terutama memilih file yang ingin diunduh. Hindari pengunduhan

file bajakan untuk menghormati *programmer* yang sudah membuat program tersebut, apalagi jika Anda adalah seseorang yang berkecimpung di dunia Informatika.

2. Pembaca diharapkan membaca makalah milik penulis yang berjudul "Aplikasi Algoritma Kruskal dalam Sistem Pengunduhan File dengan Metode BitTorrent" yang disediakan pada referensi [6] yang lebih spesifik membahas kecepatan pengunduhan dibandingkan makalah ini.

VIDEO LINK AT YOUTUBE

HTTPS://YOUTU.BE/VSDZBOUCS8I

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa karena atas berkat dan rahmatnya, penulis dapat menyelesaikan makalah yang berjudul "Aplikasi Algoritma Breadth First Search dalam Sistem Pencarian File dengan Metode BitTorrent".

Penulis juga mengucapkan terima kasih kepada tim dosen pengajar IF2211 Strategi Algoritma yang telah memberikan ilmu-ilmu baik yang berkaitan maupun tidak berkaitan dalam pembuatan makalah ini.

Penulis juga berterima kasih kepada kerabat maupun teman-teman penulis yang telah membantu baik secara langsung maupun moral dalam perkuliahan ini.

REFERENCES

[1] https://tekno.kompas.com/read/2019/02/22/07390037/kecepatan-internet-4g-indonesia-peringkat-72-dari-77-negara

[2] https://www.charis.id/jaringan-peer-to-peer/

[3] https://www.kaspersky.com/resource-center/definitions/bittorrent

[4] https://translatedby.com/you/terminology-of-bittorrent/original/

[5] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/BFS-dan-DFS-(2020).pdf

[6] http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2019-2020/Makalah2019/13518130.pdf

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.

Jakarta, 3 Mei 2020

Ttd

Ryan Daniel 13518130