Multisources Breadth-First Search untuk Menyelesaikan Pencarian Jalan Tercepat ke Tujuan Terdekat dalam Labirin

Wildan Dicky Alnatara(NIM 13516012)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha No. 10 Bandung 40132, Indonesia
alnataraw@gmail.com

***Abstrak*—Pencarian jalan terdekat atau sering disebut *shortest path finding* adalah salah satu masalah yang sangat menarik selain persoalan terkenal lainnya seperti *travelling salesman problem* karena akan mendapat hadiah jika dapat menyelesaikan travelling salesman problem dalam waktu polinomial. Makalah ini akan menjelaskan perbandingan waktu antara algoritma *breadth-first search* dengan *multisource breadth-first search*. Algoritma *breadth-first search* dimulai dengan sebuah titik sumber pencarian yang akan menyebar dengan mengunjungi tetangganya pada graf. Pada pencarian jalan tercepat dalam labirin atau *maze* semua jarak dari satu petak ke petak lain yang bertetangga adalah sama. Maksud dari bertetangga adalah petak yang berada pada arah atas,bawah, kiri dan kanan petak. Perbedaan yang sangat kelihatan antara algoritma *multisource breadth-first search* dengan algoritma *breadth-first search* adalah algoritma *multisource breadth-first search* dimulai dengan beberapa titik explorasi sedangkan *breadth-first search* hanya satu titik mulai eksplorasi (*Abstract*)**

Keywords— breadth-first search; multisource; algoritma; perbandingan.

I. PENDAHULUAN

Seiring dengan perkembangan zaman, dunia ilmu computer atau *computer science* semakin banyak berkontribusi dalam menyediakan kemudahan bagi manusia yang tinggal di muka bumi ini. Karena manusia sangat membutuhkan kemudahan dalam hidupnya maka dikembangkanlah berbagai teknologi untuk mendukung hidup manusia salah satunya adalah komputer. Komputer sangat mempengaruhi manusia terutama abad 20 ini. Banyak bidang-bidang yang berkembang seperti *artificial intelligence*, *data mining*, *networking*, dan masih banyak lagi. Komputer dapat mempermudah manusia dikarenakan dapat melakukan sesuatu dengan cepat bahkan dengan banyak operasi yang dilakukan oleh komputer. Walaupun begitu ternyata masih banyak persoalan yang belum bisa diselesaikan dengan cepat untuk mendukung bidang-bidang yang sedang berkembang pesat tersebut. Maka dari itu, pencarian algoritma yang mangkus hadir untuk lebih mempercepat komputasi sehingga lebih efisien dan efektif atau bisa dibilang lebih mangkus.

Pada makalah ini akan membahas tentang masalah labirin yang diselesaikan dengan algoritma *multisource breadth-first search* dengan banyak titik sumber eksplorasi yang selanjutnya akan dibandingkan dengan algoritma *breadth-first search* biasa. Hal ini dengan membandingkan kompleksitasnya dan juga dibuktikan dengan waktu eksekusinya.

II. DASAR TEORI

A. Algoritma Breadth-First Search

Algoritma *Breadth-First Search* yang disingkat BFS adalah algoritma yang digunakan untuk menelusuri struktur data graf. Dimulai dengan satu titik sumber lalu menelusuri simpul atau *node* tetangganya. Urutan traversal ditentukan dengan penelusuran penghidupan node yang pertama. Bisa dibilang Algoritma *Breadth-First Search* adalah algoritma yang menelusuri lapisan demi lapisan. Tidak seperti *Depth-first search* yang melakukan pencarian ke node yang lebih dalam terlebih dahulu hingga tidak ada node anak atau *children* yang bisa ditelusuri. Jika tidak ada maka *Depth-first search* akan melakukan *backtrack* untuk mencari simpul lainnya yang belum pernah dikunjungi. Algoritma *Breadth-First Search* tidak memakai konsep *backtracking*, biasanya *Breadth-First Search* memakai data struktur *queue* untuk mendata simpul atau node yang harus dikunjungi. Kompleksitas dari algoritma *Bread-first search* adalah O(V+E). Dalam kompleksitas tersebut V adalah jumlah *vertex* atau *node* atau simpul sedangkan E adalah jumlah sisi atau *edge*.

Berikut Algoritma BFS:

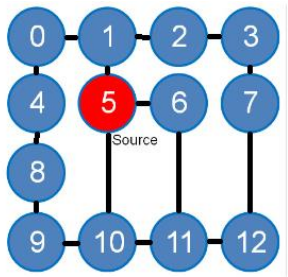
```
Algoritma :

1. Breadth-First Search dimulai dengan memasukkan sebuah titik sumber eksplorasi ke dalam antrian atau bisa dibilang queue. Bisa menggunakan priority queue atau queue biasa. Jangan lupa tandai titik sumber eksplorasi tadi sebagai titik yang sudah dikunjungi
2. Ambil simpul atau node u yang berada paling depan di queue atau antrian.
```

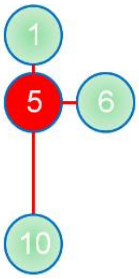
3. Masukkan semua simpul atau *node* yang belum pernah dikunjungi yang bertetangga dengan simpul *u*.

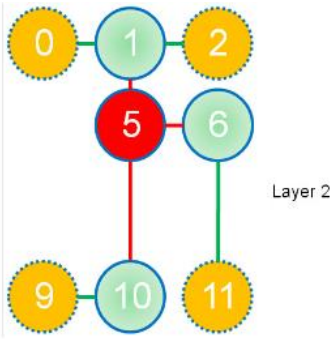
4. Tandai semua simpul atau *node* yang belum pernah dikunjungi yang bertetangga dengan simpul atau *node u* sebagai simpul yang sudah pernah dikunjungi

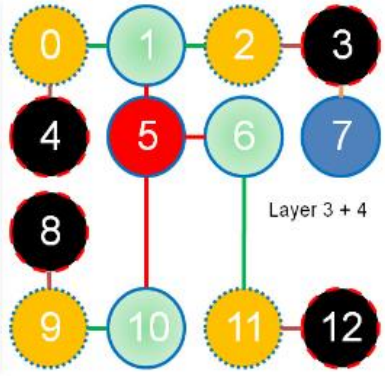
5. Jika antrian belum kosong maka kembali ke step algortma nomor 2

Berikut ilustrasinya :









Gambar 1. Ilustrasi BFS, diambil dari Buku Competitive Programming 3 oleh Steven Halim

B. Algoritma Multi-source Breadth-First Search

Algoritma *Multi-source Breadth-First Search* adalah algoritma dengan banyak titik sumber awal untuk ditelusuri. Hal ini memungkinkan perbaikan pada algoritma *Breadth-First Search* jika saja terdapat masalah dengan titik sumber yang banyak dan harus ditraversal seolah-olah berbarengan. Hal ini dapat mengurangi komputasi yang tidak perlu atau mengurangi mengulang-ulang algoritma *Breadth-First Search* di setiap titik sumber. Sangat tidak efisien tentunya. Algoritma *Multi-source Breadth-First Search* ini memakai ide *Breadth-First Search* akan tetapi *source point* ditelusuri seolah-olah berbarengan. Yaitu dengan memasukkan semua titik *source* kedalam data struktur antrian atau *queue*.

Berikut algoritma-nya:

Algoritma :

1. *Breadth-First Search dimulai* dengan memasukkan semua titik sumber eksplorasi ke dalam data struktur antrian atau bisa dibilang *queue*. Bisa menggunakan priority queue atau queue biasa.
   Jangan lupa tandai semua titik sumber eksplorasi tadi sebagai titik yang sudah dikunjungi

2. Ambil simpul atau *node u* yang berada paling depan di *queue* atau antrian.

3. Masukkan semua simpul atau *node* yang belum pernah dikunjungi yang bertetangga dengan simpul *u*.
4. Tandai semua simpul atau *node* yang belum pernah dikunjungi yang bertetangga dengan simpul atau *node u* sebagai simpul yang sudah pernah dikunjungi

```
5. Jika antrian belum kosong maka
   kembali ke step algortma nomor 2
```

Untuk kompleksitasnya untuk *worst case* yaitu O(V+E) masih sama dengan BFS biasa.

III. Penerapan Algoritma multiSOURCES Breadth-First Search untuk Memecahkan Masalah

Algoritma *Multi-source Breadth-First Search* ini akan digunakan untuk menyelesaikan masalah labirin yaitu mencari jalan tercepat ke tujuan untuk setiap petak labirin. Tujuan tersebut terdiri dari banyak tujuan sehingga harus memilih tujuan paling dekat dari alternative tujuan lainnya. Lalu setelah itu permasalahan ini akan dibandingkan dengan algoritma *breadth-first search* yang hanya menggunakan satu titik sumber eksplorasi. Hal ini berkairan dengan perbandingan kecepatan algoritma antara algoritma *Multi-source Breadth-First Search* dengan *Breadth-First Search.*

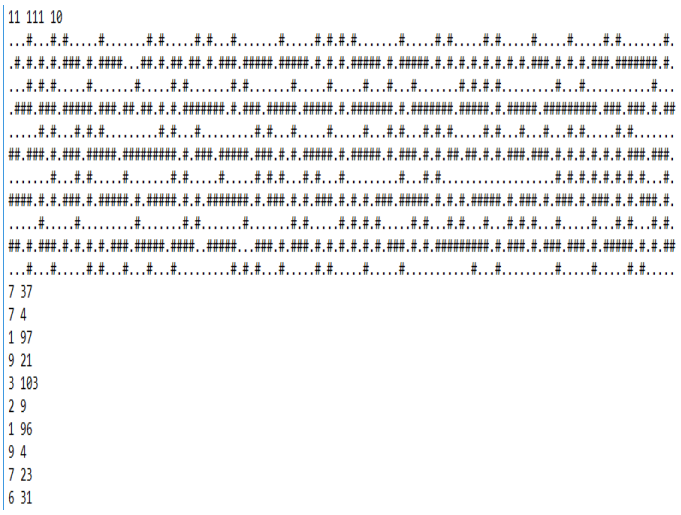
A. Pengenalan Masalah Nearest Goal Maze

Pada bagian ini akan dijelaskan salah satu *nearest goal maze problem.* Masih banyak masalah-masalah mirip seperti tapi ada makalah ini hanya akan digunakan satu contoh saja.

Diberikan sebuah peta berbentuk matriks dari labirin N x M yang mana sel kosong diwakili oleh simbol "." tanpa tanda kutip lalu sel terlarang untuk dieksplorasi diwakili oleh simbol "#" tanpa tanda kutip. Diberikan sejumlah K sel kosong sebagai tujuan dan juga tempat atau koordinat dari sel kosong sebagai tujuan. Pergerakan eksplorasi hanya bisa digerakkan ke arah atas, bawah, kiri, dan kanan.

Setiap sel kosong komputasikanlah *short distance* dan temukan *shortest path* dari sel kosong tersebut ke sel tujuan terdekat serta jumlahkanlah jarak terdekat dari setiap sel kosong ketitik tujuan. Untuk setiap perpindahan satu petak maka jaraknya adalah satu perpindahan. Untuk sel kosong yang dijadikan sebagai sel tujuan maka jarak sel tersebut ke sel tujuan terdekat adalah 0.

Berikut gambar contoh masukan



Gambar 2. Ilustrasi Input Masalah

Line pertama berupa input N,M,K lalu selanjutnya adalah peta. Setalah input peta terdapat K *line* yang adalah koordinat (x,y) yang adalah titik tujuan.

B. Analisis Algoritma dengan Bread-First Search

Pada bagian ini akan dijelaskan salah satu solusi yang mungkin jika menggunakan algoritma *Bread-First Search* biasa.

Salah satu yang mungkin dilakukan jika menggunakan menggunakan algoritma *Bread-First Search* adalah dengan menelusuri satu persatu setiap sel kosong lalu lakukanlah *Bread-First Search* pada sel tersebut lalu jangan lupa untuk selalu mencatat jarak. Jika sudah sampai pada tujuan maka catat jarak terpendek tersebut pada sel source. Lakukan untuk semua sel kosong.

Untuk mencari *shortest path* kita hanya tinggal menggunakan table jarak sel kosong yang sudah dibuat sebelumnya. Lalu traversal dengan mencari jarak terpendek yang ada pada bawah atau atas atau kiri atau kanan. Algoritma ini akan diberikan nama Algoritma 1.

Jika kompleksitas algoritma *Breadth-first search* yang paling optimal untuk saat ini adalah O(V+E) dan *Bread-First Search* dilakukan untuk setiap titik kosong eksplorasi maka kompleksitas algoritma *Bread-First Search worst case* untuk permasalahan ini dengan algoritma 1 adalah sebagai berikut

$$\boldsymbol{O\big(N * M(V + E)\big)}$$

V = N*M

E = N*M*4

Asumsi sisi mengarah ke 4 arah untuk setiap simpul atau node

Kompleksitas algoritma diatas digunakan untuk mencari jarak terpendek untuk setiap sel kosong ke tujuan terdekat. Untuk mengbangkitkan *shortest path* untuk setiap sel kosong dibutuhkan kompleksitas tambahan yaitu :

$$\boldsymbol{O(N * M * N * M)}$$

```
Algoritma 1 (Penghitungan jarak terdekat):
   1. Iterasi untuk setiap sel kosong
      koordinat (x,y). (x,y) akan dikenakan
      breadth-first search, jika sudah
      selesai lompat ke step 6, jika belum
      ke step 2

   2. Masukan (x,y) kedalam queue dengan
      jarak mulai 0 dan tandai kalau sudah
      dikunjungi
```

3. Jika queue kosong maka lompat ke langkah 1 jika tidak lanjut ke langkah 4

4. Ambil koordinat (xtemp,ytemp) dan s dari antrian atau queue terdepan, cek apakah (xtemp,ytemp) adalah tujuan , jika iya maka s adalah jarak terdekat untuk sel kosong (x,y) lalu lompat ke nomor 1, jika tidak lompat ke 5

5. Untuk semua sel kosong yang bersebelahan dengan (xtemp,ytemp). Masukan sel kosong yang bersebelahan tersebut ke dalam antrian atau queue dengan s ditambah 1 lalu lompat ke step 3.Jangan lupa tandai jika sudah dikunjungi untuk setiap sel yang dipilih

6. Selesai

Untuk Algoritma 2, salah satu yang mungkin lagi adalah dengan memandang sel-sel tujuan-tujuan sebagai titik eksplorasi atau titik *source*. Ini adalah perbaikan dari algoritma 1 yang mana nantinya sebagai pengantar agar bisa menyelesaikan persoalan ini dengan memakai *Multi-source Breadth-First Search.*

Kompleksitas untuk algoritma 2 ini adalah sebagai berikut

$$\boldsymbol{O\big(K * (V + E)\big)}$$

V = N*M

E = N*M*4

K = Jumlah titik tujuan

Asumsi sisi mengarah ke 4 arah untuk setiap simpul atau node

Kompleksitas Algoritma diatas digunakan untuk mencari jarak terpendek untuk setiap sel kosong ke tujuan terdekat. Untuk mengbangkitkan *shortest path* untuk setiap sel kosong dibutuhkan kompleksitas tambahan yaitu :

$$\boldsymbol{O(N * M * N * M)}$$

Algoritma 2(Penghitungan jarak terdekat) :

1. Iterasi untuk setiap sel kosong (x,y) yang dijadikan sebagai tujuan. .Jika sudah selesai ke step 6.
2. Masukan (x,y) berbarengan dengan jarak awal 0 kedalam data struktur queue atau bisa dibilang antrian. Jangan lupa untuk menandai sudah dikunjungi.
3. Cek apakah antrian kosong , jika kosong kembali ke step 1 , jika tidak kosong lanjut ke step 4
4. Ambil (xtemp,ytemp) dan jarak s pada antrian. Cek jarak s dengan jarak s' yang adalah jarak terpendek ke sel kosong tujuan sebelumnya. Jika lebih kecil update s' menjadi s.
5. Lalu tambahkan node atau sel yang bersebelahan dan belum pernah dikunjungi dengan simpul (xtemp,ytemp) dan tambahkan jarak s dengan 1 ke dalam antrian atau queue. Kembali ke step 3. Jangan lupa tandai bahwa telah dikunjungi.
6. Selesai

Untuk algoritma untuk mencari *shortest path* suatu sel kosong ke tujuan terdekat sebagai berikut :

Algoritma :

1. Misal untuk sel kosong yang ingin dicari jalannya adalah (x,y)
2. Tambahkan (x,y) ke list path , jika (x,y) adalah tujuan maka pencarian jalan selesai.
3. Jikan pencarian jalan belum selesai bandingkan semua sel yang bersebelahan atau bertetangga dengan (x,y) cari yang nilai jaraknya paling kecil. Lalu update (x,y) dengan koordinat simpul tetangga yang memiliki jarak terkecil lalu kembali step 2

C. Analisis Permasalahn dengan Algoritma Multisources Breadth First Search

Pada bagian ini akan dijelaskan salah satu solusi yang mungkin jika menggunakan algoritma *multisources Bread-First Search.*

Algoritma yang mungkin dengan menggunakan *multisources Bread-First Search* adalah dengan memiliki pandangan yang sama dengan algoritma 2 pada algoritma

Identify applicable sponsor/s here. If no sponsors, delete this text box (*sponsors).*

Bread-First Search biasa. Yaitu dengan memandang sel kosong tujuan-tujuan sebagai *source nodes*.

Dengan metode algoritma *multisources Bread-First Search* ini kita bisa mengurangi komputasi yang tidak penting. Seperti algoritma 2 yang selalu mengkomputasi untuk semua node untuk setiap sel kosong ataupun pada algoritma 1 komputasi *breadth-first search* dilakukan pada setiap sel kosong sehingga komputasi akan sangat besar dan waktu yang dibutuhkan akan bertambah besar untuk peta atau matriks yang sangat besar.

Kompleksitas untuk mencari jarak terpendek untuk setiap sel kosong adalah sebagai berikut :

$$\boldsymbol{O(V+E)}$$

V = N*M

E = N*M*4

Asumsi sisi mengarah ke 4 arah untuk setiap simpul atau node (yang pinggir-pinggir abaikan)

Jika dibandingkan dengan algoritma-algoritma sebelumnya, komputasinya jauh berkurang dibandingkan dengan *multisources Bread-First Search.* Hal ini sangat menguntungkan sehingga kita tidak perlu menunggu terlalu lama.

```
Algoritma MultiSources BFS :

  1. Masukkan semua sel kosong tujuan
     kedalam queue dengan jarak mulai 0.
     Tandai bahwa sudah dikunjungi.

  2. Cek apakah queue kosong, jika kosong
     maka algoritma selesai, jika belom
     kosong lanjut ke langkah selanjutnya

  3. Ambil (xtemp,ytemp)  dan s yang
     adalah jarak dari queue atau antrian
     paling depan.

  4. Masukkan semua node atau sel kosong
     yang bertetangga dengan (xtemp,ytemp)
     dan belum dikunjungi. Tambahkan jarak
     1 untuk node yang dimasukkan. Tandai
     koordinat yang dimasukkan tersebut
     bahwa telah dikunjungi. Kembali ke
     step 2.
```

IV. Hasil percobaan

Penulis telah membuat program yang menerapkan pseudocode diatas dengan beberapa penyesuain sesuai bahasa pemrograman yang dipakai. Bahasa pemrograman yang dipakai adalah C++.

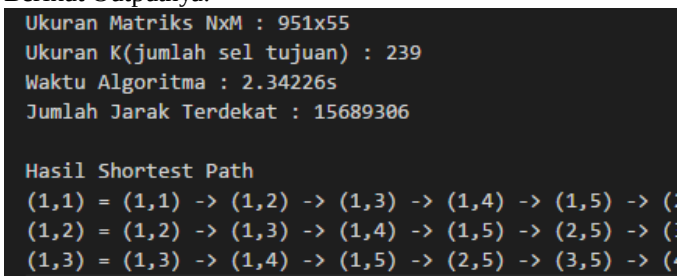
Berikut adalah pembahasan mengenai hasil keluaran dengan input seperti pada gamabr III.A.

A. Hasil Keluaran Program Breadth-First Search dan Waktu Eksekusi

Pertama, program akan meminta masukan pengguna, lalu program akan mengeluarkan waktu eksekusi, Jumlah semua jarak terdekat dari tiap sel kosong, dan juga shortest path untuk tiap sel. Program dapat dilihat pada pranala github yang saya cantumkan di referensi.

Penulis akan menggunakan input yang relatif besar sehingga performansi bisa dibandingkan secara signifikan. Contoh Input dapat dilihat pada pranala ke jejaring github profil saya yang ada pada referensi. Begitu Pula juga programnya.

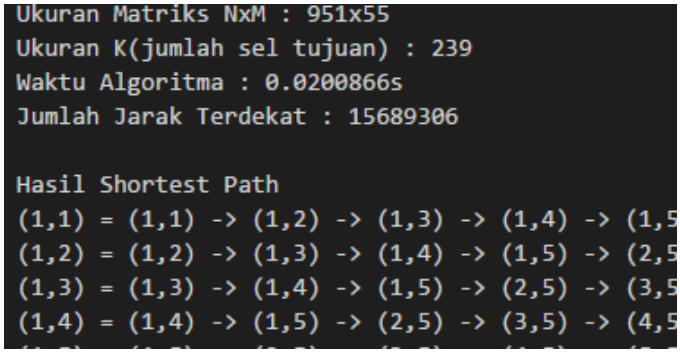
Berikut Outputnya.



Gambar 3. Sebagian output untuk BFS biasa

B. Hasil Keluaran Program Multisources Breadth-First Search dan Waktu Eksekusi

Program sama hanya saja algoritma yang digunakan berbeda. Pertama, program akan meminta masukan pengguna, lalu program akan mengeluarkan waktu eksekusi, Jumlah semua jarak terdekat dari tiap sel kosong, dan juga shortest path untuk tiap sel. Program dapat dilihat pada pranala github yang saya cantumkan di referensi.



Gambar 4. Sebagian output untuk Multi Source BFS biasa

C. Analisis Perbandingan

Jika dilihat pada gambar 3 dan gambar 4. Performansi algoritma sangat terlihat yaitu dengan perbedaan jauh antara waktu algoritma. Maka hasil sesuai dengan teori dan kompleksitas algoritma yang sudah dibahas pada bab Dasar Teori bahwa algoritma *Multisources Breadth-First Search* dapat menyelesaikan masalah ini lebih cepat dan signifikan dibandingkan BFS biasa.

KESIMPULAN

Algoritma *Multisources Breadth-First Search* dapat menyelesaikan masalah lebih cepat dibandingkan BFS biasa untuk persoalan yang mengharuskan terdapat beberapa titik sumber dan harus ditelusuri secara berbarengan. Hal ini dapat dijadikan sebagai pengantar untuk Algoritma Multi-sources Djikstra.

Penerapan algoritma yang dibahas pada makalah ini yaitu mencari jarak terdekat ke tujuan terdekat. Misal pada google maps kita ingin pergi ke toko baju terdekat. Dari sekian banyak toko baju. Kita ingin mengambil yang terdekat. Algoritma ini bisa diterapkan.

UCAPAN TERIMAKASIH

Pertama-tama, penulis mengucap syukur kepada Allah SWT karena atas berkat dan rahmat-Nya penulis dapat menyelesaikan makalah ini dengan baik. Juga berterimakasih kepada orang tua yang selalu mendukung selama perkuliahan semester 4 ini berlangsung. Dan juga terimakasih kepada Dr. Masayu Leylia Khodra, S.T., M.T., Dr. NurUlfa Maulidevi, S.T., M.Sc., dan Dr. Ir. Rinaldi Munir, M.T. selaku dosen mata kuliah IF2211 Strategi Algoritma yang telah mengajar, memberikan materi serta memberikan dukungan dengan baik,

REFERENSI

[1] Munir, Rinaldi.Diktat Kuliah Strategi Algoritma.unpublished.

[2] Levitin, Anany, Introduction to the Design and Analysis of Algorithms, 3rd ed. Addison-Wesley, Boston MA, 2002.

[3] Halim, Steven, Competitive Programming 3, 2013

[4] Source Code Program Makalah Stima: https://github.com/wildansupernova/Source-Makalah-Stima, diakses 14 Mei 2018

[5] CSAcademy Matrix Exploration, https://csacademy.com/contest/archive/task/matrix_exploration/, diakses 14 Mei 2018

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, 12 Mei 2018

Wildan Dicky Alnatara 13516012