Penerapan Algoritma BFS dan DFS untuk Mencari Langkah Terpendek Menyelesaikan Permainan 2048

Johanes 13517012
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13516052@std.stei.itb.ac.id

Permainan adalah hal yang standar bagi semua orang. Salah satu permainan yang sempat ramai dibicarakan adalah permainan 2048, banyak cara dan algoritma yang dapat diterapkan untuk menyelesaikan dan mendapat poin yang tinggi di permainan ini. Algoritma BFS dan DFS pun dapat digunakan untuk menyelesaikan masalah ini. Setelah diterapkan pada permainan 2048 dengan ukurang papan 2x2, ternyata didapatkan dibutuhkan 9 langkah minimum untuk menyelesaikan permainan yaitu dengan mendapat nilai 16.

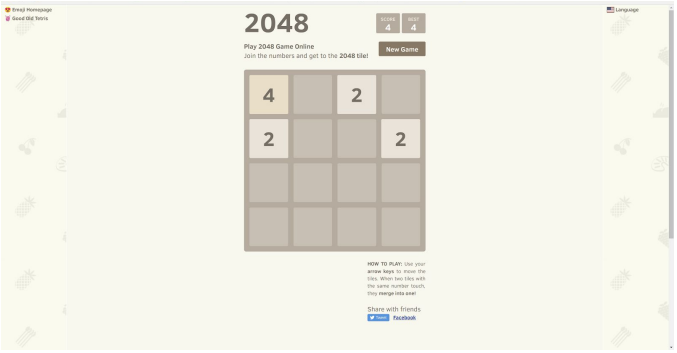
Keywords—2048; BFS; DFS; minimum

I. PENDAHULUAN

Dengan semakin berkembangnya jaman, banyak permain-permainan digital baru yang bermunculan. Mulai dari permainan yang sederhana hanya meng-c*lick* layar, hingga permainan yang kompleks seperti *e-sport* yang ada turnamennya. Permainan-permainan tersebut sangat berguna untuk mengisi waktu senggang dan melepas penat setelah melakukan tugas yang banyak.

Walau banyak permainan-permainan baru bermunculan, tidak sedikit orang yang masih memainkan permainan-permainan lama dengan harapan mengejar skor atau nilai tertinggi. Salah satu permainan yang sempat menjadi *trend* adalah permainan 2048. Permainan ini sangat sederhana dan cocok dimainkan di waktu senggang. Permainannya hanya perlu menggerakkan kotak-kotak untuk mencapai skor yang tinggi atau mencapai nilai 2048. Permainan ini dapat dimainkan online misalnya pada web:2048game.com/.

Sudah banyak algoritma bahkan kecerdasan yang dibuat untuk menyelesaikan permainan ini. Namun pada makalah ini akan dibahas khususnya pada penerapan algoritma BFS dan DFS untuk mencari langkah terpendek untuk mendapatkan nilai 2048.



Gambar 1. Tampilan halaman permainan di web 2048game.com

(sumber: http://2048game.com/)

II. DASAR TEORI

Pada bab ini akan dijelaskan beberapa teori pendukung dalam makalah ini.

2.1 Algoritma

Menurut Google, algoritma adalah "*a process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer*" [1], yang berarti algoritma adalah sebuah atau kumpulan dari proses atau aturan yang digunakan dalam pemecahan masalah dan terutama digunakan pada komputer. Biasanya algoritma merupakan alur yang sistematis dan runut

2.2 Traversal Graf

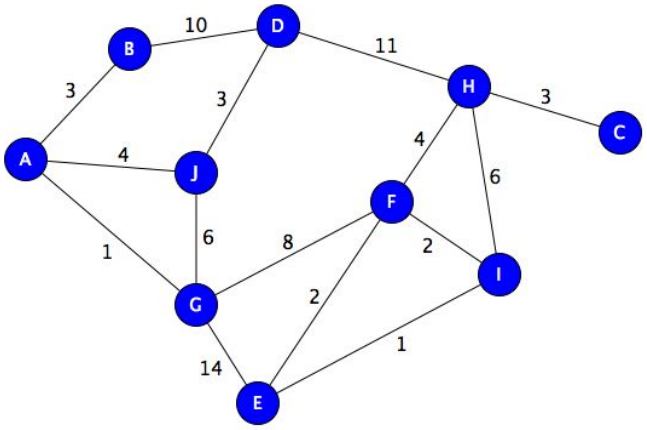
Traversal graf merupakan cara untuk mengunjungi setiap simpul pada sebuah graf dengan cara tertentu sehingga setiap simpul hanya dikunjungi tepat sekali. Pada umumnya cara yang paling mudah untuk mengetahui apakah sebuah simpul sudah dilewati adalah dengan memberikan tanda pada simpul tersebut.[2]

Dalam hal traversal graf dengan menggunakan algoritma, ada dua macam graf yaitu graf statis dan graf dinamis. Graf statis adalah graf yang setiap simpul dan vertexnya sudah terdefinisi dan dilakukan traversal graf dengan menggunakan algoritma. Sedangkan graf dinamis adalah graf yang dibangun selama proses traversal dilakukan oleh algoritma. Beberapa algoritma yang dapat digunakan untuk traversal graf adalah BFS dan DFS.

2.3 BFS

BFS adalah singkatan dari Breadth-First-Search. BFS merupakan salah satu algoritma yang digunakan dalam melakukan traversal graf. Sesuai dengan namanya, algoritma BFS melakukan traversal terhadap semua simpul yang bersebelahan dengan simpul awal atau simpul sekarang secara melebar dan dilakukan secara bertahap *layer* demi *layer*.

Karena pada graf dapat mengandung siklus, maka tentu saja untuk menghindari eksekusi algoritma yang tidak berhenti diperlukan suatu penanda untuk simpul-simpul yang pernah dilewati. Cara paling mudah adalah dengan membuat suatu larik *boolean* yang menandakan apakah suatu node ke-*i* sudah pernah dikunjungi. Solusi lain untuk masalah ini adalah dengan membuat suatu struktur simpul sendiri yang memiliki nilai *boolean* sebagai atributnya, sehingga selama traversal dapat diubah nilai atribut tersebut untuk menandakan dan akan selalu dicek saat akan menambahkan simpul ke barisan.

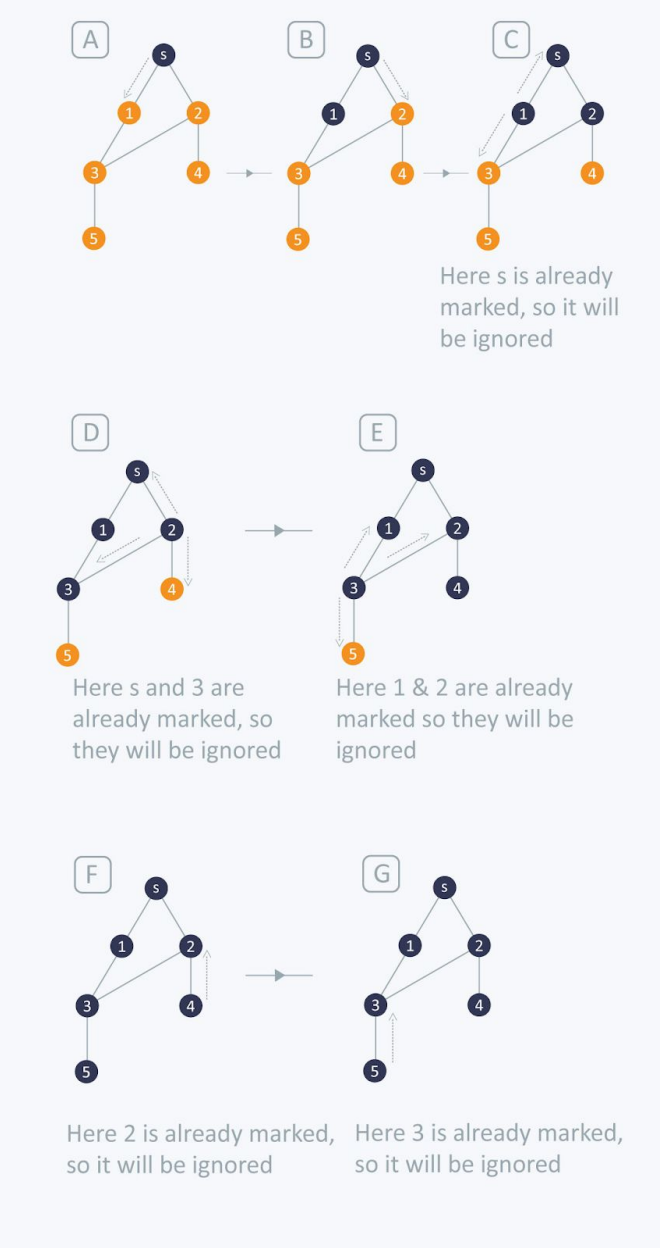


Gambar 2. Contoh graf

(sumber: https://i.stack.imgur.com/sImrp.png)

Diberikan contoh graf di atas. Misalkan kita akan memulai traversal graf dari simpul A, maka urutan simpul yang akan dikunjungi adalah: {A,B,J,G,D,F,E,H,I,C}. Penerapan algoritma ini sangat mirip dengan prinsip *queue* atau barisan yang menerapkan konsep *First In First Out* (FIFO). Jadi untuk setiap simpul yang dikunjungi, masukan simpul tetangga ke belakang barisan secara terurut, sehingga akan dikunjungi dengan bertahap.

Untuk gambaran yang lebih jelasnya dapat dilihat gambar di bawah ini.



Gambar 3. Cara traversal graf dengan menggunakan algoritma BFS

(sumber: https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/)

2.4 DFS

Serupa namun tak sama, DFS dan BFS sama-sama digunakan untuk melakukan traversal pada graf. DFS merupakan singkatan dari Depth-First-Search. Algoritma BFS dan DFS memang umumnya digunakan dalam traversal graf untuk melakukan pencarian terhadap suatu tujuan. Misalnya

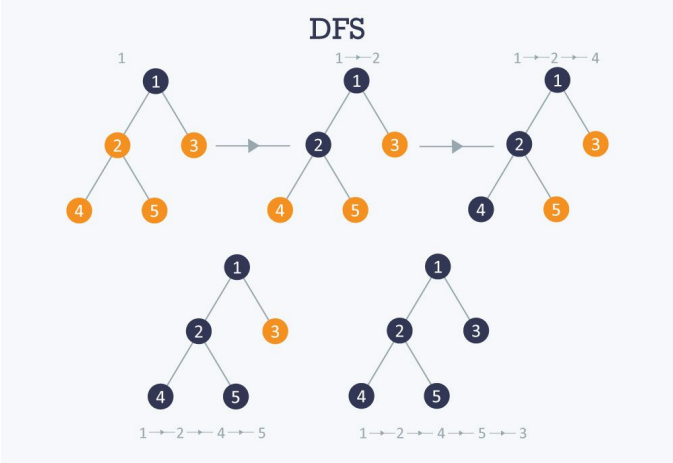
mencari suatu simpul yang memenuhi kriteria tertentu atau mencari jalur vertex yang memenuhi syarat tertentu.

Cara kerja algoritma DFS adalah dengan mengunjungi satu simpul yang bersebelahan dan diulangi terus-menerus hingga menemukan simpul ujung, Yaitu simpul yang tidak memiliki tetangga lain selain simpul sebelumnya. Lalu setelah itu akan dilanjutkan dengan mengunjungi simpul tetangga lain yang belum dikunjungi dengan menggunakan metode *back-tracking*.

Back-tracking pada graf adalah cara untuk kembali ke simpul sebelum simpul sekarang dan mencari tetangga lain yang belum dikunjungi. Karena pada saat menjalankan algoritma DFS mungkin kembali ke simpul asal jika terdapat siklus maka juga diperlukan suatu penanda misalnya dengan menggunakan larik atau atribut.

Misalkan kita menggunakan graf pada gambar 2 untuk menerapkan DFS untuk traversal graf yang dimulai dari simpul A dengan simpul yang memiliki urutan abjad yang lebih kecil yang dikunjungi dahulu. Maka urutan simpul yang dikunjungi adalah: {A,B,D,H,C,F,E,G,J,I}. Algoritma ini bekerja mirip dengan stack yang menerapkan prinsip Last In First Out (LIFO). Karena setiap simpul yang baru dikunjungi akan mem-*push* simpul tetangganya ke dalam stack dan simpul dengan nilai abjad yang besar diletakkan terlebih dahulu di stack. Sehingga akan selalu dipilih simpul dengan nilai abjad yang terkecil dengan terurut.

Untuk gambaran yang lebih jelasnya dapat dilihat pada gambar di bawah.



Gambar 4. Cara traversal graf dengan menggunakan algoritma DFS

(sumber: https://www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial/)

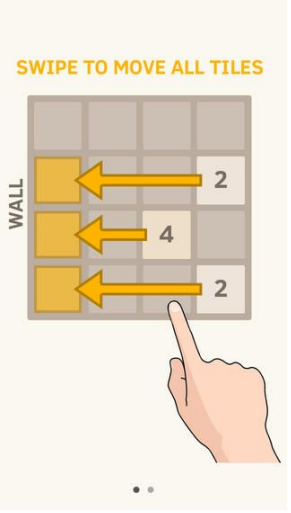
2.5 Permainan 2048

Permainan 2048 diciptakan oleh Gabriele Cirulli, seorang pengembah web asal Italia pada saat ia berusia 19 tahun. Ia berhasil membuat permainan puzzle ini hanya dalam waktu dua hari di akhir pekan dan dipublikasikan pada tanggal 9 Maret 2014.[3] Game ini adalah game pertama yang ia ciptakan dan dalam seketika permainan ini menjadi sangat terkenal di seluruh dunia, layaknya permainan angry bird. Hingga saat ini sudah banyak aplikasi permainan yang meniru dan mengambil ide dari permainan 2048 ini.

Gambar 5. Gambar permainan 2048 yang original

(sumber: https://fscl01.fonpit.de/userfiles/4774964/image/AndroidPIT-2048-Gabriele-Cirulli-w628.jpg)

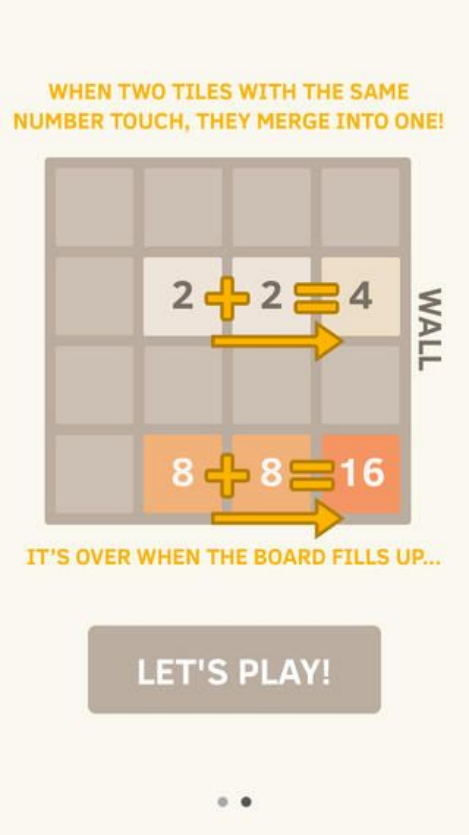
Cara bermain permainan ini sangat sederhana, namun sangat membuat kecanduan. Tujuan dari game ini adalah untuk mencapai nilai 2048 dengan cara menggabungkan kotak-kotak yang ada.



Gambar 6. Cara bermain 2048

(sumber: http://iphonetop25app.blogspot.com/2014/03/2048.html)

Kotak dapat digerakkan dengan men-*swipe* pada layar (permainan di gawai). Lalu semua kotak akan bergerak ke sisi tersebut.



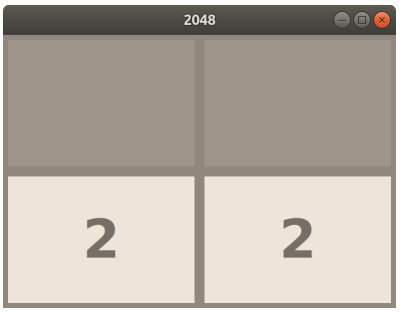
Gambar 7. Cara bermain 2048

(sumber: http://iphonetop25app.blogspot.com/2014/03/2048.html)

Jika kotak yang bergerak memiliki nilai yang sama, maka akan dijumlahkan dan menjadi kotak baru. Hal ini hanya berlaku untuk dua kotak terdekat. Jika terdapat tiga kotak yang bernilai dua dan bersebelahan, jika di *swie* hanya akan menghasilkan satu buah kotak empat dan kotak yang bernilai dua akan tetap, hanya posisinya saja yang berubah.

III. ALGORITMA

Algoritma yang digunakan ada dua, yaitu BFS dan DFS, dengan maksud mencari yang performanya lebih baik. Namun ada batasan dalam percobaan kali ini yaitu, jika menggunakan permainan 2048 dengan ukuran 4x4, akan memakan waktu yang sangat lama untuk diselesaikan. Untuk menyederhanakan permasalahan, digunakan permainan 2048 dengan ukuran 2x2 dengan nilai maksimum yang dapat dicapai adalah 16.



Gambar 8. Permainan 2048 berukuran 2x2

(sumber: dokumentasi penulis)

Permainan ini disimpan dalam bentuk matriks 2x2 sehingga mudah untuk diiterasi. Graf yang akan digunakan adalah graf dinamis yang akan dibangun bersamaan dengan proses algoritma yang dibuat dari state-state keadaan matriks sekarang.

Berikut ini adalah langkah proses pengerjaan dengan algoritma BFS:

1. Dibuat sebuah game dengan GUI agar terlihat hasil matriksnya
2. Ambil matriksnya dan masukan ke dalam suatu barisan
3. Keluarkan *state* matriks paling depan dari barisan
4. Gerakan *state* tersebut ke keempat arah yang mungkin
5. Untuk setiap arah diperiksa apakah matriks berubah dari kondisi sebelumnya
6. Jika berpindah, maka untuk setiap tempat kosong di matriks tersebut dibentuk suatu *state* baru yang menggambarkan kondisi papan yang telah berubah dan telah bertambah nilai yang baru
7. Langkah yang diperlukan untuk mencapai *state* tersebut juga disimpan
8. Tambahkan *state* baru tersebut ke akhir barisan
9. Ulangi dari langkah 3 hingga ditemukan *state* yang memenuhi kondisi akhir yaitu mendapat nilai 16
10. Gerakan pertamanya diaplikasikan pada game sekarang
11. Ulangi dari langkah 2 hingga mencapai kondisi menang di game.

Adanya pengulangan pada langkah 11 dikarenakan hasil pemunculan kotak baru pada game sebenarnya adalah acak. yang kita simulasikan menggunakan algoritma BFS adalah mencari kemungkinan langkah terbaik sehingga solusinya akan mencapai tujuan.

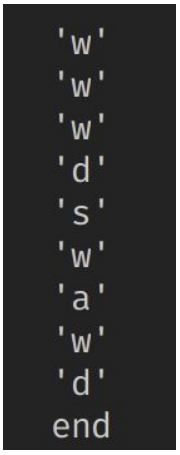
Dengan algoritma ini didapatkan data bahwa dibutuhkan 9 langkah minimum untuk mencapai tujuan yaitu mendapat nilai 16.

Gambar 9. Kondisi akhir permainan

(sumber: dokumentasi penulis)

Sedangkan untuk algoritma DFS, sama seperti langkah-langkah pada BFS namun mengganti barisan dengan stack, sehingga *state* yang baru masuk akan langsung keluar. Setiap *state* baru akan diletakan di paling belakang suatu list dan untuk setiap pengambilan akan mengambil dari paling belakang pula. Meskipun pada python tidak ada tipe data stack, dengan pendekatan ini maka list pun dapat berperan sebagai stack

Hasil yang didapat dari pengaplikasian algoritma DFS juga menyatakan bahwa diperlukan 9 langkah minimal untuk mencapai tujuan dari game ini.



Gambar 10. Langkah-langkah yang dilakukan untuk mendapatkan hasil yang optimal

(sumber: dokumentasi penulis)

Untuk setiap langkah yang diterapkan pada game, akan di print pada terminal langkah yang dijalankan. ‘w’ menyatakan berpindah ke atas, ‘d’ menyatakan berpindah ke kanan, ‘s’ menyatakan berpindah ke bawah, dan ‘a’ menyatakan berpindah ke kiri.

IV. Analisis

Dari hasil yang didapatkan setelah percobaan, dapat dilihat bahwa hasil yang diperlukan dari algoritma BFS dan DFS adalah sama yaitu 9. Namun seharusnya hal ini tidak terjadi, karena algoritma yang diaplikasikan berbeda.

Kesamaan ini diakibatkan oleh papan permainan yang terlalu kecil yaitu 2x2, sehingga *state* yang dibangun juga terbatas dan kebetulan DFS dan BFS mendapatkan hasil yang sama. Karena langkah yang diambil untuk diterapkan pada game adalah langkah yang pertama kali berhasil menjadikan *state* awal menjadi *state* akhir. Bukan berarti merupakan langkah terbaik. namun karena jumlah langkah yang terbatas hanya sekedar 4 langkah maka kemungkinan benarnya sangat tinggi.

Untuk membuktikan perbedaan antara DFS dan BFS dapat dilakukan dengan menambah ukuran kotak, misalnya menjadi 3x3 atau 4x4. Namun penambahan ukuran tersebut ternyata sangat berpengaruh pada waktu eksekusi program sehingga tidak dapat ditampilkan pada laporan ini. Selain itu, nilai akhir yang harus dicapai oleh program adalah 512. Nilai tersebut tidaklah kecil dan banyak sekali langkah yang dapat diambil untuk mendapatkan nilai tersebut.

Untuk percobaan papan dengan ukuran 3x3, setelah program dijalankan selama 30 menit lebih, program masih belum berhasil menghasilkan satu langkah pun yang dikarenakan kemungkinan yang sangat banyak.

Salah satu solusi yang dapat diterapkan untuk memecahkan masalah ini adalah dengan algoritma IDS atau iterative deepening search. Sebelum membahas mengenai IDS ada baiknya kita mengetahui mengenai DLS, Depth-Limited Search. Algoritma DLS adalah pengembangan dari algoritma BFS dengan menambahkan batasan kedalaman simpul yang akan dibangun. Jika suatu simpul sudah melebihi batas level tertentu, maka akan dianggap mati dan tidak dapat dikembangkan lebih lanjut. Kembali ke IDS, IDS adalah algoritma yang memanfaatkan DLS dengan melakukan iterasi terhadap level kedalaman yang diperbolehkan. Misalnya mulai dari level 1, 2, 3, dst hingga ditemukan solusi atau memang ingin dihentikan.

Penerapan algoritma IDS dapat menyelesaikan masalah yang ada karena, pada level awal, sebenarnya tidak masalah langkah mana yang diambil karena tempat yang luas dan kemungkinan yang banyak, sehingga semua langkah adalah langkah yang benar. Jika tidak dibatasi pencariannya maka akan menjadi sia-sia. Yang seharusnya membutuhkan perhitungan yang lebih detail adalah saat sudah mencapai titik akhir saat kurang dari 50% area permainan sudah terisi, karena jika hanya menggunakan perhitungan yang dangkal maka ada kemungkinan jalur yang diambil dapat menyebabkan kekalahan.

Penerapan IDS ini dapat berdasarkan basis waktu, misalnya untuk kondisi awal perhitungan hanya memerlukan 2 detik saja. Maka setelah 2 detik, akan diperiksa level DFS yang sudah terbentuk dengang IDS tersebut lalu dibandingkan antar *state* dan dipilih state yang menghasilkan skor paling besar. Karena semakin besar skornya maka akan semakin dekat pula *state* tersebut dengan tujuan.

Metode ini juga dapat diterapkan pada BFS untuk membatasi jumlah level yang akan dibangun sehingga dapat mengurangi waktu komputasi

VII. UCAPAN TERIMA KASIH

Pertama-tama, penulis ingin mengucapkan syukur dan terima kasih kepada Tuhan yang telah memberikan waktu dan kesempatan sehingga penulis dapat membuat makalah ini. Selain itu penulis ingin mengucapkan terima kasih banyak kepada dosen pembimbing mata kuliah Strategi Algoritma, Bapak Rinaldi Munir, yang telah bersedia mengajar dan membantu penulis hingga dapat dihasilkan makalah ini. Banyak sekali informasi dan pelajaran berharga yang dapat diambil dari ajaran Bapak Rinaldi ini. Sehingga saya merasa sangat beruntung dapat diajar oleh Bapak.

REFERENCES

[1] https://translate.google.com/#view=home&op=translate&sl=en&tl=id&text=algorithm

[2] https://www.hackerearth.com/practice/algorithms/graphs/breadth-first-search/tutorial/

[3] https://www.latimes.com/business/technology/la-fi-tn-2048-hit-game-creator-gabriele-cirulli-20140327-story.html#axzz2yiskNuep

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 2012

ttd

Johanes 13517012