Penerapan strategi BFS untuk menyelesaikan permainan Unblock Me beserta perbandingannya dengan DFS dan Branch and Bound

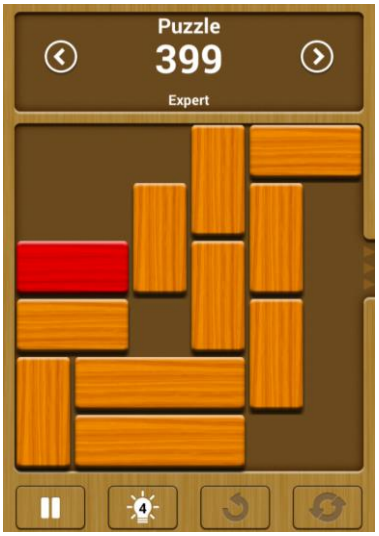
Eric 13512021
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13512021@std.stei.itb.ac.id

***Abstrak*—Makalah ini menjabarkan tentang cara untuk menerapkan strategi pencarian graf BFS untuk menyelesaikan permainan Unblock me. Makalah ini juga membandingkan strategi BFS dengan strategi DFS dan Branch and Bound. Permainan Unblock me adalah permainan sederhana yang memiliki banyak versi. Tujuan dari permainan ini adalah mengeluarkan balok merah yang dihalangi banyak balok lainnya. Setelah memperkenalkan permainan ini kepada pembaca, makalah ini membahas cara mengimplementasikan strategi BFS pada permainan ini unuk mencari solusi optimal untuk konfigurasi apapun.**

***Kata Kunci*—BFS, Branch and Bound, DFS, perbandingan, Unblock me.**

I. PENDAHULUAN

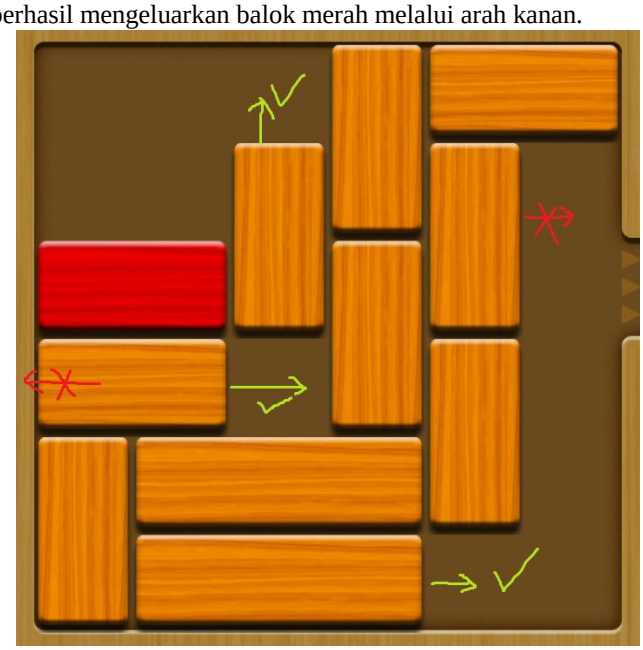
Permainan Unblock me adalah sebuah permainan *genre* teka-teki yang cukup menarik. Tujuan dari permainan ini adalah untuk membebaskan sebuah balok dari sebuah tempat dimana terdapat banyak balok-balok lain yang menghalangi. Balok yang berwarna merah adalah balok yang hendak dibebaskan, balok yang berwarna seperti balok kayu biasa adalah balok yang menghalangi kita dalam membebaskan balok merah tersebut.



Gambar 1: Permainan Unblock me sedang berlangsung

Seluruh balok dalam permainan ini berbentuk N × 1, dimana N dapat berupa 2 atau 3. Balok merah selalu berkuran 2 × 1. Ukuran tempat juga sudah tetap yaitu berukuran 6 × 6.

Balok-balok tersebut hanya dapat digerakkan sejajar dengan balok. Balok juga hanya dapat digerakkan jika tidak ada balok lain yang menghalangi atau tidak terdapat dinding yang menghalangi. Permainan selesai ketika kita berhasil mengeluarkan balok merah melalui arah kanan.



Gambar 2: Contoh gerakan yang valid dalam panah hijau, yang invalid dalam panah merah.

Terdapat banyak variasi dari game ini, biasanya balok-balok tersebut diganti dengan mobil. Variasi yang dapat terjadi lainnya adalah ukuran tempat diperbesar, dan lain sebagainya.

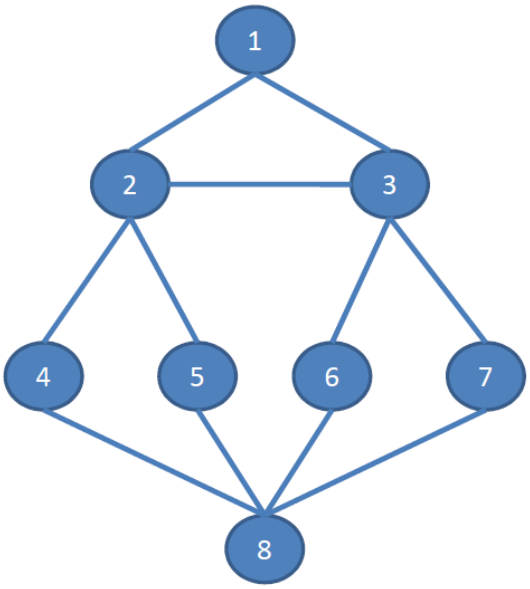
II. DASAR TEORI

A. Breadth-First Search

Pencarian melebar atau *Breadth-First Search* (BFS) adalah sebuah strategi pencarian pada graf yang bekerja dengan cara mengunjungi simpul secara melebar. Strategi ini akan mengunjungi simpul awal *v*, kemudian "mengingat" simpul yang bertetangga dengan *v* yang

belum dikunjungi. Setelah itu untuk setiap simpul yang diingat, dikunjungi satu per satu dan sambil "mengingat" simpul yang bertentangga dengannya yang belum dikunjungi. Pemilihan simpul dari daftar simpul yang diingat adalah dengan meprioritaskan simpul yang lebih dekat ke simpul asal *v*.

Pada ilustrasi di bawah ini nomor simpul merupakan urutan pencarian dengan strategi BFS.



Gambar 3: Ilustrasi Pencarian Graf dengan BFS
(Terakhir diakses pada tanggal 18 Mei 2014, sumber: http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2013-2014-genap/BFS%20dan%20DFS.pdf)

Secara algoritmik strategi BFS dapat dituliskan dalam empat tahap:

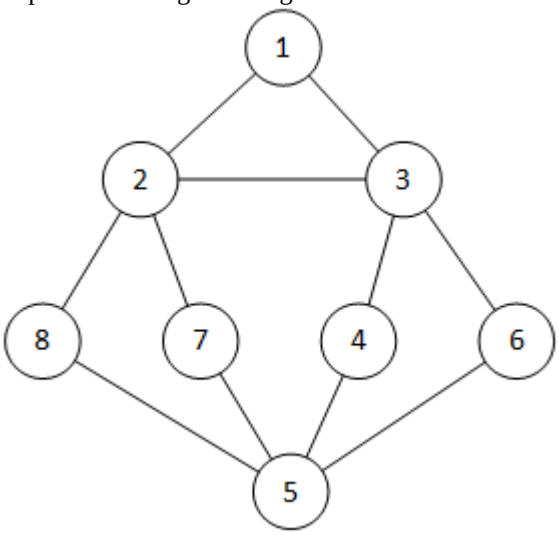
1. Kunjungi simpul *v*.
2. Catat semua simpul yang yang bertentangga dengan simpul *v* dengan cara memasukkannya ke dalam *Queue*.
3. Ambil simpul terdepan dari *Queue* dan masukkan ke dalam variabel *w*. Kunjungi simpul *w* jika belum pernah dikunjungi sebelumnya, tambahkan seluruh tetangga *w* ke dalam *Queue*.
4. Ulangi langkah ke-3 sampai *Queue* kosong atau solusi ditemukan.

Masalah yang sering dihadapi oleh penggunaan strategi BFS adalah penggunaan memori yang besar. Hal ini dikarenakan BFS perlu mengingat seluruh simpul yang bertetangga dengan simpul yang dikunjungi.

B. Depth First Search

Pencarian mendalam atau *Depth First Search* (DFS) adalah sebuah strategi pencarian pada graf yang bekerja dengan mengunjungi simpul secara mendalam. Strategi ini akan mengunjungi simpul awal *v*, kemudian langsung mengunjungi simpul yang bertetangga dengan *v* yang belum pernah dikunjungi. Demikian seterusnya sampai dicapai suatu simpul dimana tidak ada lagi simpul tetangga yang belum dikunjungi. Jika dicapai simpul tersebut, DFS akan melakukan runut-balik (*backtracking*) ke simpul sebelumnya dan melanjutkan pencarian secara mendalam. Pencarian dihentikan ketika solusi sudah tercapai atau terjadi runut-balik hingga ke simpul awal *v*, yang artinya tidak terdapat solusi.

Pada ilustrasi di bawah ini nomor simpul merupakan urutan pencarian dengan strategi DFS.



Gambar 4: Ilustrasi pencarian graf dengan DFS

Secara algoritmik strategi DFS dapat dituliskan dalam lima tahap:

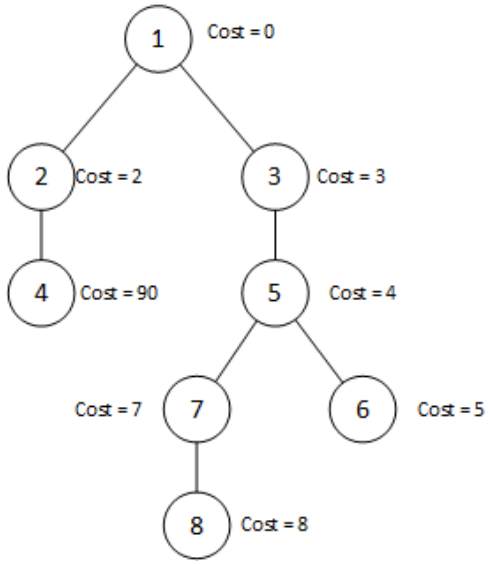
1. Kunjungi simpul *v*.
2. Kunjungi simpul *w* yang bertetangga dengan simpul *v*.
3. Ulangi DFS mulai dari simpul *w*.
4. Ketika mencapai simpul *u* sedemikian sehingga semua simpul yang bertentangga dengannya telah dikunjungi, pencarian dirunut-balik (*backtrack*) ke simpul terakhir yang dikunjungi sebelumnya.
5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

Masalah yang sering dihadapi oleh penggunaan strategi DFS adalah penelusuran mungkin dilakukan sampai ke ujung graf, padahal solusi dekat dengan simpul asal. Bahkan kadang-kadang "ujung graf" tidak diketahui. Contoh: Penelusuran link website pada Web Crawler dengan DFS.

C. Branch and Bound

Branch and Bound adalah strategi pencarian graf yang mirip seperti BFS tetapi menambahkan fungsi *cost* sehingga pada saat mengambil elemen dari *Queue* tidak berdasarkan urutan saja, tetapi juga berdasarkan nilai dari simpul tersebut. Hal ini membuat *Queue* yang dipakai di strategi B&B berupa *Priority Queue*.

Pada ilustrasi di bawah ini nomor simpul merupakan urutan penelusuran dengan menggunakan strategi B&B. Penelusuran dilakukan dengan memprioritaskan *cost* terendah.



Gambar 5: Ilustrasi pencarian graf dengan B&B

Secara algoritmik strategi B&B dapat dituliskan dalam empat tahap:

1. Kunjungi simpul *v*.
2. Catat semua simpul yang yang bertentangga dengan simpul *v* dengan cara memasukkannya ke dalam *Priority Queue*.
3. Ambil simpul terdepan dari *Priority Queue* dan masukkan ke dalam variabel *w*. Kunjungi simpul *w* jika belum pernah dikunjungi sebelumnya, tambahkan seluruh tetangga *w* ke dalam *Priority Queue*.
4. Ulangi langkah ke-3 sampai *Priority Queue* kosong atau solusi ditemukan.

Pencarian menggunakan B&B umumnya lebih baik daripada DFS maupun BFS. Akan tetapi, penentuan fungsi *cost* sangat memengaruhi performa pencarian B&B. Menentukan fungsi *cost* yang baik tidak selalu dapat dilakukan.

III. IMPLEMENTASI

Pada bagian ini akan dibahas strategi BFS yang akan digunakan untuk mencari solusi untuk permainan teka-teki Unblock me. Selain itu, kita juga akan membandingkannya dengan strategi DFS dan B&B.

Sebelum kita dapat melakukan pencarian solusi dengan graf, kita harus dapat memodelkan permainan ini menjadi model graf terlebih dahulu. Permainan ini dapat dimodelkan menjadi matriks berukuran 6 × 6. Dimana tempat kosong ditandai dengan angka 0, balok merah (balok yang ingin dikeluarkan) ditandai dengan angka 1, dan balok lainnya ditandai dengan angka 2 s/d 18. Terdapat batasan pada banyak balok yaitu sebanyak 18. Dengan asumsi kasus terburuk, setiap balok merupakan balok horizontal berukuran 2 × 1 maka banyak balok maksimal adalah sebanyak luas permainan dibagi luas balok (sebanyak 36/2 atau 18) balok, termasuk dengan balok yang ingin dikeluarkan.

Sebagai contoh, untuk merepresentasi keadaan permainan seperti pada gambar 1, kita dapat menggunakan matriks:

0	0	0	2	3	3
0	0	4	2	5	0
1	1	4	6	5	0
7	7	0	6	8	0
9	10	10	10	8	0
9	11	11	11	0	0

Supaya kita dapat menghasilkan kembali langkah-langkah yang telah dilakukan dari awal permainan, kita perlu mencatat langkah-langkah yang digunakan untuk mencapai matriks kita. Suatu simpul pada graf menyimpan matriks model permainan saat ini beserta langkah-langkah yang telah digunakan. Berikut ini struktur simpul graf yang lengkap:

```
type Matrix: array[1..6][a..6] of
integer
```

```
struct Node
state: Matrix
langkah: list of string
```

A. Perbandingan dengan strategi DFS

DFS kurang praktis untuk digunakan dalam mencari solusi untuk permainan ini, hal ini dikarenakan pencarian dengan DFS sering menghasilkan solusi yang tidak optimal. Solusi yang tidak optimal maksudnya adalah solusi yang banyak langkahnya lebih banyak daripada solusi optimal. Solusi optimal sendiri adalah solusi yang memerlukan langkah paling sedikit.

Keunggulan menggunakan BFS daripada DFS dalam permainan ini adalah selalu ditemukan solusi optimal.

B. Pebandingan dengan strategi B&B

B&B juga kurang praktis untuk digunakan dalam mencari solusi untuk permainan ini. Hal ini dikarenakan fungsi *cost* yang pasti sangat sulit diperoleh. Jika kita tidak memakai fungsi *cost* yang pasti untuk persoalan ini, maka konsekuensinya adalah solusi yang dihasilkan belum tentu optimal. Peningkatan performa untuk penggunaan strategi B&B juga tidak terlalu signifikan untuk permainan ini. Penambahan fungsi *cost* pada setiap simpul menambah kerumitan program, *Queue* juga harus diganti dengan *Priority Queue* sehingga pada akhirnya peningkatan performa untuk penggunaan strategi B&B tidak signifikan. Bahkan mungkin dengan menggunakan strategi BFS bisa menghasilkan performa yang lebih baik.

Kelebihan menggunakan BFS daripada B&B dalam permainan ini adalah program lebih sederhana, selalu mendapatkan solusi optimal, dan mungkin memiliki performa yang lebih baik daripada B&B.

C. Implementasi strategi BFS

Berikut ini pseudocode untuk implementasi pencarian solusi menggunakan strategi BFS:

```
function BFS(initial: Matrix) → Node
Node el
```

```
el.state ← initial
el.langkah ← []

Node ans ← el
Queue of Node q
q.push(el)
Boolean marker ← true
while not q.empty() and marker do
  Node front ← q.pop()
  if front.state belum pernah
dikunjungi then
    if permainan sudah selesai then
      ans ← front
      marker ← false
    else
      foreach langkah yang mungkin do
        el.state ← Matrix baru setelah
melakukan langkah
        el.langkah ← front.langkah +
langkah
        q.push(el)
      end for
    end if
  end if
end while
return ans
```

Algoritma pseudocode di atas dapat dituliskan menjadi:

1. Menambahkan simpul asal ke dalam *Queue*.
2. Jika *Queue* sudah kosong atau sudah ditemukan solusi lanjut ke langkah 9.
3. Ambil simpul terdepan dari *Queue* dan masukkan ke variabel *front*.
4. Jika matriks dari simpul *front* sudah pernah dikunjungi sebelumnya, kembali ke langkah 2.
5. Jika permainan sudah selesai, simpan simpul ini sebagai jawaban dan nyatakan sudah ditemukan solusi kemudian ke langkah 2.
6. Bangkitkan semua kemungkinan langkah dari simpul saat ini (simpul *front*).
7. Untuk setiap kemungkinan langkah dari simpul saat ini, buatlah sebuah matriks yang menyatakan kondisi permainan jika seandainya langkah tersebut dilakukan. Matriks ini langsung dimasukkan ke sebuah Simpul el. Selain itu, masukkan juga langkah yang dilakukan untuk menghasilkan matriks tersebut dengan menambahkan langkah tersebut dengan langkah-langkah sebelumnya ke da dalam Simpul el. Simpul ini kemudian di masukkan ke dalam *Queue*.
8. Kembali ke langkah 2
9. Kembalikan jawaban ke fungsi pemanggil.

Untuk menandai apakah suatu matriks keadaan permainan sudah pernah dikunjungi atau belum bisa digunakan sebuah *Map* yang memetakan Matriks ke sebuah *Boolean*. *Map* memetakan Matriks ke *Boolean* dengan menyimpan Elemen dalam bentuk pasangan Matriks dan *Boolean*. Kita dapat mencek apakah suatu Matriks sudah dipetakan ke sebuah *Boolean* dengan mudah. Jika sudah terdapat Pemeteaan dari Matriks ke *Boolean* maka Matriks tersebut sudah dikunjungi. *Boolean* dalam pemakaian ini selalu bernilai *true* jika ada.

Selain cara di atas, cara lain untuk mengetahui apakah Matriks sudah pernah dikunjungi adalah dengan menggunakan *hash table* atau *set*. Pemakaian *hash table* akan membuat rumit persoalan karena harus diimplementasikan fungsi *hash* terlebih dahulu, sedangkan pemakaian *set* hanya dapat dilakukan untuk segelintir bahasa pemrograman.

Setelah dihasilkan Node yang merupakan solusi dari permainan, kita dapat menggunakan langkah-langkah dari Node tersebut untuk mengkonstruksi ulang transisi matriks permainan dari awal sampai ditemukannya solusi.

IV. HASIL EKSPERIMEN

Penulis sudah membuat program untuk mencari solusi permainan Unblock me dengan menggunakan strategi pencarian *Breadth-First Search*. Penulis membuat program tersebut dalam bahasa pemrograman JAVA.

Program menerima masukan berupa konfigurasi awal permainan dalam bentuk matriks 6 × 6. Matriks ini sudah dijelaskan di bagian sebelumnya. Untuk memisahkan nilai dari matriks dapat dipisahkan dengan *white space* (spasi, *tab*, bahkan *enter*). Program ini akan mencari sebuah solusi optimal dari konfigurasi awal tersebut. Perhatikan bahwa akhir permainan ditandai dengan balok merah bisa keluar dari papan permainan dari sebelah kanan.

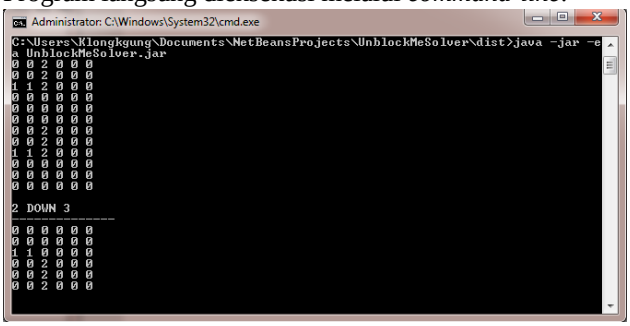
Program ini akan menghasilkan matriks-matriks yang merupakan transisi dari matirks awal hingga mencapai solusi beserta langkahnya ke layar. Langkah yang dihasilkan dari program ini berbentuk <nomor balok> <ARAH> <perpindahan>.

Program yang dibuat penulis memiliki performa yang cukup baik, pencarian solusi yang dilakukan tidak pernah memakan waktu lebih dari 1 detik. Program juga hanya memakai 10 *MegaByte* Memori RAM pada rata-rata kasus pemakaian program.

TestCase I

Testcase ini didesain hanya untuk menunjukkan bagaimana *output* program yang sesungguhnya.

Program langsung dieksekusi melalui *command-line*.



Gambar 6: Pemakaian program untuk *testcase* 1

TestCase II

TestCase ini merupakan *testcase* nyata permainan Unblock me FREE pada Android, yaitu *puzzle* nomor 399 pada level *expert*.

Program dieksekusi dengan menggunakan file kemudian hasilnya juga disimpan ke dalam file.

Persiapkan sebuah file yaitu input.txt pada folder dimana program berada, yang berisi:

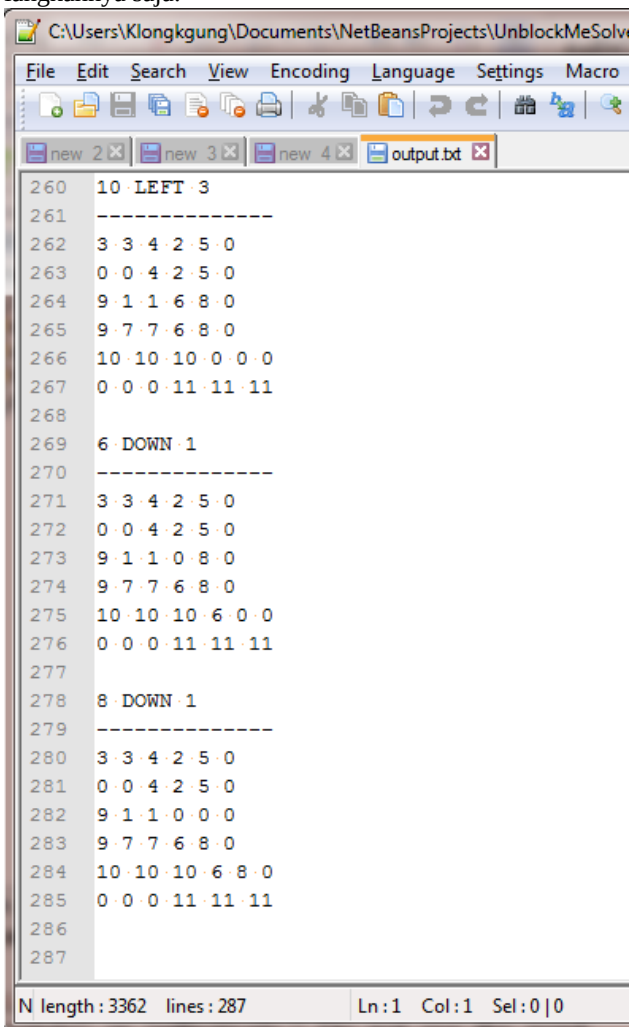
File: input.txt

```
0       0       0       2       3       3
0       0       4       2       5       0
1       1       4       6       5       0
7       7       0       6       8       0
9       10      10      10      8       0
9       11      11      11      0       0
```

Eksekusi Program dengan menggunakan perintah *command-line*:

```
java -jar -ea UnblockMeSolver.jar < input.txt > output.txt
```

Karena *output* yang dihasilkan cukup besar (250+ baris), penulis memutuskan untuk memberikan langkahnya saja.



Gambar 7: Sebagian dari file output.txt yang sesungguhnya

Output yang dipersingkat:

```
7 RIGHT 1
11 RIGHT 2
4 UP 1
1 RIGHT 1
9 UP 4
1 LEFT 1
7 LEFT 1
10 LEFT 1
4 DOWN 1
6 DOWN 1
2 DOWN 1
3 LEFT 3
2 UP 1
6 UP 1
5 UP 1
8 UP 1
10 RIGHT 3
4 DOWN 3
1 RIGHT 1
7 RIGHT 1
9 DOWN 4
3 LEFT 1
1 LEFT 1
7 LEFT 1
4 UP 4
1 RIGHT 1
7 RIGHT 1
9 UP 2
10 LEFT 3
6 DOWN 1
8 DOWN 1
```

V. KESIMPULAN

Strategi pencarian graf BFS merupakan strategi yang sangat baik digunakan untuk menyelesaikan masalah Unblock me. Strategi BFS lebih baik daripada strategi DFS, dan mungkin lebih baik daripada strategi B&B. Solusi yang dihasilkan juga merupakan solusi optimal.

VII. UCAPAN TERIMA KASIH

Penulis ingin mengucapkan terima kasih kepada Tuhan atas segala yang diberikan-Nya sehingga makalah ini dapat selesai tepat waktu. Penulis juga ingin mengucapkan terima kasih kepada Orang tuanya yang telah membesarkan beliau sehingga dapat menjadi mahasiswa ITB. Penulis juga ingin mengucapkan terima kasih kepada dosen mata kuliah IF-2211 Dr. Ir. Rinaldi Munir, MT. dan Masayu Leylia Khodra, ST., MT. atas ilmu yang telah dibagikannya melalui kuliah sehingga penulis mampu menyelesaikan makalah ini.

REFERENSI

[1] Munir, Rinaldi. Diktat Kuliah IF2211 Strategi Algoritma. 2009. Bandung: Teknik Informatika ITB.

[2] game Unblock me FREE.

[3] http://play.google.com/store/apps/details?id=com.kiragames.unblockmefree diakses tanggal 18 Mei 2014.
[4] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2013-2014-genap/BFS%20dan%20DFS.pdf diakses tanggal 18 Mei 2014.
[5] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2013-2014-genap/Branch%20&%20Bound%20(2014).pptx diakses tangal 18 Mei 2014.

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, 18 Mei 2014

Eric - 13512021