Analisis Penerapan Algoritma Runut Balik (*Backtracking*) untuk Menyelesaikan Permainan *Minesweeper*

Hansel Valentino Tanoto – 13520046
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13520046@std.stei.itb.ac.id

***Abstrak*—Permainan *minesweeper* merupakan salah satu permainan teka-teki yang cukup populer pada sistem operasi *Windows*. Permainan ini bertujuan untuk membersihkan (*clearing*) area papan permainan tanpa terkena bom. Terdapat banyak algoritma yang bisa digunakan untuk menyelesaikan permainan *minesweeper* ini salah satunya algoritma *backtracking*. Penerapan algoritma *brute force* menghasilkan kompleksitas algoritma sebesar O(n.n!/(n-d)!) sedangkan algoritma *backtracking* menghasilkan kompleksitas algoritma sebesar $O(2^n.n)$ dengan n dan d berturut-turut jumlah kotak dan ranjau pada papan permainan.**

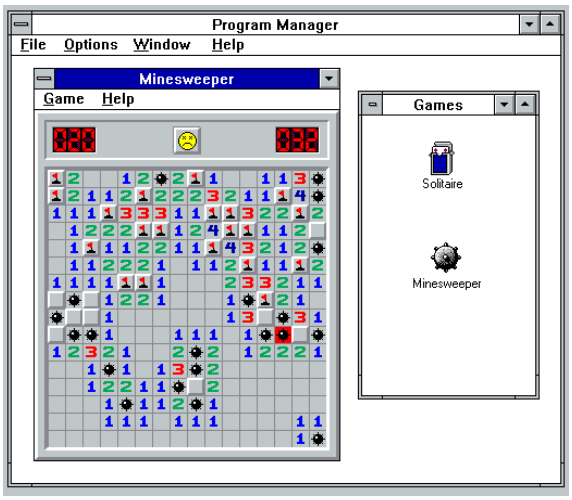
Kata Kunci—minesweeper, algoritma, brute force, exhaustive search, backtracking,

I. PENDAHULUAN

Minesweeper adalah sebuah permainan komputer bertema teka-teki (*puzzle*) yang cukup populer di kalangan pengguna sistem operasi Windows pada sekitar tahun 1990-an sampai 2000-an. Permainan *minesweeper* hanya dimainkan dengan seorang pemain saja pada sebuah papan permainan yang tersusun atas kotak-kotak kecil yang beberapa di antaranya mengandung bom atau ranjau. Jadi, tujuan dari permainan ini adalah untuk membersihkan arena dengan cara membuka semua kotak yang aman yaitu kotak yang kosong dan tidak membuka kotak yang mengandung bom atau ranjau (*mine*). Jjika pemain terkena satu bom saja, maka permainan akan langsung selesai (*game over*) dan pemain pun dinyatakan kalah. Jadi permainan ini menekankan pada logika dan probabilitas berdasarkan informasi yang terdapat pada kotak-kotak di papan permainan untuk dapat memilih kotak yang tepat.

Sebenarnya permainan ini sudah diciptakan sejak tahun 1960-an dan juga diimplementasikan pada sistem operasi lainnya dengan nama yang berbeda seperti GNOME mines di GNOME, MineHunt di Palm OS, dan lain-lain. Namun, seperti yang dikatakan di awal, salah satu versi yang paling dikenal dari pernainan ini adalah versi yang dikeluarkan oleh Windows terutama untuk versi pada sistem operasi Windows 3.1 ke atas.

Seiring waktu, permainan *minesweeper* ini sudah mulai berkembang dan bervariasi baik dalam hal *platform* maupun *gameplay*-nya. Contoh variasi *gameplay* tersebut misalnya pada aplikasi *game mobile* bernama Plato yang membuat *minesweeper* bisa dimainkan secara *multiplayer* dan *online* dengan pemenang ditentukan berdasarkan pemain yang berhasil mengumpulkan poin terbanyak hingga permainan berakhir. Selain itu ada juga variasi permainan yang memperhitungkan waktu penyelesaian game sebagai poin tambah. Oleh karena itu, muncullah berbagai algortima yang dikembangkan untuk menyelesaikan permainan *minesweeper* ini mulai dari sesederhana algoritma *brute force*, hingga algoritma DFS, BFS, *backtracking*, dan lain-lain. Namun, sebenarnya belum ada algoritma yang benar-benar efisien dalam menyelesaikan permainan ini karena persoalan *minesweeper* sendiri termasuk ke dalam persoalan NP (*Non-Deterministic Polinomial*) yang artinya termasuk persoalan matematika yang cukup sulit diselesaikan.



Gambar 1. Ilustrasi *minesweeper* versi Windows 3.1

(Sumber: https://minesweepergame.com/download/windows-31-minesweeper.png)

Pada makalah ini, akan dibahas lebih lanjut mengenai penerapan algoritma *backtracking* untuk menyelesaikan

permainan *minesweeper* dan dibandingkan dengan algoritma *brute force* sebagai algoritma yang paling dasar dan sederhana. Algoritma *backtracking* sendiri adalah algoritma yang mirip dengan algoritma DFS (*Depth First Search*), dimana algoritma ini dilengkapi dengan kemampuan untuk merunut balik (*backtrack*) apabila ditemukan simpul status yang tidak mungkin bisa menghasilkan solusi.

II. Landasan Teori

A. Algoritma Brute Force

Algoritma *brute force* adalah pendekatan yang paling sederhana, *straightforward*, dan mudah dipahami dalam menyelesaikan suatu permasalahan. Penyelesaian masalah menggunakan algoritma ini didasarkan secara langsung dari *problem statement* dan konsep yang dilibatkan.

Salah satu teknik yang umum digunakan adalah Teknik *exhaustive search*. *Exhaustive search* merupakan teknik yang digunaka untuk menyelesaikan permasalahan-permasalahan yang bersifat kombinatorik (kombinasi, permutasi, dan himpunan bagian). Cara kerja teknik ini adalah dengan mengenumerasi dan mengevaluasi semua kemungkinan solusi sambil menyimpan solusi terbaik sejauh ini hingga pencarian berakhir.

Karena kesederhanannya itu, algoritma *brute force* memiliki beberapa kelebihan, yaitu:

1. Hampir semua permasalahan pasti bisa diselesaikan menggunakan algoritma *brute force* sehingga algoritma ini sering digunakan sebagai basis pembanding untuk algoritma lainnya.
2. Algortima *brute force* sangat sederhana dan mudah dipahami.
3. Algoritma *brute force* dapat menghasilkan algoritma yang layak (cukup baik) untuk beberapa masalah penting, misalnya pencarian (*searching*), pengurutan (*sorting*), perkalian matriks, dan lain-lain.
4. Algoritma *brute force* dapat menghasilkan algoritma yang baik dan baku untuk beberapa permasalahan, seperti menentukan nilai ekstrim (maksimum atau minimum).

Di samping kelebihannya tersebut, terdapat beberapa kelemahan dari algoritma *brute force*, yaitu:

1. Algoritma *brute force* jarang dan sulit menghasilkan algoritma yang efektif dan efisien
2. Secara umum, algoritma *brute force* tergolong algoritma yang lambat dan memerlukan banyak ruang *memory*. Namun, algoritma *brute force* masih cukup baik untuk persoalan yang berukuran kecil.
3. Karena *straightforward*, algoritma *brute force* tidak sekonstruktif algoritma lainnya.

Namun demikian, efisiensi algoritma *brute force* dapat diperbaiki dengan menggunakan teknik *heuristic*, yaitu dengan cara mengeliminasi kemungkinan-kemungkinan solusi yang sudah diketahui tidak layak agar tidak perlu mengenumerasi semua kemungkinan solusi. Teknik heuristic ini merupakan pendekatan yang informal sehingga terkadang tidak bisa dibuktikan secara matematis.

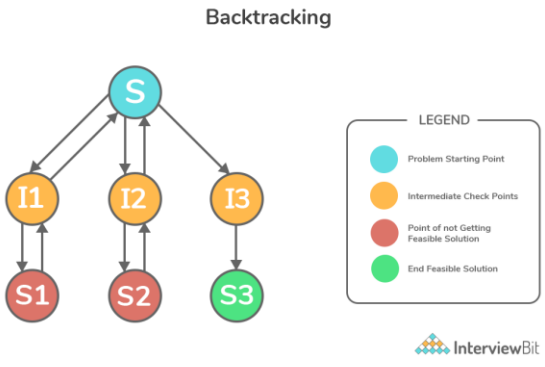
B. Algoritma Backtracking

Algoritma runut balik (*backtracking*) dapat dikatakan merupakan algoritma yang merupakan perbaikan dari *exhaustive search* dimana pada algoritma backtracking, hanya pilihan yang mengarah ke solusi yang akan dievaluasi lebih lanjut, sedangkan pilihan / simpul yang tidak mengarah ke solusi akan dipangkas (tidak dievaluasi lebih lanjut). Algoritma ini pertama kali diperkenalkan oleh D. H. Lehmer pad tahun 1950. Kemudian, dalam perkembangannya, Golomb, Baumert, dan Rwalker menyajikan uraian umum dan penerapan dari algoritma ini di berbagai persoalan.

Terdapat tuga buah properti umum dari algoritma runut balik, yaitu:

1. Solusi yang dinyatakan dalam *tuple* $X = (x_1, x_2, \ldots, x_n)$ dengan $x_i \in S_i$ dan biasanya $S_1 = S_2 = \ldots = S_n$.
2. Fungsi pembangkit yang umumnya dinyatakan dengan predikat T(x[1], x[2], …, x[k-1]) untuk membangkitkan nilai x_k yang berperan sebagai komponen vektor solusi di atas.
3. Fungsi pembatas (*bounding function*) yang umumnya dinyatakan sebagai predikat $B(x_1, x_2, \ldots, x_n)$. Fungsi pembatas akan bernilai *true* jika parameternya tidak melanggar *constraint* atau mengarah ke solusi. Jika bernilai *false*, maka pembangkitan nilai selanjutnya tidak dilakukan dan vektor solusi saat ini dibuang.

Dalam proses algoritma *backtracking* tersebut, akan dihasilkan banyak kemungkinan vektor solusi. Semua kemungkinan vektor solusi tersebut disebut ruang solusi yang bisa distrukturisasi menjadi sebuah pohon ruang status (*state space tree*) dimana simpulnya menyatakan status dari persoalan dan sisinya menyatakan nilai x_i. Jadi, pada algoritma *backtracking*, pencarian solusi dilakukan dengan membangkitkan simpul-simpul mengikuti aturan DFS (*Depth First Search*) sehingga membentuk lintasan dari akar ke simpul daun yang akan menyatakan kemungkinan solusi tersebut.



Gambar 2. Ilustrasi pohon ruang status pada algoritma *backtracking*

(Sumber: https://ibpublicimages.s3-us-west-2.amazonaws.com/tutorial/backtracking1.png)

Dalam pohon pencarian algoritma *backtracking*, terdapat beberapa istilah simpul, yaitu:

1. Simpul hidup (*live node*), yaitu simpul yang sudah dibangkitkan.
2. Simpul ekspan (*expand node*), yaitu simpul hidup yang sedang diekspansi.
3. Simpul mati (*dead node*), yaitu simpul ekspan yang dipangkas (di-*pruning*) karena tidak mengarah ke solusi. Simpul mati dihasilkan dari hasil penerapan fungsi pembatas.
4. Simpul hasil (*goal node*), yaitu simpul yang merupakan solusi dari persoalan yang ingin dipecahkan.

Jika ditemukan simpul mati, maka pencarian dilanjutkan kembali ke simpul *parent*-nya, lalu dilanjutkan ke simpul anaknya yang lain. Inilah perbedaan algoritma ini dengan algoritma *brute force* yang akan mencari semua kemungkinan simpul status.

Berikut ini adalah gambar skema algoritma *backtracking* secara umum:

```
procedure RunutBalikR(input k : integer)
{Mencari semua solusi persoalan dengan metode runut-balik; skema rekursif
Masukan: k, yaitu indeks komponen vektor solusi, x[k]. Diasumsikan x[1], x[2], ..., x[k – 1] sudah
ditentukan nilainya.
Luaran: semua solusi x = (x[1], x[2], ..., x[n])
}
Algoritma:                                   T(x[1], x[2], ..., x[k – 1])
                                             membangkitkan nilai untuk x_k
  for setiap x[k] ∈ T(x[1], x[2], ..., x[k – 1]) do
    if B(x[1], x[2], ... ,x[k]) = true then //mengarah ke solusi
      if (x[1], x[2], ... , x[k]) adalah lintasan dari akar ke simpul solusi then
        write(x[1], x[2], ..., x[k])     { cetak solusi }
      endif
      if k < n then
        RunutBalikR(k+1)    { tentukan nilai untuk x[k+1]}
      endif
    endif
  endfor                                     Pemanggilan pertama kali: RunutBalikR(1)
```

Gambar 3. Tingkat kesulitan permainan *minesweeper*

(Sumber: http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian1.pdf)

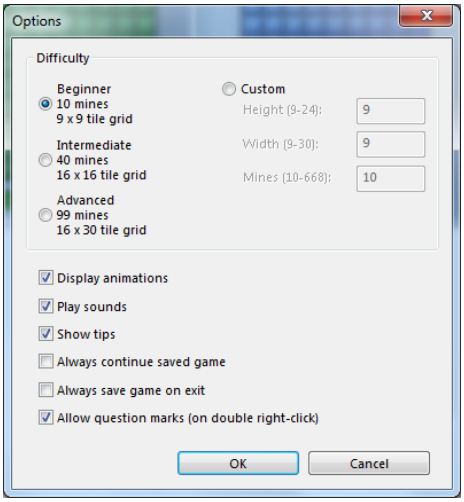
Secara umum, untuk pohon ruang status yang memiliki jumlah simpul 2^n atau n!, komplekstitas algoritma *backtracking* adalah $O(p(n)2^n)$ atau O(q(n)n!) dengan p(n) dan q(n) merupakan suatu fungsi polinom berderajat n yang menyatakan waktu komputasi setiap simpul dalam pohon ruang status.

C. *Permainan Minesweeper*

Minesweeper merupakan permainan bergenre teka-teki yang cukup populer pada sistem operasi Windows. Versi yang paling terkenal merupakan *minesweeper* pada versi Windows 3.1 yang dibuat oleh Robert Donner and Curt Johnson pada tahun 1990.

Permainan *minesweeper* memiliki tujuan untuk membersihkan atau membuka semua kotak yang diketahui kosong berdasarkan petunjuk-petunjuk yang ada dan menghindari untuk membuka kotak yang mengandung bom (*mines*). Ukuran dari papan permainan dan jumlah bom yang tersembunyi akan bergantung pada tingkat kesulitan yang dipilih. Secara umum, tingkat kesulitan pada permainan *minesweeper* adalah sebagai berikut:

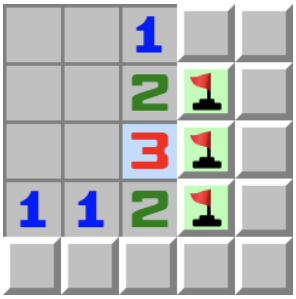
1. *Beginner*, papan permainan berukuran 9 x 9 = 81 kotak dengan bom sebanyak 10 buah
2. *Intermediate*, papan permainan berukuran 16 x 16 = 256 kotak dengan bom sebanyak 40 buah
3. *Advanced*, papan permainan berukuran 16 x 30 = 480 kotak dengan bom sebanyak 99 buah
4. *Custom*, dapat memilih sendiri ukuran papan dan banyaknya bom



Gambar 3. Tingkat kesulitan permainan *minesweeper*

(Sumber: https://www.kodyaz.com/images/games/windows-minesweeper/windows-minesweeper-game-options-for-children-in-windows7.png)

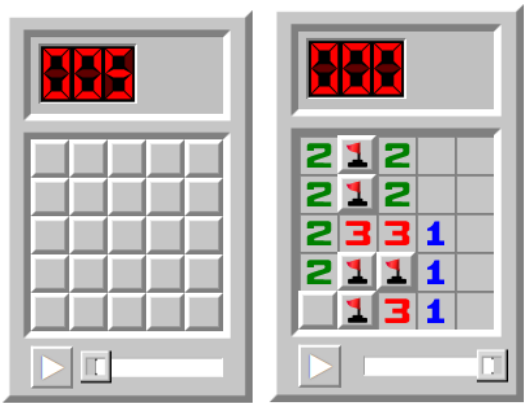
Setiap kotak dalam permainan menyimpan informasi berupa angka *integer* dengan *range* dari 0 sampai 8 yang menyatakan jumlah bom yang terdapat di sekeliling kotak tersebut. Informasi inilah yang digunakan untuk menentukan kotak mana yang tidak mengandung bom dan aman untuk dibuka. Sedangkan kotak yang mengandung bom dapat ditandai dengan sebuah bendera untuk membantu pemain mengingat kotak mana yang sudah diketahui mengandung bom.



Gambar 4. Ilustrasi penggunaan bendera untuk menandai kotak pada papan permainan

(Sumber: https://minesweeper.online/img/help/b1/2.png)

Pada gambar di atas, terlihat ada kotak yang berisi nomor 3 (kotak yang di-*highlight* biru) yang artinya di sekeliling kotak tersebut terdapat tiga buah bom. Dari delapan buah kotak di sekeliling kotak bernomor 3, lima buah di antaranya sudah terbuka dan tidak mengandung bom. Jadi, dapat dipastikan tiga kotak sisanya pasti memiliki bom sehingga ketiga kotak tersebut (kotak yang di-*highlight* hijau) dapat ditandai menggunakan bendera.



Gambar 5. Tampilan awal dan akhir permainan *minesweeper*

(Sumber: Dokumen Pribadi)

Cara bermain *minesweeper* secara singkat adalah sebagai berikut:

1. Permainan dimulai dengan papan permainan yang berisi kotak-kotak yang masih tertutup. Pemain dapat memilih 1 kotak untuk dipilih pertama kali. Pilihan kotak pertama ini bersifat *random* karena belum diketahui informasi apapun mengenai posisi bom.
2. Jika pilihan kotak pertama langsung berisi bom, maka game akan langsung berakhir dan artinya pemain kalah. Namun, jika tidak berisi bom, sebanyak 1 atau lebih kotak akan terbuka dan muncul angka yang mengindikasikan jumlah bom yang terdapat di sekeliling tiap kotak.
3. Apabila suatu kotak dicurigai mengandung bom, kotak tersebut dapat ditandai dengan lambang bendera.
4. Lanjutkan permainan dengan menandai kotak lain yang dicurigai memiliki bom atau membuka kotak yang sudah diyakini tidak mengandung bom.
5. Permainan akan selesai apabila semua kotak yang tidak mengandung bom sudah terbuka.

III. Pembahasan

Pada makalah ini, akan dijelaskan penggunaan algoritma *brute force* dan *backtracking* untuk menentukan susuan posisi bom pada papan permainan *minesweeper* bila diketahui angka-angka yang menyatakan jumlah bom yang terdapat disekeliling tiap kotak.

A. Penyelesaian Permainan Minesweeper dengan Algoritma Brute Force

Algoritma *brute force* yang digunakan berupa teknik *exhaustive search* dengan langkah-langkah sebagai berikut:

1. Mengenumerasi semua kemungkinan kombinasi lokasi bom dalam papan permainan. Persoalan ini termasuk ke dalam persoalan permutasi.
2. Mengevaluasi setiap kombinasi susunan bom dengan data input apakah susunan tersebut valid.
3. Jika sudah ada susunan bom yang merupakan solusi (valid), pencarian akan dihentikan.

Pada *worst case scenario*, yaitu susunan lokasi bom yang benar diperoleh pada enumerasi kemungkinan terakhir, telah dilakukan enumerasi sebanyak P(n, b) dengan n dan b masing-masing menyatakan jumlah kotak dan jumlah bom pada papan permainan. Tentu hasil enumerasi ini sangat besar dan membutuhkan waktu yang cukup lama apalagi jika ukuran papan permainan cukup besar.

Salah satu cara untuk mengevaluasi hasil enumerasi merupakan solusi atau bukan adalah:

1. Representasikan susunan bom sebagai matriks *Boolean* dengan nilai *true* menandakan di lokasi tersebut terdapat bom. Untuk langkah-langkah selanjutnya, matriks ini akan disebut matriks m2.
2. Representasikan info angka-angka penunjuk jumlah bom sebagai matriks *Integer*. Untuk langkah-langkah selanjutnya matriks ini akan disebut matriks m1.
3. Kunjungi setiap elemen dalam matriks m2. Jika elemen yang sedang dikunjungi merupakan bom, maka kurangi dengan 1, semua nilai elemen matriks m1 yang berada di sekeliling elemen matriks m1 yang lokasinya identik dengan lokasi bom pada matriks m2.
4. Ulangi langkah ketiga hingga semua bom sudah dikunjungi.
5. Apabila hasil akhirnya, matriks m1 bernilai nol semua, maka artinya, matriks m1 berisi susunan bom yang benar.

Kompleksitas algoritma fungsi evaluasi di atas adalah O(row . col) atau setara dengan O(n) dengan n menyatakan total jumlah kotak pada papan permainan. Jadi untuk mengevaluasi 1 kombinasi susunan bom, diperlukan kompleksitas algoritma sebesar O(n). Sehingga, kompleksitas algoritma totalnya adalah O(n.P(n,d)) atau setara dengan O(n.n!/(n-d)!).

B. Penyelesaian Permainan Minesweeper dengan Algoritma Backtracking

Berikut ini adalah properti dari algoritma *backtracking* yang diterapkan pada permainan *minesweeper*:

1. Solusi adalah $X = (x_1, x_2, \ldots, x_n)$ dengan $x_i \in \{0,1\}$ dan n merupakan jumlah total kotak dalam papan

permainan. Nilai 1 pada x_i menyatakan bahwa pada posisi tersebut terdapat bom.

2. Fungsi pembangkit akan membangkitkan posisi kotak pada papan permainan yang belum pernah dikunjungi. Hal ini bisa diimplementasikan dalam program dengan membuat sebuah matriks *Boolean* berukuran sama dengan matriks papan permainan dimana nilai *true* akan menandakan bahwa posisi kotak (elemen matriks) tersebut sudah pernah dikunjungi.
3. Fungsi pembatas yang digunakan berupa fungsi untuk mengecek apakah dengan meletakkan bom di posisi kotak tertentu akan melanggar constraint dari matriks info angka-angka penunjuk jumlah bom (matriks m1) dan sudah tidak ada simpul status lain yang bisa dicapai dari simpul status saat ini. Maksud dari pernyataan tersebut adalah, apabila kita mengurangi nilai elemen matriks m1 di sekeliling titik yang diduga bom dengan 1, tidak mengakibatkan nilai elemennya menjadi negatif.

Langkah-langkah penerapan algoritma *backtracking* pada permainan *minesweeper* adalah sebagai berikut:

1. Membangkitkan posisi kotak yang belum pernah diperiksa sebelumnya menggunakan fungsi pembangkit. Misalkan hasil dari fungsi pembangkit tersebut adalah titik C(x,y)
2. Lakukan pengecekan pada elemen di sekeliling C(x,y) untuk memastikan apakah dengan mengurangi 1 dari elemen-elemen di sekitar C tidak mengakibatkan nilai elemen tersebut menjadi negatif.
3. Jika constraint di atas tidak dilanggar, ulangi langkah 1. Namun, jika dilanggar, lakukan *pruning* dan *backtrack* ke simpul status sebelumnya
4. Lakukan Kembali langkah 1 sampai 3 secara rekursif hingga ditemukan simpul status yang merupakan solusi atau seluruh simpul status sudah dikunjungi yang artinya tidak ada solusi yang ditemukan.

Secara umum, algoritma ini lebih baik dibanding dengan algoritma *brute force* karena algoritma ini tidak perlu mengenumerasi semua kombinasi susunan bom pada papan permainan melainkan bisa segera melakukan *pruning* dan *backtrack*. Kompleksitas dari algoritma ini pada *worst case scenario* adalah $O(2^n.n)$.

IV. Kesimpulan

Permainan *minesweeper* merupakan permainan suatu permainan teka-teki yang bertujuan untuk membuka semua kotak yang kosong (tidak mengandung bom) pada permainan dan menghindari membuka kotak yang berisi bom berdasarkan informasi yang tertera pada papan permainan berupa angka yang menyatakan jumlah bom yang terdapat di sekeliling kotak tersebut. Ada banyak algoritma yang bisa digunakan untuk menyelesaikan permainan *minesweeper* ini, beberapa di antaranya adalah algoritma *brute force* dan algoritma *backtracking*.

Algoritma *brute force* adalah algoritma dasar biasanya digunakan sebagai basis pembanding dengan algoritma lainnya yang lebih kompleks. Hal ini karena algoritma *brute force* yang bersifat *straightforward* dan sederhana

Algoritma *backtracking* dapat dikatakan merupakan algoritma pengembangan dari algoritma *exhaustive search*, yaitu dengan adanya kemampuan untuk melakukan *pruning* (pemotongan simpul status yang tidak mengarah ke solusi.

Berdasarkan hasil dari subbab pembahasan, dapat disimpulkan penyelesaian *minesweeper* menggunakan algoritma *bruteforce* membutuhkan ruang *memory* dan waktu yang besar karena perlu mengenumerasi semua kemungkinan / kombinasi susunan bom. Sedangkan algoritma backtracking dapat lebih baik dalam menyelesaikan permainan *minesweeper* karena dapat segera melakukan *pruning* ketika menemui simpul status yang sudah tidak mengarah ke solusi. Penggunaan algoritma *brute force* pada permainan *minesweeper* menghasilkan kompleksitas sebesar O(n.n!/(n-d)!) sedangkan pada algoritma *backtracking*, dihasilkan kompleksitas algoritma sebesar $O(2^n.n)$ dengan n merupakan jumlah total kotak pada papan permainan.

Ucapan Terima Kasih

Penulis ingin memanjatkan puji dan syukur kepada Tuhan Yang Maha Esa atas berkat dan rahmat-Nya sehingga penulis bisa menyelesaikan pembuatan makalah ini. Penulis juga ingin mengucapkan terima kasih yang sebesar-besarnya kepada semua pihak yang sudah membantu penulis dalam menyelesaikan pengerjaan makalah ini. Dimulai dari ucapan terima kasih kepada kedua orang tua yang sudah memberi dukungan baik fisik maupun mental. Kemudian ucapan terima kasih juga diberikan kepada Ibu Dr. Masayu Leylia Khodra, S.T, M.T. dan Ibu Dr. Nur Ulfa Maulidevi, S.T, M.Sc. selaku dosen mata kuliah IF2120 Matematika Diskrit yang mengajar di kelas saya, atas segala ilmu dan bimbingan yang telah diberikan selama satu semester ini. Kemudian juga kepada Bapak Dr. Ir. Rinaldi Munir M.T. atas *website* beliau yang berisi kumpulan materi dan makalah terdahulu yang telah menjadi referensi penulis dalam pembuatan makalah ini. Tidak lupa, penulis juga ingin mengucapkan terima kasih kepada teman-teman yang telah membantu penulis selama forum diskusi. Penulis juga ingin meminta maaf apabila terdapat kekurangan dalam penulisan makalah ini. Semoga isi dari makalah ini dapat bermanfaat dan bisa menambah wawasan pembaca.

Referensi

[1] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf, diakses pada tanggal 5 Mei 2022

[2] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag2.pdf, diakses pada tanggal 6 Mei 2022

[3] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian1.pdf, diakses pada tanggal 9 Mei 2022

[4] https://minesweepergame.com/website/authoritative-minesweeper/wiki/Windows_Minesweeper, diakses pada tanggal 12 Mei 2022

[5] https://lvngd.com/blog/solving-minesweeper-python-constraint-satisfaction-problem/, diakses pada tanggal 20 Mei 2022

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, 23 Mei 2022

Hansel Valentino Tanoto 13520046