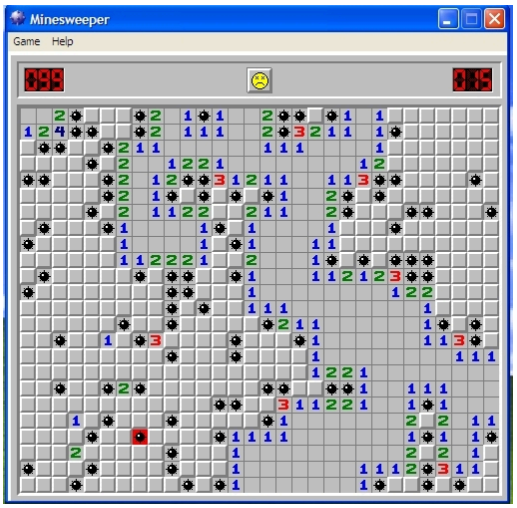
Penerapan Algoritma Brute Force, Backtracking, dan Greedy untuk Penyelesaian Permainan Minesweeper

Muhammad Ashkar - 13524007
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13524007@std.stei.itb.ac.id, ashkar.second@gmail.com

***Abstract*—Minesweeper adalah sebuah permainan puzzle interaktif dimana pemain akan diberikan sebuah papan permainan yang berisikan petak-petak yang awalnya tertutup yang di setiap petaknya terdapat sebuah bom atau angka. Tugas pemain adalah membuka semua petak dalam papan tanpa membuka petak yang berisi bom. Pemain bisa mengetahui lokasi bom dari dengan menggunakan petunjuk berupa petak-petak yang berisi angka. Setiap angka pada petak merepresentasikan berapa jumlah petak bom yang bertetanggaan dengan petak tersebut. Permasalahan minesweeper ini bisa dimodelkan menjadi sebuah masalah pemrograman yang solusinya bisa didapat dengan menggunakan algoritma-algoritma yang ada, seperti algoritma brute force, algoritma greedy, dan algoritma backtracking.**

Keywords—minesweeper; brute force; greedy; backtracking; puzzle; games

I. PENDAHULUAN



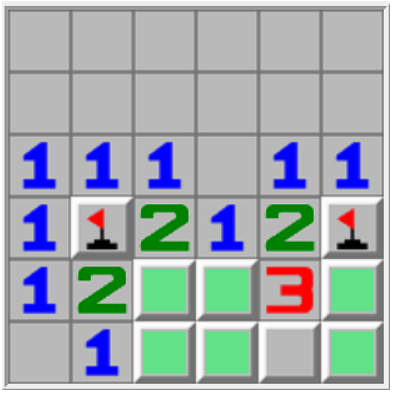
Gambar Permainan Minesweeper
(Sumber: https://cdn.mos.cms.futurecdn.net/f873f2282e16faeebdb4a09e2f3cef32.jpg)

Minesweeper adalah sebuah permainan puzzle interaktif dimana pemain akan diberikan sebuah papan yang berisikan petak-petak yang awalnya tertutup yang di setiap petaknya terdapat sebuah bom atau angka. Tugas pemain adalah membuka semua petak dalam papan tanpa membuka petak yang berisi bom. Pemain bisa mengetahui lokasi bom dari dengan menggunakan petunjuk berupa petak-petak yang berisi angka. Setiap angka pada petak merepresentasikan berapa jumlah petak bom yang bertetanggaan dengan petak tersebut. Pemain perlu menggunakan petunjuk tersebut untuk bisa menebak dimana saja lokasi petak bom.

Pada awal permainan, pemain bisa memilih petak mana saja yang dijamin bahwa petak tersebut bukan petak bom. Lalu semua petak yang berada di sebelah petak dengan angka 0 atau petak kosong akan terbuka secara otomatis. Dari situ pemain bisa mulai menebak dimana saja lokasi bom. Pemain juga bisa menandai bom dengan sebuah flag untuk mencegah jika pemain lupa atau tidak sengaja menekan petak tersebut. Permainan akan terus berlanjut hingga pemain berhasil membuka semua petak selain petak bom.

Permainan ini mungkin pada awalnya terlihat mudah untuk diselesaikan karena ada beberapa pola yang jelas seperti jika ada angka yang jumlah petak tertutup di sekitarnya sama dengan angka tersebut, maka semua petak tertutup tersebut bisa dipastikan adalah petak bom. Pada tingkat kesulitan yang mudah, strategi tersebut bisa terus digunakan hingga semua petak bom ditemukan. Namun pada tingkat kesulitan yang lebih sulit, terkadang strategi tersebut tidak bisa digunakan untuk menemukan semua petak bom. Terkadang perlu dilakukan simulasi dengan menganggap suatu petak adalah petak bom untuk mengetahui apakah petak tersebut sudah pasti bom atau bisa jadi bukan. Bahkan pada level yang lebih lanjut, petak bom tidak bisa dipastikan hanya dengan logika saja, perlu dilakukan perhitungan peluang untuk meminimalisir risiko menebak petak yang salah.

Karena kesulitan penyelesaian permainan tersebut, sebuah program komputer dibutuhkan untuk memastikan apakah permainan benar-benar bisa diselesaikan atau kita perlu mengambil risiko dengan menebak petak yang masih tertutup. Untuk membuat program yang bisa menyelesaikan permainan ini, dibutuhkan strategi algoritma tertentu yang bisa membantu dalam pencarian solusi.



Gambar Papan yang Tidak Bisa Diselesaikan dengan Aman (sumber: https://static.wikia.nocookie.net/minesweeper/images/7/7e/Extendedfinnedbox-2.png/revision/latest?cb=20240121162643)

II. Landasan Teori

A. *Algoritma Brute Force*

"When in doubt, use brute force." Begitulah kata Ken Thompson, penemu sistem operasi UNIX dan salah seorang tokoh terkenal dalam ilmu komputer. Semua permasalahan pasti bisa diselesaikan dengan algoritma brute force, namun tentu saja brute force memiliki satu kekurangan yaitu kompleksitas waktu yang sangat besar.

Sesuai dengan sebutannya, brute force tidak menggunakan pendekatan atau heuristik apapun dalam mencari solusi dari suatu persoalan, melainkan menggunakan cara paksa, yaitu dengan mencoba semua kemungkinan solusi sampai didapat solusi yang valid. Algoritma ini seringkali bisa dilakukan hanya dengan melakukan validasi constraint suatu persoalan pada suatu solusi. Jadi secara garis besar algoritma brute force hanya dibagi menjadi dua bagian yaitu membuat kemungkinan solusi dan melakukan validasi terhadap solusi.

Kebanyakan algoritma brute force sebenarnya bisa dioptimasi dengan menggunakan beberapa heuristik atau bahkan sebenarnya bisa dikembangkan menjadi algoritma yang lain. Tetapi, brute force yang murni tidak menggunakan heuristik apapun untuk mencari solusi dari suatu persoalan. Brute force akan terus mencoba tanpa informasi tambahan apapun untuk mencari solusi dari persoalan.

B. *Exhaustive Search*

Exhaustive search termasuk ke dalam jenis algoritma brute force. Hal ini disebabkan karena cara kerja exhaustive search yang mencoba semua kemungkinan yang ada hingga mendapat suatu solusi yang benar. Hal ini menyebabkan kompleksitas waktu dari exhaustive search sangat besar. Karena pada setiap kemungkinan solusi, perlu dilakukan pengecekan apakah jawaban valid atau tidak.

Untuk kasus minesweeper ini, berarti untuk setiap petak pada papan perlu dilakukan pengecekan apakah kondisi papan seperti itu valid atau tidak. Tentunya karena tidak menggunakan heuristik, maka proses optimasi algoritma dengan menggunakan pengetahuan tentang permainan minesweeper tidak bisa dilakukan. Algoritma ini menjamin bahwa solusi bisa ditemukan, namun dengan waktu yang belum tentu bisa dilakukan.

C. *Algoritma Backtracking*

Algoritma backtracking adalah algoritma brute force yang dioptimasi dengan memangkas kemungkinan solusi yang mirip dan tidak valid karena disebabkan oleh sumber yang sama. Daripada mencoba semua kemungkinan solusi yang mungkin untuk menyelesaikan persoalan, algoritma backtracking akan mencoba solusi untuk masing-masing petak daripada langsung mencoba solusi untuk langsung satu papan.

Algoritma backtracking akan mencoba semua kombinasi untuk sebuah petak, hingga mendapat sebuah nilai yang valid untuk petak tersebut. Selanjutnya, algoritma backtracking akan lanjut mencari solusi untuk petak berikutnya. Jika ternyata sama sekali tidak terdapat solusi yang valid untuk petak selanjutnya, maka algoritma backtracking akan melakukan proses backtrack ke petak sebelumnya dan akan mencoba solusi lain yang valid yang belum dicoba untuk petak tersebut.

Suatu solusi akan dihentikan di tengah jalan jika terdapat petak yang tidak valid sehingga semua kemungkinan kedepannya juga bisa dipastikan tidak akan valid. Hal membuat algoritma backtracking jauh lebih efisien daripada algoritma brute force. Karena jika terdapat salah satu petak yang membuat solusi tidak valid, maka kombinasi tersebut akan langsung dihentikan dan tidak akan dilanjutkan. Hal ini membuat pengecekan untuk banyak solusi yang tidak valid bisa langsung dilakukan dari sumber yang menyebabkan solusi itu menjadi tidak valid.

D. *Algoritma Greedy*

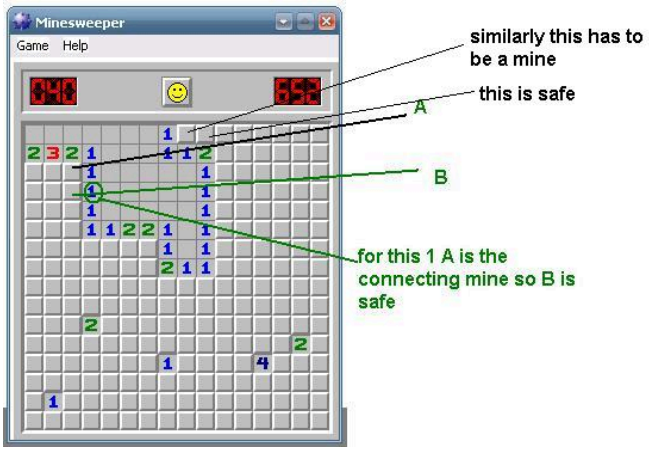
Algoritma Greedy adalah algoritma yang jika diberikan pilihan, maka algoritma greedy akan memilih pilihan dengan nilai 'paling'. Umumnya algoritma greedy mengambil nilai ekstrim, yaitu nilai paling minimum atau nilai paling maksimum. Karena sifat greedynya, algoritma greedy tidak memedulikan constraint lain dalam mengambil keputusan. Selama nilainya merupakan nilai 'paling', algoritma greedy akan mengambil keputusan tersebut.

Tapi dibanding dengan algoritma lain, algoritma tidak selalu menjamin bahwa solusi yang ditemukan adalah solusi yang benar atau valid. Algoritma greedy kadang bisa menghasilkan hasil yang salah atau tidak valid jika heuristik yang digunakannya salah. Bahkan algoritma greedy kadang bisa buntu dalam pencarian solusi. Hal ini adalah kekurangan utama dari algoritma greedy.

Namun di samping kekurangannya tersebut, algoritma greedy umumnya memiliki kompleksitas algoritma yang jauh lebih kecil daripada bruteforce. Hal ini bisa dilihat terutama pada masalah-masalah knapsack problem yang performa algoritma greedy jauh lebih baik dibanding algoritma algoritma lainnya.

E. *Heuristik Greedy*

Heuristik algoritma greedy yang akan digunakan adalah dengan mengecek petak-petak yang pasti. Hal ini dilakukan dengan menghitung kesesuaian jumlah antara angka yang ada pada petak dengan petak flag dan petak tertutup disekitarnya.



Gambar Petunjuk Menentukan Tile Aman (sumber: https://world2talkabout.wordpress.com/wp-content/uploads/2012/02/m3.jpg)

Petak yang jumlah angkanya sama dengan jumlah petak tertutup disekitarnya dan petak flag disekitarnya, sudah pasti petak tertutup disekitarnya adalah petak bom dan akan ditandai sebagai petak flag. Hal ini sudah pasti karena tidak ada kemungkinan lain yang menyebabkan petak tersebut adalah petak non-bom.

III. METODOLOGI

Masalah-masalah dalam penyelesaian puzzle minesweeper bisa diselesaikan dengan menggunakan beberapa algoritma, namun untuk bisa melakukan hal itu, kita perlu tahu dulu bagaimana cara memodelkan bagian dari permainan minesweeper menjadi bagian dari program. Kita perlu membuat representasi dari semua bagian yang membuat permainan tersebut ada ke dalam representasi kode. Untuk itu kita perlu memetakan permasalahan minesweeper menjadi struktur program.

Pertama kita perlu menentukan tempat untuk menyimpan data papan permainan minesweeper yang memuat semua data petak. Untuk itu kita perlu membuat matriks yang menyimpan informasi papan, dengan setiap elemen pada matriks menyimpan karakter yang merepresentasikan petak apa yang ada di posisi tersebut pada papan.

Selanjutnya perlu dilakukan proses validasi pada papan, karena jika papan sudah tidak valid dari awal, maka tidak akan ada solusi benar yang bisa didapatkan. Proses validasi dilakukan dengan mengecek jumlah baris dan kolom dari matriks yang dibuat. Jika jumlah baris dan kolom tidak konsisten, maka board tidak valid. Setelah itu perlu dilakukan juga validasi untuk apakah jumlah petak yang diberi flag tidak melebihi jumlah bom yang mungkin ada.

Setelah papan divalidasi, pengecekan dengan algoritma bisa dilakukan. Hasil dari algoritma akan disimpan dalam bentuk list posisi petak yang dipastikan adalah bom dan list posisi yang pasti aman. Dari hasil itu, pemain bisa menggunakan informasi yang diberikan untuk membuka petak-petak lainnya agar bisa mendapat informasi lebih banyak.

IV. IMPLEMENTASI ALGORITMA

A. *Alur Program*

Pertama-tama, pengguna perlu menginput algoritma apa yang ingin digunakan lalu memasukkan ukuran papan dan kondisi papan saat ini dengan format seperti berikut:

5 5

1 1 1 1 1

F 1 1 ? 2

? ? ? ? ?

? ? ? ? ?

? ? ? ? ?

Dengan dua angka pertama adalah jumlah baris dan kolom, lalu dilanjut dengan input petak dengan:

? = petak yang belum dibuka

F = petak yang ditandai dengan flag

1-8 = angka pada petak

Lalu dari input tersebut akan dilakukan validasi dan akan dilanjut dengan pengecekan dengan algoritma yang dipilih.

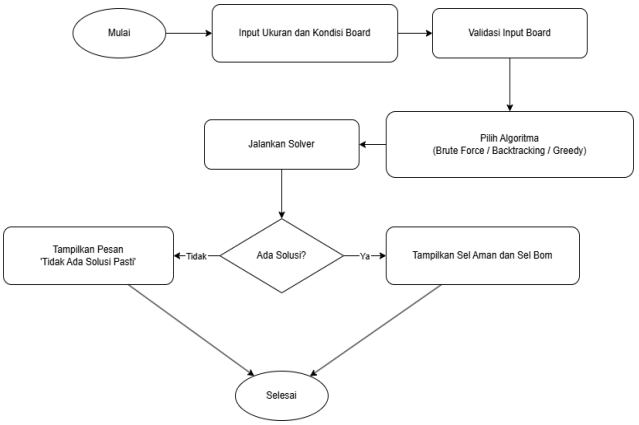


Diagram Alur Program

B. *Algoritma Brute Force*

Algoritma brute force dilakukan dengan mencoba satu persatu semua kemungkinan solusi papan yang mungkin. Lalu dari solusi tersebut dilakukan pengecekan apakah papan seperti itu valid atau tidak. Jika papan tidak valid, maka akan dicoba solusi selanjutnya hingga ketemu solusi yang valid.

Cara kerja algoritma brute force secara garis besar kurang lebih seperti berikut:

- Buat papan baru dengan mengubah semua simbol ‘?’ dari papan lama menjadi petak bom atau petak non-bom.
- Lakukan validasi pada papan.
- Jika papan tidak valid, gunakan kombinasi yang berbeda untuk peletakan petak bom pada papan.
- Lakukan hingga mendapat solusi yang valid.

Algoritma brute force untuk menyelesaikan minesweeper bisa dilihat pada pseudocode berikut:

```
BRUTE_FORCE_SOLVER(board):
    constraints = buat constraint dari board
    variables = semua cell UNKNOWN di board
    solutions = kosong

    SEARCH_ALL_ASSIGNMENTS(index):
        jika semua variables sudah diberi nilai:
            jika assignment memenuhi semua constraint:
                simpan assignment sebagai solusi valid
            return

        pilih variable saat ini

        coba isi SAFE
        SEARCH_ALL_ASSIGNMENTS(index + 1)

        coba isi MINE
        SEARCH_ALL_ASSIGNMENTS(index + 1)

        hapus nilai variabel saat ini

    jalankan SEARCH_ALL_ASSIGNMENTS(0)

    analisis semua solusi:
        cell yang selalu SAFE -> masukkan ke SAFE
        cell yang selalu MINE -> masukkan ke MINE

    kembalikan result
```

C. *Algoritma Backtracking*

Algoritma backtracking dilakukan dengan melakukan pengecekan setiap ada petak baru yang ditambahkan sebagai solusi, berbeda dengan algoritma brute force yang melakukan pengecekan setelah semua petak dalam papan diisi. Hal ini menyebabkan algoritma backtracking jauh lebih cepat daripada algoritma brute force.

Cara kerja algoritma backtracking secara garis besar adalah sebagai berikut:

- Ubah petak dengan simbol ‘?’ paling pertama yang ditemukan menjadi petak bom
- Lakukan validasi pada papan.
- Jika papan tidak valid, ubah petak bom menjadi petak non-bom.
- Jika petak sudah diubah menjadi non-bom dan masih belum valid, ubah petak sebelumnya menjadi petak non-bom.
- Lakukan hingga mendapat solusi yang valid.
- Jika semua petak sudah diubah menjadi petak non-bom tapi masih belum valid, maka tidak ada solusi valid untuk menyelesaikannya.

Algoritma backtracking untuk menyelesaikan minesweeper bisa dilihat pada pseudocode berikut:

```
BACKTRACKING_SOLVER(board):
    constraints = buat constraint dari board
    variables = semua cell unknown yang muncul di constraint
    solutions = kosong

    DFS(index):
        jika semua variabel sudah diberi nilai:
            simpan assignment sebagai solusi valid
            return

        pilih variable saat ini

        coba tandai sebagai SAFE
        jika assignment sementara masih mungkin valid:
            DFS(index + 1)

        coba tandai sebagai MINE
        jika assignment sementara masih mungkin valid:
            DFS(index + 1)

        hapus nilai variabel saat ini

    jalankan DFS(0)

    untuk setiap variabel:
        jika selalu SAFE di semua solusi:
            masukkan ke SAFE

        jika selalu MINE di semua solusi:
            masukkan ke MINE

    kembalikan result
```

D. *Algoritma Greedy*

Algoritma greedy yang digunakan untuk menyelesaikan permainan minesweeper adalah greedy dari petak yang paling aman. Petak yang paling aman adalah petak yang hasilnya sudah jelas, yaitu petak yang jumlah petak yang masih tertutup disekitarnya ditambah petak flag disekitarnya sama dengan angka pada petak tersebut, atau petak yang jumlah petak flag disekitarnya sama dengan angka pada petak tersebut.

Cara kerja algoritma greedy secara garis besar adalah sebagai berikut:

- Cari petak yang sudah terbuka
- Pada setiap petak terbuka, hitung jumlah petak tetangga yang masih tertutup dan petak tetangga yang ditandai dengan flag.
- Jika jumlah petak tertutup yang ada disekitar ditambah jumlah petak flag yang ada di sekitar sama dengan angka pada petak yang sedang dicek, maka tandai semua petak tertutup di sekitar sebagai petak bom.
- Jika jumlah petak flag yang ada disekitar sama dengan angka pada petak yang sedang dicek, maka tandai semua petak tertutup di sekitar sebagai petak aman.
- Lakukan sampai semua petak berhasil dicek.

Algoritma greedy untuk menyelesaikan minesweeper bisa dilihat pada pseudocode berikut:

```
GREEDY_SOLVER(board):
    result = kosong

    untuk setiap cell terbuka di board:
        unknown = tetangga cell yang masih tertutup
        flagged = jumlah tetangga yang ditandai bom
        remainingMines = clue cell - flagged
```

```
        jika remainingMines == 0:
            semua unknown adalah SAFE

        jika remainingMines == jumlah unknown:
            semua unknown adalah MINE

    kembalikan result
```

V. Analisis

A. *Kompleksitas Algoritma*

Bagus atau tidaknya suatu algoritma biasanya dilihat dari kompleksitas algoritma tersebut. Kompleksitas algoritma dibagi menjadi 2, yaitu kompleksitas waktu dan kompleksitas ruang. Namun, umumnya yang menjadi permasalahan dalam dunia pemrograman adalah kompleksitas waktu. Alasannya adalah karena ruang umumnya tidak terlalu menjadi masalah utama karena program sebenarnya tidak menggunakan ruang yang cukup besar jika dibandingkan dengan kompleksitas waktunya.

Kompleksitas waktu dari setiap algoritma tentunya berbeda-beda. Semakin rendah kompleksitas waktunya maka semakin bagus program tersebut. Kita bisa merepresentasikan kompleksitas waktu dari program dalam bentuk notasi Big O atau notasi $O(n)$. n adalah ukuran dari program dan angka-angka yang mengubah nilai n adalah kompleksitasnya seiring ukurannya berubah.

Untuk penyelesaian permainan minesweeper dengan algoritma brute force, kompleksitasnya adalah jumlah semua kemungkinan kombinasi dikali dengan kompleksitas untuk melakukan validasi papan. Kompleksitas dari validasi papan adalah $O(n\,x\,m)$ dengan n adalah jumlah baris dan m adalah jumlah kolom. Hasil tersebut didapat karena untuk setiap petak yang ada pada papan, dilakukan pengecekan petak tetangganya yang kompleksitasnya konstan. Lalu kompleksitas untuk semua kemungkinan yang ada adalah $O(2^{n\,x\,m})$. Maka kompleksitas untuk algoritma brute force adalah $O(n\,x\,m\,x\,2^{n\,x\,m})$ atau bisa disederhanakan menjadi $O(2^{n\,x\,m})$.

Untuk penyelesaian permainan minesweeper dengan algoritma backtracking, kompleksitasnya bisa sama dengan brute force pada worst case-nya, yaitu $O(2^{n\,x\,m})$. Hanya saja pada kasus umum, algoritma backtracking akan jauh lebih cepat daripada brute force, tergantung bagaimana kondisi papan pada saat pengecekan.

Untuk penyelesaian permainan minesweeper dengan algoritma greedy, kompleksitasnya adalah jumlah semua petak yang terbuka dikali dengan jumlah petak yang ada di sekitarnya. Karena jumlah petak yang ada disekitar konstan dan kita asumsikan bahwa jumlah petak yang terbuka adalah $n\,x\,m$, maka kita mendapat kompleksitas waktu untuk algoritma greedy adalah $O(n\,x\,m)$.

B. *Kelengkapan Solusi*

Algoritma brute force dan algoritma backtracking menjamin bahwa solusi pasti ditemukan selama papan valid. Hal ini disebabkan karena kedua algoritma tersebut berangkat dari solusi dan melakukan pengecekan pada solusi tersebut. Masalahnya, pada permainan minesweeper dengan tingkat kesulitan yang cukup tinggi, terkadang terdapat beberapa kemungkinan lokasi petak bom yang valid. Karena algoritma ini akan berhenti begitu menemukan solusi yang valid, algoritma ini mungkin saja memberikan solusi yang salah. Namun, pada kondisi seperti itu, memang tidak ada cara yang pasti untuk mengetahui petak mana yang aman dan petak mana yang bom.

Gambar Papan yang Tidak Memiliki Solusi Pasti (sumber: https://preview.redd.it/who-else-loves-minesweeper-v0-qslwspr3hvng1.jpeg?width=640&crop=smart&auto=webp&s=9d9c7539a4ef8f4b00710652adc65400ff0dfba8)

Sebaliknya, algoritma greedy dengan heuristik petak yang paling aman tidak akan menemukan solusi jika dihadapkan dengan kondisi yang sama. Hal ini karena algoritma greedy tidak bisa menemukan petak yang aman. Namun sayangnya algoritma ini juga bisa tidak menemukan solusi walaupun solusinya sebenarnya hanya ada satu. Hal tersebut bisa terjadi ketika ada sebuah kondisi dimana tidak ada petak yang terlihat aman menurut heuristik yang digunakan, namun sebenarnya jika dicoba secara brute force, akan ditemukan solusi yang pasti.

Gambar Papan yang Tidak Bisa Diselesaikan Algoritma Greedy (sumber: https://b.thumbs.redditmedia.com/K90WJUpYHmcyM4qVz7ct8XVIkgerukD0skOCLlRchdA.jpg)

VI. Kesimpulan

Dari hasil pengujian dan analisis, bisa kita lihat bahwa algoritma brute force adalah algoritma yang paling lambat, walaupun memberikan hasil yang pasti. Lalu dari algoritma brute force tersebut, bisa dilakukan optimasi sehingga kemungkinan solusi yang memang tidak akan menghasilkan solusi yang valid bisa langsung dipotong dari awal dengan cara langsung melakukan backtracking ketika menemukan solusi yang tidak, sehingga algoritma yang asalnya brute force berubah menjadi algoritma backtracking yang performa pada kasus umumnya jauh lebih baik daripada backtracking.

Algoritma yang paling bagus adalah algoritma greedy karena kompleksitas waktunya sangat rendah. Namun algoritma greedy memiliki kelemahan, yaitu sangat bergantung pada heuristik yang digunakan. Jika tingkat kesulitan permainan dinaikan, heuristik yang digunakan sekarang mungkin tidak akan bisa digunakan lagi. Di saat itulah algoritma backtracking dan brute force akan lebih unggul karena bisa menemukan solusi tanpa bantuan heuristik.

Jadi sebenarnya masing-masing algoritma memiliki kelebihannya masing-masing. Algoritma greedy memiliki keunggulan dalam segi kecepatan, sedangkan algoritma brute force dan backtracking memiliki keunggulan dalam segi kelengkapan solusi.

Apendiks

Github:
https://github.com/ashkar-scar/Minesweeper-Solver

Referensi

[1] R. Munir, "IF2211 Strategi Algoritma Semester I Tahun 2025/2026," STEI Institut Teknologi Bandung. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/stima25-26.htm [Accessed: 16-Jun-2026].

[2] "Minesweeper Online." [Online]. Available: https://minesweeper.online/ [Accessed: 16-Jun-2026].

[3] M. Burn, "How Minesweeper Solvers Work — Algorithms Explained," Minesweeper Blast, Mar. 11, 2026. [Online]. Available: https://minesweeperblast.com/minesweeper-solver-algorithms/ [Accessed: 19-Jun-2026].

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, 1 Juni 2026

Muhammad Ashkar
13524007