Perbandingan Algoritma *Greedy* & *Bactracking* Dalam Penyelesaian Permainan 2048

Stephen (13512025)[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13512025@std.stei.itb.ac.id

***Abstrak*—Makalah ini menjelaskan tentang penggunaan 2 buah algoritma dalam penyelesaian permainan yang cukup terkenal yaitu 2048. Kedua algoritma tersebut merupakan algoritma yang cukup populer dan mudah untuk digunakan dalam permainan, dan akan dibandingkan mana yang lebih baik antara *Greedy* atau *Bactracking*. Algoritma *Greedy* mentitikberatkan pada pengambilan keputusan yang terbaik yang ada saat itu juga dan tidak boleh kembali ke kondisi sebelum pengambilan keputusan tersebut. Algortima *Bactracking* secara umum menggambarkan adanya runut balik ke kondisi sebelum adanya pengambilan keputusan yang berkebalikan dengan algoritma *Greedy*.**

***Kata Kunci*—2048, greedy, backtrack, runut, balik.**

I. PENDAHULUAN

Di abad ke – 21 ini, istilah *game* atau permainan sudah tidak asing lagi di telinga kita. Begitu banyak permainan – permainan yang beredar di pasaran dan setiap harinya akan ada permainan baru yang bertambah, baik itu untuk permainan *console* seperti untuk *Playstation, Xbox, Wii*, serta untuk permainan *mobile* untuk *iOS* maupun *android*. Permainan – permainan tersebut sudah menjadi santapan sehari – hari kita, dimulai dari anak – anak sampai orang dewasa pun banyak yang memainkan permainan – permainan tersebut karena bersifat *addicting* dan dapat menjadi pengisi waktu di saat kita bosan atau kelelahan. Dengan permainan inilah kita bisa kembali bersemangat, kembali gembira, dan tentu saja hal tersebut semakin menyenangkan jika kita bisa bermain bersama teman – teman kita.

Terdapat sebuah permainan yang kini cukup terkenal, sebuah permainan yang cukup simpel yang berjudul ”2048”, permainan ini sebenarnya cukup mudah untuk dimainkan namun butuh sedikit keberuntungan dan kesabaran, kesimpel – an dan ide yang menarik dari permainan inilah yang membuat permainan ini cukup populer di kalangan remaja sampai orang dewasa. Permainan ini dirilis untuk berbagai macam *platform* yaitu untuk *android*, *iOS*, serta *windows phone*, dan bahkan permainan ini tersedia versi *flash* – nya di internet sehingga orang – orang yang tidak mempunya ketiga platform tersebut bisa memainkannya di komputer masing – masing, dan hal tersebut semakin mendongkrak kepopularitasan permainan ini.

Terdapat bermacam – macam cara yang digunakan oleh orang – orang untuk menyelesaikan permainan ini, berbagai macam algoritma sudah diciptakan untuk menyelesaikannya dengan secepat mungkin serta seaman mungkin. Di antara sekian banyak algoritma tersebut, dipilihlah 2 macam algoritma yang biasa digunakan yaitu algoritma *Greedy* dan *Backtracking*. Algoritma *Greedy* dalam permainan ini dilakukan dengan selalu menggeser ke arah yang mendapat poin terbesar daripada menggeser ke arah lainnya, serta algoritma *Backtracking* digunakan untuk kembali ke *state* sebelum dilakukannya perpindahan sehingga kita bisa memilih untuk mengganti arah gerakan.

Dari kedua algoritma tersebut, manakah yang dapat digunakan untuk penyelesaian yang lebih cepat, dan lebih aman untuk digunakan? Berikut akan dijelaskan perbandingan antara kedua algoritma tersebut dalam penyelesaian permainan “2048”.

II. DASAR TEORI

II.A. Permainan 2048

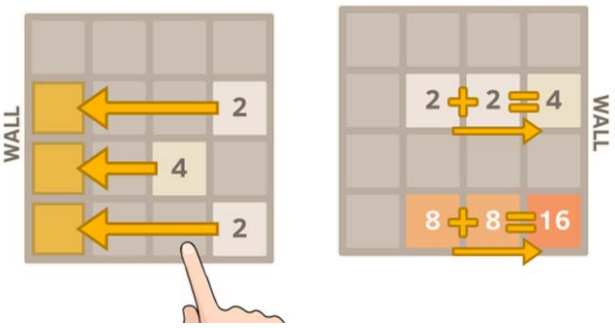
2048 merupakan permainan untuk pemain tunggal (dimainkan sendiri) yang diciptakan pada bulan Maret 2014 oleh seorang *programmer* asal Italia yang bernama Gabriele Cirulli. Berikut adalah tampilan dari permainan :

Gambar 2.1 Tampilan 2048

Ketika pertama kali dirilisnya permainan ini, permainan ini langsung melesat ke peringkat 1 dunia

dalam hanya waktu seminggu dan mencapai lebih dari 4 juta *download* dan hal tersebut merupakan rekor yang cukup fantastis. Pada kenyataannya, permainan ini hanyalah sebuah proyek "iseng" yang dikerjakan oleh *programmer* Italia tersebut yang hanya ingin mencoba apakah ia mampu membuat sebuah permainan dari awal dan ia berhasil menyelesaikan pembuatannya hanya dalam waktu seminggu. Permainan ini juga bukanlah yang pertama kali diciptakan, sebelumnya juga pernah ada permainan berjudul *Three!* yang memiliki tampilan serta cara bermain yang mirip, dan juga permainan *1024* yang cara bermainnya sama persis, hanya saja dengan tujuan permainan yang sedikit berbeda.

Cara bermain dari permainan tersebut cukup mudah yaitu menggerakann jari ke atas untuk menggeser semua kotak ke atas, menggerakkan jari ke kanan untuk menggeser seluruh kotak ke kanan, menggerakkan jari ke kiri untuk menggeser seluruh kotak ke kiri, serta menggerakkan jari ke bawah untuk menggeser seluruh kotak ke bawah, atau jika kita bermain dengan komputer, maka yang digunakan adalah tombol panah atas, kanan, kiri, dan bawah. Setiap kali kita melakukan pergeseran, jika terdapat 2 kotak yang berisi angka yang sama saling bertumpuk, maka kotak tersebut akan tergabung menjadi satu dengan berisikan hasil penjumlahan kedua angka tadi. Tujuan dari permainan ini adalah mendapatkan sebuah kotak yang berisikan bilangan 2048, yang berarti harus dilakukan penjumlahan dari 1024 + 1024 = (512 + 512) + (512 + 512) = ... dan seterusnya. Ukuran persegi yang disediakan adalah 4 kotak x 4 kotak, dan masing – masing kotak dengan nomor berbeda akan memiliki warna yang berbeda pula untuk mempermudah penglihatan kita terhadap angka – angka tersebut. Setiap selesainya pergerakan kita, maka berikutnya akan muncul lagi angka sembarang di sembarang tempat pula, dan hal tersebut lah yang menjadi salah satu tantangan dalam penyelesaian permainan ini.



Gambar 2.2 Cara Bermain

II.B. Algoritma Greedy

Algoritma *Greedy* merupakan salah satu algoritma yang populer digunakan dalam permasalahan optimasi, baik untuk pencarian solusi minimum maupun maksimum. Pada kenyataannya, algoritma *Greedy* seringkali tidak memberikan solusi yang paling optimum, karena menggunakan heuristik pemecahan masalah yang kurang baik, namun algoritma ini membuat pemecahan masalah menjadi jauh lebih singkat dibanding algoritma *Bruteforce* dan *Exhaustive Search*.

Secara bahasa, *Greedy* dalam bahasa Inggris berarti rakus, tamak, yang sesuai dengan prinsipnya yaitu ambil yang terbaik yang bisa diambil sekarang (*take what you can get now*). Algoritma ini membentuk solusi langkah per langkah, dan pada setiap langkah terdapat banyak pilihan yang perlu dievaluasi, oleh karena itu pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan Pada setiap langkah, kita membuat pilihan optimum lokal, dengan harapan bahwa langkah sisanya mengarah ke solusi optimum global. Berikut adalah contoh kasus yang menggunakan Algoritma *Greedy* :

1. Persoalan penukaran uang (solusi optimum)

Diberikan uang senilai 32, uang tersebut ingin ditukar menjadi uang – uang yang lebih kecil (uang koin). Bila uang koin yang tersedia bernilai 1, 5, 10, dan 25, berapakah minimum koin yang diperlukan untuk menukar uang tersebut?

Langkah penyelesaian :

Pada setiap langkah, memilih koin dengan nilai terbesar dari himpunan koin yang tersisa

Langkah 1 : Memilih 1 koin 25 (Total = 0 + 25 = 25)
Langkah 2 : Memilih 1 koin 5 (Total = 25 + 5 = 30)
Langkah 3 : Memilih 2 koin 1 (Total = 30 + 2 = 32)

Solusi : Jumlah koin minimum = 4 (**optimum**)

2. Persoalan penukaran uang (solusi tidak optimum)

Diberikan uang senilai 20, uang tersebut ingin ditukar menjadi uang – uang yang lebih kecil (uang koin). Bila uang koin yang tersedia bernilai 1, 10, dan 15, berapakah minimum koin yang diperlukan untuk menukar uang tersebut?

Langkah penyelesaian :

Pada setiap langkah, memilih koin dengan nilai terbesar dari himpunan koin yang tersisa

Langkah 1 : Memilih 1 koin 15 (Total = 0 + 15 = 15)
Langkah 2 : Memilih 5 koin 1 (Total = 15 + 5 = 20)

Solusi : Jumlah koin minimum = 6 (**tidak optimum**)
Solusi yang optimum adalah dengan memilih 2 buah koin 10 (10 + 10 = 20)

Algoritma *Greedy* memiliki 5 elemen, dan dalam permasalahan penukaran uang, elemen – elemen tersebut adalah :

1. Himpunan Kandidat
   Himpunan koin yang merepresentasikan nilai 1, 5, 10, 25, paling sedikit mengandung satu koin untuk setiap nilai.
2. Himpunan Solusi
   Total nilai koin yang dipilih tepat sama jumlahnya dengan nilai uang yang ditukarkan
3. Fungsi Seleksi
   Memilih koin yang bernilai tinggi dari himpunan kandidat yang tersisa
4. Fungsi Kelayakan

Memeriksa apakah nilai total dari himpunan koin yang dipilih tidak melebihi jumlah uang yang harus dibayar

5. Fungsi Obyektif
   Jumlah koin yang digunakan minimum

Skema umum Algoritma Greedy :

```
function Greedy (input C : himpunan_kandidat)→
himpunan_kandidat

{ Mengembalikan solusi dari persolan optimasi
dengan greedy
Masukan : himpunan_kandiat C
Keluaran : himpunan solusi yang bertipe
himpunan_kandidat}

Deklarasi
  x : kandidat
  S : himpunan _kandidat
Algoritma
  S←{}
  while (not SOLUSI(S) and (C≠{}) do
    x←SELEKSI(C)
    C←C – {X}
    if LAYAK(S∪{x}) then
      S← S∪{x}
    endif
  endwhile
  {SOLUSI(S) or C={})

  if SOLUSI(S) then
    return S
  else
    write('tidak ada solusi')
  endif
```

Dapat dilihat bahwa Algoritma *Greedy* di atas tidak memberikan solusi optimum pada persoalan yang berbeda, alasannya adalah :

1. Algoritma *Greedy* tidak beroperasi secara menyeluruh terhadap semua alternatif solusi yang ada (sebagaimana pada metode *exhaustive search*)
2. Terdapat beberapa fungsi seleksi yang berbeda, sehingga kita harus memilih fungsi yang tepat jika kita ingin algoritma menghasilkan solusi optimum.

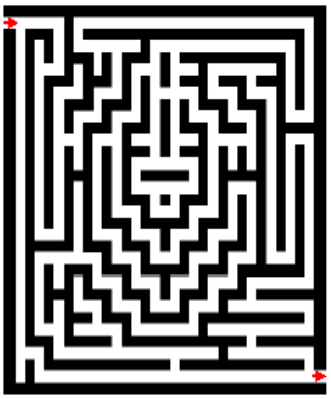
Algoritma *Greedy* sebaiknya digunakan jika jawaban **optimum mutlak** tidak diperlukan, dan solusi hampiran (*approximation*) sudah cukup untuk menyelesaikan permasalahan, karena untuk solusi optimum maka dilakukan waktu pemrosesanyang sangat lama. Bila algoritma *greedy* tersebut optimum, maka keoptimalannya dapat dibuktikan secara matematis.

II.C. Algoritma Backtracking

Algoritma *Backtracking* adalah cara yang metodologis mencoba beberapa sekuens keputusan, sampai ditemukan sekuens yang ”bekerja”. Algoritma ini digunakan ketika harus dibuat rangkaian keputusan di antara beberapa pilihan, dimana :

- Tidak punya cukup informasi untuk mengetahui apa yang akan dipilih
- Tiap keputusan mengarah pada sekumpulan pilihan baru
- Bebeapa sekuens pilihan (bisa lebih dari satu) mungkin merupakan solusi persoalan

Contoh persoalan yang menggunakan algoritma bactracking adalah *Maze Problem* : Diberikan sebuah labirin (*maze*), temukan lintasan dari titik awal sampai titik akhir.



Gambar 2.3 Contoh *maze problem*

Pada tiap perpotongan, harus diputuskan satu di antara tiga pilihan, yaitu antara maju terus, belok kiri, atau belok kanan. Kita tidak punya cukup informasi untuk memilih pilihan yang benar (yang mengarah ke titik akhir). Tiap pilihan mengarah ke sekumpulan pilihan lain dan satu atau lebih sekuens pilihan mengarah ke solusi, *bactracking* (runut – balik) dapat digunakan untuk persoalan seperti ini. Cara penyelesaian persoalan *maze* tersebut adalah :

- Membagi lintasan menjadi sederetan langkah
- Sebuah langkah terdiri dari pergerakan satu unit sel pada arah tertentu
- Arah yang mungkin : lurus (*straight*), kiri (*left*), dan kanan (*right*)

Bagaimana cara mengetahui langkah mana yang perlu dijejaki kembali? Ada 2 solusi untuk masalah ini :

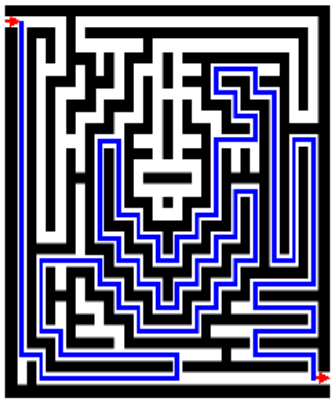
- Simpan semua langkah yang pernah dilakukan, atau kedua
- Gunakan rekursif (yang secara implisit menyimpan semua langkah)

Skema penyelesaian persoalan *maze* :

```
function SolveMaze (input M : labirin)→boolean
{ true jika pilihan mengarah ke solusi }

Deklarasi
  Arah : integer {up=1, down=2, left=3, right=4}

Algoritma :
  if pilihan arah merupakan solusi then
    return true
  else
    for tiap arah (lurus, kiri, kanan) do
      move(M, arah)   {pindah satu langkah
                       sesuai arah tersebut}
      if SolveMaze(M) then
        return true
      else
        unmove(M, arah) { backtrack }
      endif
    endfor
    return false    {semua arah sudah dicoba,
                     tetapi tetap buntu, maka
                      kesimpulan : bukan solusi}
  endif
```



Gambar 2.4 Penyelesaian persoalan *maze*

Runut – balik (*backtracking*) adalah algoritma pencarian solusi yang berbasis pada DFS. Algoritma ini banyak diterapkan untuk program permainan seperti permainan *tic – tac – toe*, catur, *sudoku*, *crossword puzzle*, dan masalah – masalah pada bidang kecerdasan buatan. Algoritma ini merupakan perbaikan dari algoritma *Bruteforce* dan *Exhaustive Search*, pada kedua algoritma itu semua kemungkinan solusi dieksplorasi satu per satu, sedangkan pada *backtracking*, hanya pilihan yang mengarah ke solusi yang dieksplorasi, pilihan yang tidak mengarah ke solusi tidak dipertimbangkan lagi (memangkas simpul – simpul yang tidak mengarah ke solusi).

Properti umum metode runut – balik :

1. Solusi Persoalan
   Solusi dinyatakan sebagai vektor dengan *n-tuple* :
   $X = (x_1, x_2, \ldots, x_n)$, $x_i \in S_i$
   Terdapat kemungkinan bahwa $S_1 = S_2 = ... = S_n$
   Contoh : $S_i = \{0, 1\}$, $x_i = 0$ atau 1
2. Fungsi Pembangkit
   Dinyatakan sebagai predikat : $T(k)$
   $T(k)$ akan membangkitkan nilai untuk x_k, yang merupakan komponen vektor solusi
3. Fungsi Pembatas
   Dinyatakan sebagai predikat $B(x_1, x_2, \ldots, x_k)$
   B akan bernilai *true* jika $(x_1, x_2, \ldots, x_k)$ mengarah ke solusi. Jika *true*, maka pembangkitan nilai untuk x_{k+1} dilanjutkan, tetapi jika *false*, maka $(x_1, x_2, \ldots, x_k)$ dibuang.

Semua kemungkinan solusi dari persoalan disebut **ruang solusi** (*solution space*), contoh :

{(0, 0, 0), (0, 1, 0), (0, 0, 1), (1, 0, 0),
(1, 1, 0), (1, 0, 1), (0, 1, 1), (1, 1, 1) }

Ruang solusi diorganisasikan ke dalam struktur pohon. Tiap simpul pohon menyatakan status (*state*) persoalan, sedangkan sisi (cabang) dilabeli dengan nilai – nilai x_i. Lintasan dari akar ke daun menyatakan solusi yang mungkin. Seluruh lintasan dari akar ke daun membentuk ruang solusi. Pengorganisasian pohon ruang solusi diacu sebagai **pohon ruang status** (*state space tree*). *Bactracking* dapat dipandang sebagai pencarian di dalam pohon menuju simpul daun (*goal*) tertentu.

III. PENYELESAIAN PERMAINAN 2048 MENGGUNAKAN ALGORITMA GREEDY DAN BACKTRACKING

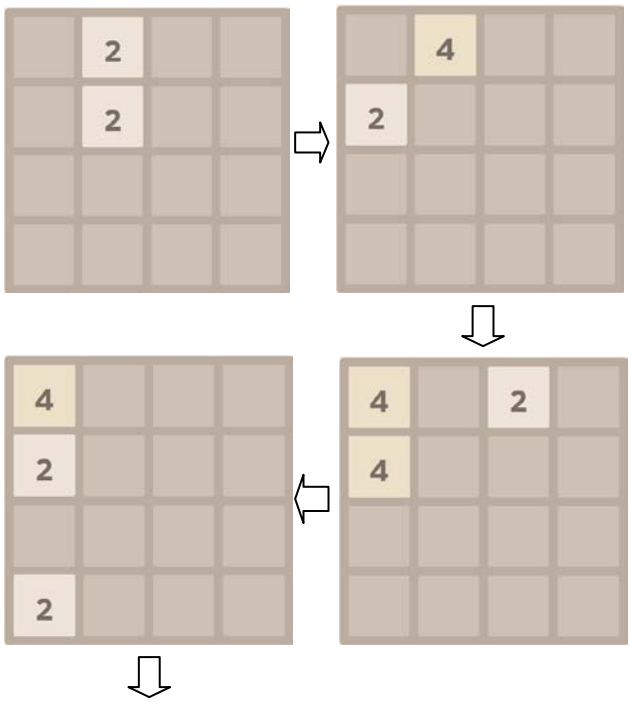
III.A. Penyelesaian Dengan Algoritma Greedy

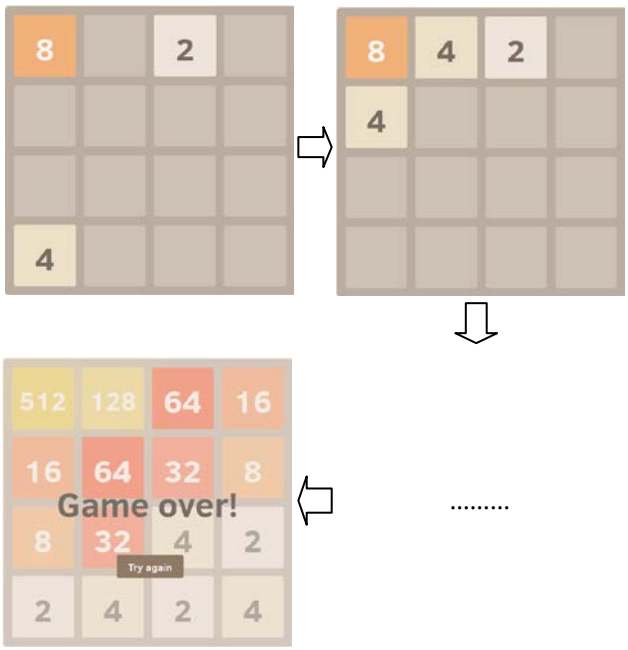
Algoritma *Greedy* dalam permainan ini digunakan dengan cara selalu menggeser papan permainan ke arah di mana akan didapatkan kotak baru dengan hasil penjumlahan yang paling besar, jika tidak ada maka selalu bergerak ke kiri dan atas, dengan harapan bahwa dengan cara ini maka suatu saat kemudian akan didapatkan kotak dengan nilai 2048.

Kelima elemen algoritma *Greedy* untuk persoalan ini :

1. Himpunan Kandidat
   Himpunan arah gerakan yaitu gerakan ke arah atas, arah bawah, arah kiri, dan arah kanan.
2. Himpunan Solusi
   Arah gerakan yang dipilih dari himpunan kandidat sebagai arah yang terbaik.
3. Fungsi Seleksi
   Mimilih arah gerakan sedemikian rupa sehingga pergerakan tersebut menghasilkan kotak baru dengan bilangan terbesar pada kotak tersebut.
4. Fungsi Kelayakan
   Memeriksa apakah pergerakan yang dipilih tidak menyebabkan adanya kotak lain yang tertutup sehingga tidak bisa ditambah lagi.
5. Fungsi Obyektif
   Pergerakan yang dipilih menghasilkan kotak dengan bilangan terbesar.

Berikut adalah percobaan pergerakan yang dilakukan dengan algoritma *Greedy* :

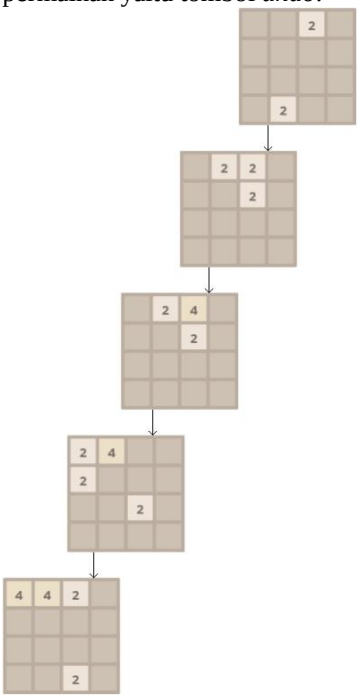




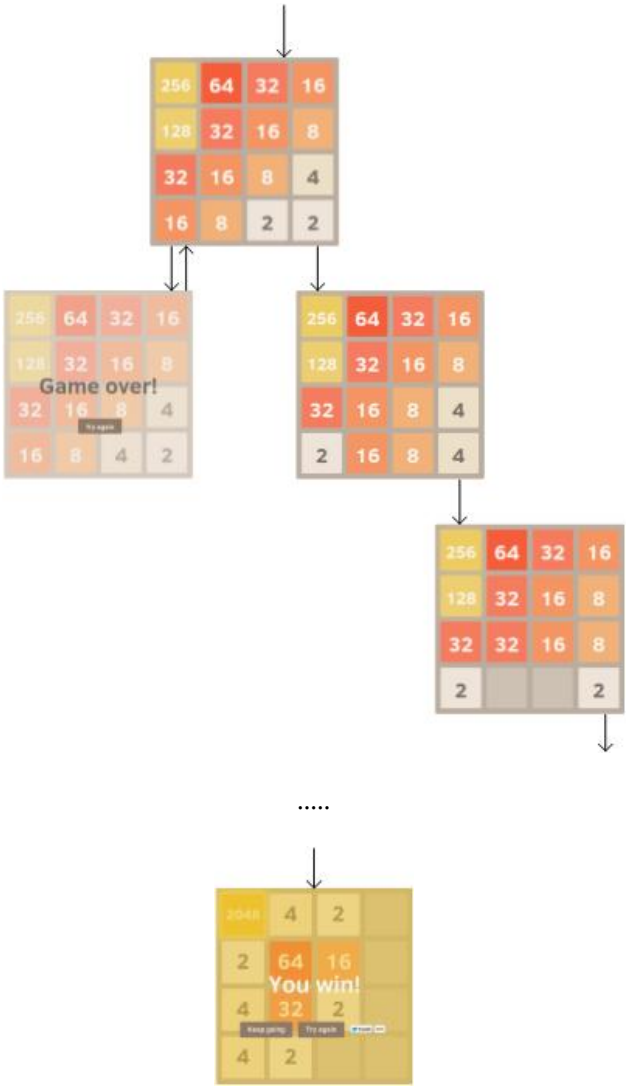
Jumlah percobaan : 11x
Rata – rata waktu per percobaan : 4 menit 32 detik
Hasil terbaik : 512

III.B. Penyelesaian Dengan Algoritma Backtracking

Penyelesaian dengan algoritma *backtracking* dilakukan dengan cara menggeser layar ke urutan arah : atas, kiri, kanan, bawah, jika bisa terus ke atas maka ke atas terus, jika tidak maka ke kiri, dan seterusnya sesuai dengan algoritma *Depth First Search* (DFS). Bedanya adalah ketika ditemukan bahwa sudah tidak memungkinkan untuk memenangkan permainan (tidak bisa membentuk lagi 2048 atau papan sudah penuh), maka akan dilakukan *backtrack* ke *state* sebelumnya dengan menggunakan tombol yang sudah disediakan oleh permainan yaitu tombol *undo*.



Setelah beberapa puluh langkah kemudian, pohon yang terbentuk adalah :



Jumlah percobaan : 1x
Rata – rata waktu per percobaan : 23 menit 18 detik
Hasil terbaik : 2048

IV. Analisis Perbandingan Antara Penyelesaian Dengan Greedy & Bactracking

Penyelesaian permainan menggunakan algoritma *Greedy* tidak dapat dilakukan, dari 11 kali percobaan menggunakan algoritma ini, tidak satupun kemenangan yang didapatkan, namun waktu yang dibutuhkan untuk mencapai nilai 512 cukup singkat yaitu hanya butuh waktu 4 menit dan 32 detik. Mendapatkan angka 2048 menjadi cukup sulit karena kemunculan angka baru yang cukup *random* atau acak dan tidak terduga, sehingga sangat memungkinkan untuk membuat papan menjadi penuh dan tidak bisa digeser lagi, perlu sedikit keberuntungan untuk menyelesaikannya dengan algoritma ini.

Penyelesaian permainan menggunakan algoritma *bakctracking* dapat dilakukan dan bahkan hanya dalam 1 kali percobaan. Walaupun berhasil, penyelesaiannya

memerlukan waktu yang cukup lama yaitu selama 23 menit 18 detik (hampir setengah jam) untuk satu kali permainan. Hal ini dikarenakan banyaknya jumlah *backtracking* yang memakan waktu cukup banyak, namun hal tersebut sangat penting dan berguna untuk menghindari papan yang penuh dan tidak bisa digerakkan lagi.

Jika dibandingkan, algoritma *Greedy* dan *backtracking* sama – sama memiliki kekurangan dan kelebihan, algoritma *Greedy* memiliki penyelesaian yang cepat dan hanya butuh sedikit keberuntungan untuk dapat menyelesaikan permainan, namun justru keberuntungan inilah yang sulit didapat sehingga permainan menjadi cukup sulit, sedang algoritma *backtracking* akan menjamin bahwa 2048 bisa dicapai namun tentu membutuhkan waktu yang cukup lama. Kedua algoritma ini bisa digunakan untuk menyelesaikan permainan 2048 dan tergantung diri kita masing – masing ingin menggunakan algoritma yang mana.

V. KESIMPULAN

Terdapat kelebihan dan kekurangan pada masing – masing algoritma, kelebihan dari algoritma *Greedy* adalah penyelesaiannya yang cepat, walaupun cukup sulit untuk melakukannya karena kemungkinan kegagalan cukup besar, faktor keberuntungan sangat berpengaruh di sini yang menentukan munculnya kotak yang baru. Kelebihan algoritma *backtracking* adalah penyelesaian permainan akan cukup mudah untuk dilakukan namun membutuhkan waktu yang cukup lama, karena akan sering dilakukannya runut – balik ke *state* sebelumnya.

VI. UCAPAN TERIMA KASIH

Pada kesempatan ini saya ingin mengucapkan terima kasih kepada Tuhan yang Maha Esa karena atas berkat-Nyalah makalah ini dapat diselesaikan. Saya juga ucapkan banyak terima kasih kepada Bu Masayu Leylia Khodra serta Bapak Rinaldi Munir selaku dosen Strategi Algoritma karena berkat pendidikan bimbingannya saya mendapat ilmu yang bisa terapkan dalam penyelesaian makalah ini. Terima kasih juga kepada semua pihak yang membantu dalam penyelesaian makalah ini.

REFERENSI

[1] Munir, Rinaldi, “Diktat Kuliah IF2211 Strategi Algoritma”, Bandung : Institut Teknologi Bandung .
[2] Introduction to Algorithms (Cormen, Leiserson, and Rivest) 1990, Chapter 17 "Greedy Algorithms" p. 329.
[3] http://www.geeksforgeeks.org/tag/backtracking/.
[4] http://www.cis.upenn.edu/~matuszek/cit594-2012/Pages/backtracking.html
[5] http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Greedy_algorithm.html

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, 16 Mei 2014

Stephen / 13512025