Penerapan Algoritma Backtracking dalam Permainan Mastermind

Christian Gunawan / 13519199
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13519199@std.stei.itb.ac.id:

***Abstract*—Mastermind adalah permainan papan dua pemain dimana salah satu pemain akan bertindak sebagai pembuat kode, dan yang lainnya bertindak sebagai pemecah kode. Pembuat kode akan memasang empat dari enam warna yang tersedia, dan pemecah kode harus menebak dengan benar kombinasi warna dalam ronde tertentu (biasanya 8-12 ronde). Permainan ini berakhir menang untuk pemecah kode saat berhasil menebak kombinasi warna yang tepat sebelum ronde selesai, sedangkan kemenangan bagi pembuat kode saat penebak kode kehabisan ronde. Untuk memecahkan permasalahan ini, dapat digunakan banyak algoritma. Pada kali ini, akan digunakan algoritma runut balik atau biasa dikenal sebagai algoritma *backtracking*. Begitu sebuah node terbukti tidak mengarah menuju solusi, node itu akan dipangkas dan oleh karena itu node ditempatkan di bawah simpul tersebut tidak akan diperiksa. Dengan melakukan ini, file sejumlah solusi yang mungkin harus diperiksa dapat menurun secara signifikan**

Keywords: Mastermind, Backtracking.

I. Introduction

Mastermind adalah permainan papan yang ditemukan pada tahun 1970-an oleh Mordecai Meirowitz, tetapi ia menyerupai permainan banteng dan sapi, permainan pensil dan kertas yang mungkin sudah ada sejak satu abad yang lalu atau lebih. Sepanjang waktu, game ini telah dimodifikasi menjadi banyak versi berbeda dan berganti nama yang berbeda. Selain menebak keempat warna, variasi *mastermind* termasuk word mastermind dimana pemain menebak empat huruf dari 26 kemungkinan huruf/alfabet yang ada. Selain *word mastermind*, terdapat *number mastermind* yang menggunakan angka sebagai pengganti warna, dan seri *mastermind* lainnya yang disebut *grand mastermind* yang tidak hanya menggunakan warna, tetapi juga menggunakan bentuk atau simbol.

Didalam permainan *mastermind*, terdapat sebuah papan dengan sebuah penutup untuk menutupi sebuah baris dengan empat buah lubang yang akan dimainkan oleh pembuat kode serta dua belas baris lainnya dengan empat buah lubang kecil setiap barisnya. Dalam menembak warna, setiap ronde dalam menebak warna selesai, pembuat kode akan memberikan kode yang biasa disebut pasak atau *pegs* dengan dua warna berbeda (Setiap game *mastermind* memiliki warna berbeda dan arti berbeda). Untuk lebih lanjutnya, akan dijelaskan pada bab berikutnya.

Gambar 1. Papan Permainan Mastermind

Banyak algoritma yang dapat diterapkan untuk menyelesaikan permainan ini. Salah satunya adalah brute force. Untuk menemukan jawaban yang benar, algoritma *brute force* akan menguji setiap kemungkinan jawaban untuk melihat apakah itu solusinya. Dengan menggunakan algoritma *brute force*, berarti kita akan menebak kombinasi empat warna yang disusun oleh pembuat kode dan kita memiliki enam kemungkinan warna tersebut, maka akan ada 6^4 kemungkinan jawaban yang perlu diuji. Hal ini sangat tidak efisien, oleh karena itu dalam penulisan ini akan dibahas solusi game *mastermind* menggunakan algoritma *backtracking* atau runut balik yang diharapkan dapat menyelesaikan masalah dengan lebih cepat dan efisien.

II. TEORI DASAR

A. Peraturan Mastermind

Mastermind dimainkan di papan yang dibuat khusus yang disebut *papan decoding*. Papan ini memiliki penutup di ujungnya guna menutupi empat warna yang akan dibuat oleh pembuat kode dan terdapat lebih dua belas baris untuk menebak oleh penebak kode. Kedua belas baris tersebut dapat dijadikan acuan sebagai putaran yang diberikan kepada pemecah kode. Pada setiap baris memiliki empat lubang untuk

pasak kode, dan empat lubang kecil tepat di samping pasak kode yang disebut pasak kunci. Pasak kode adalah pasak enam warna berbeda yang digunakan untuk penebak kode menebak kombinasi warna dengan meletakkannya di lubang di papan. Perlu diperhatikan, bahwa posisi dan warna akan berpengaruh untuk selanjutnya Pasak kunci adalah pasak berkepala datar, ada yang berwarna (seringkali berwarna hitam) dan ada yang berwarna putih, yang kedua ukurannya berukuran lebih kecil dari pasak kode. Pasak kunci ini nantinya akan ditempatkan di lubang yang lebih kecil di papan atau di samping pasak kode sebagai umpan balik dari pembuat kode. [1]

Pertama kali pembuat kode akan membuat kombinasi empat warna dari enam kemungkinan warna dan menempatkan di posisi yang diinginkan oleh pembuat kode. Warna yang diletakkan boleh lebih dari satu, yang berarti kombinasi tersebut dapat berupa empat pasak dengan warna yang sama. Kombinasi yang dipilih diletakkan di empat lubang yang ditutupi dan hanya dapat dilihat oleh pembuat kode, tetapi tidak dapat dilihat oleh pemecah kode. Ketika kombinasi warna telah dipilih, pemecah kode dapat mulai menebak posisi dan warna dari pasak yang telah dibuat oleh pembuat kode. Ketika tebakan telah dibuat, pembuat kode kemudian akan memberikan umpan balik dengan menempatkan nol hingga empat pasak kunci di lubang yang lebih kecil yang disediakan. Pasak kunci berwarna menunjukkan adanya pasak kode dari tebakan yang benar dalam posisi dan warna. Sedangkan pasak kunci berwarna putih yang diletakkan berarti untuk setiap pasak warna kode berarti warnanya benar, tetapi ditempatkan di posisi yang salah.

Setelah pembuat kode memberikan umpan balik, pemecah kode diizinkan untuk membuat tebakan lagi di baris di depannya. Tebak-tebakan kode dari pemecah kode dan umpan balik dari pembuat kode akan terus bergantian. Pemecah kode menang jika kombinasi warna yang benar sebelum ronde selesai atau baris habis, atau pembuat kode menang jika pemecah kode kehabisan giliran.

B. Pohon

Pohon adalah sebuah struktur data non-linear. Sebuah pohon merepresentasikan struktur hierarkikal yang dimiliki oleh simpul-simpulnya yang salah satunya di antara mereka berperan sebagai akar. [2]

Beberapa istilah dalam struktur data pohon adalah sebagai berikut:

1. Simpul adalah sebuah elemen yang menyusun sebuah struktur data pohon yang didalamnya bisa menyimpan informasi dan biasanya diberikan identitas unik berupa indeks.
2. Akar adalah sebuah simpul yang memiliki letak paling tinggi pada struktur data pohon.
3. Lintasan atau *path* adalah simpul-simpul yang harus dilewati untuk berpindah dari suatu simpul a ke simpul b.
4. Tingkat atau *level* sebuah simpul pada suatu pohon adalah besar jarak simpul tersebut terhadap akar.
5. Tinggi atau kedalaman dari sebuah pohon adalah tingkat maksimum dari seluruh simpul-simpul yang dimiliki oleh pohon.
6. Predecessor adalah istilah yang menyatakan simpul yang berada satu level di atas suatu simpul yang sedang diacu.
7. Successor adalah istilah yang menyatakan simpul yang berada satu level di bawah suatu simpul yang sedang diacu.
8. Orang tua atau parent adalah Predecessor satu level di atas suatu simpul.
9. Anak atau *child* adalah Successor satu level di bawah suatu simpul.
10. Keturunan (*descendant*) dan leluhur (*ancestor*) dapat dimisalkan sebagai berikut. Simpul a dikatakan sebagai keturunan simpul b apabila terdapat lintasan dari simpul b ke simpul a, pada kasus ini simpul b juga dapat dikatakan sebagai leluhur dari simpul a.
11. Saudara kandung atau *Sibling* adalah Simpul-simpul yang memiliki parent dan leluhur yang sama.
12. Upapohon atau subpohon (*subtree*) adalah bagian dari pohon yang berupa suatu simpul beserta keturunannya dan memiliki karakteristik dari pohon itu sendiri.
13. Derajat sebuah simpul pada pohon menyatakan banyak anak yang dimiliki atau banyak simpul yang bertetanggaan.
14. Daun adalah sebuah simpul pada pohon yang tidak memiliki anak.
15. Simpul dalam adalah simpul pada pohon yang memiliki anak dan juga memiliki orang tua.

C. Algoritma Brute Force

Algoritma Brute Force adalah pendekatan yang lempang (*straightforward*) untuk memecahkan suatu persoalan atau dapat dibilang algoritma yang “memaksa” cara untuk menemukan solusi. Biasanya algoritma ini didasarkan pada persoalan (problem statement) dan definisi/konsep yang dilibatkan. Algoritma ini memecahkan persoalan dengan sangat sederhana, langsung, dan jelas caranya. [3]

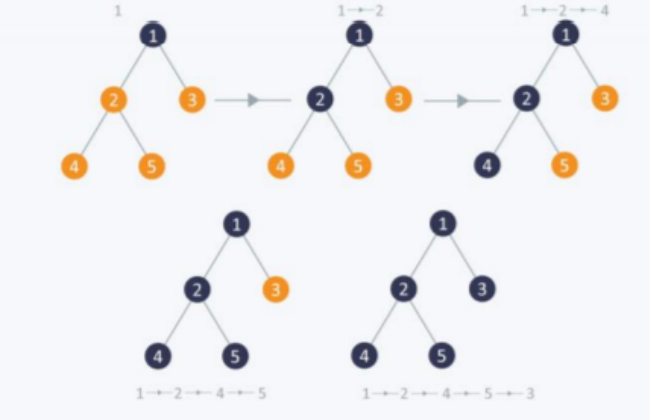
Beberapa karakteristik algoritma brute force antara lain Bukanlah solusi yang 'cerdas' dan efektif dalam menyelesaikan penyelesaian karena membutuhkan banyak komputasi dan waktu, lebih baik digunakan untuk masalah skala kecil, dan hampir setiap masalah bisa diselesaikan dengan menggunakan algoritma brute force.

D. Depth-First Search

Depth-First Search atau biasa disebut DFS adalah salah satu algoritma penelusuran struktur graf / pohon berdasarkan kedalaman. Simpul ditelusuri dari root kemudian ke salah satu simpul anaknya (misalnya prioritas penelusuran berdasarkan anak pertama [simpul sebelah kiri]), maka penelusuran dilakukan terus melalui simpul anak pertama dari simpul anak pertama level sebelumnya hingga mencapai level terdalam. Setelah sampai di level terdalam, penelusuran akan kembali ke 1 level sebelumnya untuk menelusuri simpul anak kedua pada pohon biner [simpul sebelah kanan] lalu kembali ke langkah sebelumnya dengan menelusuri simpul anak pertama lagi sampai level terdalam dan seterusnya.

Algoritma dari Depth-First Search dengan bantuan gambar dibawah adalah sebagai berikut :

1. Kunjungi simpul 1
2. Kunjungi simpul 2 yang bertetangga dengan simpul 1.
3. Ulangi DFS mulai dari simpul 2.
4. Ketika mencapai simpul 4 sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian di runut-balikan (*backtracking*) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul 2 yang belum dikunjungi.
5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi



Gambar 2. Contoh Kasus DFS

Sumber:
www.hackerearth.com/practice/algorithms/graphs/depth-first-search/tutorial

E. Algoritma Backtracking

Algoritma Backtracking dapat dipandang sebagai sebuah fase di dalam algoritma traversal DFS atau Sebagai sebuah metode pemecahan masalah yang mangkus, terstruktur, dan sistematis, baik untuk persoalan optimasi maupun non-optimasi. Algoritma *backtracking* atau runut-balik merupakan perbaikan dari *exhaustive search*. Pada *exhaustive search*, semua kemungkinan solusi dieksplorasi dan dievaluasi satu per satu. Sedangkan pada algoritma *backtracking*, hanya pilihan yang mengarah ke solusi yang akan dieksplorasi, sedangkan pilihan yang tidak mengarah ke solusi tidak dipertimbangkan lagi. Hal itu dapat dilakukan dengan memangkas simpul-simpul yang tidak mengarah ke solusi.

Algoritma runut-balik pertama kali diperkenalkan oleh D. H. Lehmer tahun 1950. Kemudian R.J Walker, Golomb, dan Baumert menyajikan uraian umum tentang algoritma *backtracking*.

Terdapat properti umum pada algoritma backtracking, antara lain,

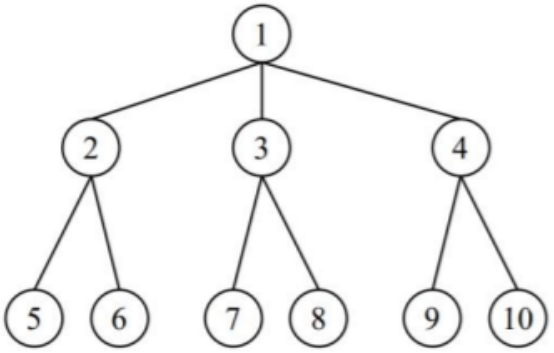
a. Solusi Persoalan

Solusi dinyatakan sebagai vektor dengan n-tuple: X = (x1, x2, …, xn), xi ϵ Si, Umumnya S1 = S2 = … = Sn. Contohnya pada persoalan 1/0 knapsack Si = {0, 1}, xi = 0 atau 1

b. Fungsi pembangkit nilai xk

Dinyatakan sebagai predikat T(). T(x[1], x[2], …, x[k – 1]) membangkitkan nilai untuk xk, yang merupakan komponen vektor solusi.

Agar lebih jelas dapat dilihat contoh berikut:



Gambar 3. Contoh Kasus Algoritma Backtracking

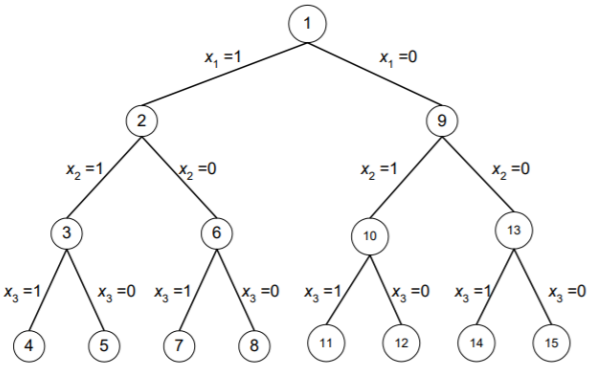
Misalkan pohon diatas menggambarkan solusi dari suatu permasalahan. Untuk mencapai solusi (5), maka jalan yang ditempuh adalah (1,2,5), demikian juga dengan solusi-solusi yang lain. Algoritma backtrack akan memeriksa mulai dari solusi yang pertama yaitu solusi (5). Jika ternyata solusi (5) bukan solusi yang layak maka algoritma akan melanjutkan ke solusi (6). Jalan yang ditempuh ke solusi (5) adalah (1,2,5) dan jalan untuk ke solusi (6) adalah (1,2,6). Kedua solusi ini memiliki jalan awal yang sama yaitu (1,2). Jadi daripada memeriksa ulang dari (1) kemudian (2) maka hasil (1,2) disimpan dan langsung memeriksa solusi (6). Pada pohon yang lebih rumit, cara ini akan jauh lebih efisien daripada algoritma *brute force*.

c. Fungsi pembatas (bounding function)

Dinyatakan sebagai predikat B(x1, x2, …, xk). B bernilai true jika (x1, x2, …, xk) mengarah ke solusi. Mengarah ke solusi artinya tidak melanggar kendala (*constraints*). Jika true, maka pembangkitan nilai untuk xk+1 dilanjutkan, tetapi jika

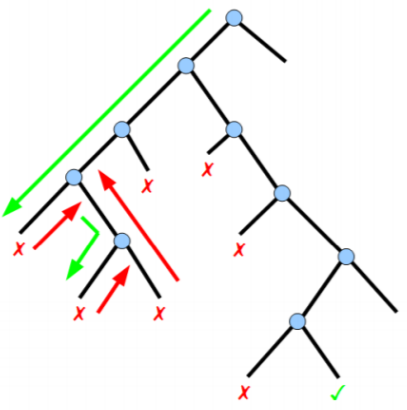
false, maka (x1, x2, …, xk) dibuang atau dipangkas. Algoritma *backtracking* dapat dipandang sebagai pencarian di dalam pohon dari akar menuju daun (simpul solusi).

Prinsip pengorganisasian solusi algoritma *backtracking* antara lain, Semua kemungkinan solusi dari persoalan disebut ruang solusi (solution space). Solusi persoalan dinyatakan sebagai X = (x1, x2, x3) dengan xi {0,1}. Misal, persoalan *knapsack* 1/0 dengan n=3, maka ruang solusi yang dapat terbentuk adalah {(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 berakar. Tiap simpul pohon menyatakan status (state) persoalan, sedangkan sisi (cabang) dilabeli dengan nilai-nilai xi. 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*).



Gambar 4. Pohon Solusi dari Ruang Solusi

Prinsip pencarian solusi dengan Algoritma *Backtracking* antara lain, Solusi dicari dengan membangkitkan simpul-simpul status sehingga menghasilkan lintasan dari akar ke daun. Aturan pembangkitan simpul yang dipakai adalah mengikuti aturan *depth first order* (DFS). Simpul-simpul yang sudah dibangkitkan dinamakan simpul hidup (*live node*). Simpul hidup yang sedang diperluas dinamakan simpul-E (*Expand-node*). Tiap kali simpul-E diperluas, lintasan yang dibangun olehnya bertambah panjang. Jika lintasan yang sedang dibentuk tidak mengarah ke solusi, maka simpul-E tersebut dimatikan atau dipangkas sehingga menjadi simpul mati (*dead node*). Fungsi yang digunakan untuk mematikan simpul-E adalah dengan menerapkan fungsi pembatas (*bounding function*). Ketika sebuah simpul berarti, maka secara implisit kita telah memangkas atau mematikan simpul-simpul anaknya. Jika pembentukan lintasan berakhir dengan simpul mati, maka proses pencarian mundur sampai ke simpul pada aras diatasnya. Lalu, teruskan dengan membangkitkan simpul anak yang lainnya. Selanjutnya simpul ini menjadi simpul-E yang baru. Pencarian dihentikan bila kita telah sampai pada simpul tujuan(*goal node*).



Gambar 5 Pohon Solusi dengan Backtracking

F. *Skema Umum Algoritma Backtracking*

Dalam algoritma *backtracking*, pencarian dapat dilakukan dengan iteratif maupun rekursif. Akan tetapi jika pencarian dilakukan secara iteratif, dibutuhkan pula penambahan dan pengurangan secara manual beserta pengontrolnya. Pencarian menggunakan skema rekursif dianggap lebih tepat dan efisien dikarenakan pencarian solusi dalam pembangkitan simpul lebih sesuai dengan algoritma backtracking. Berikut ini skema iteratif dan skema rekursif dari algoritma Backtracking:

Skema Umum versi iteratif

```
procedure RunutBalikI(input k : integer)
{Mencari semua solusi persoalan dengan metode runut-balik; skema iteratif
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:
  while k ≠ 0 do
    if terdapat nilai x[k] yang belum dicoba sedemikian sehingga x[k] ∈ T(x[1], x[2], …, x[k – 1]) and
       B(x[1], x[2], ... ,x[k]) = true then
       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
       k ← k + 1     { tentukan nilai x[k] selanjutnya }
    else
       k ← k – 1
    endif
  endwhile
                                                    Pemanggilan pertama kali: RunutBalikI(1)
```

Gambar 6. Algoritma Backtracking versi iteratif

Skema Umum versi rekursif

```
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:
  for setiap x[k] ∈ T(x[1], x[2], …, x[k – 1]) do
     if B(x[1], x[2], ... ,x[k]) = true then
        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 7. Algoritma Backtracking versi rekursif

III. Analisis

Dalam menyelesaikan game mastermind, sebelum memulai mencari dengan algoritma *backtracking*, kita perlu menentukan status game tersebut. Di sini kami mengamati bahwa, untuk setiap tebakan yang dibuat, ada empat belas kemungkinan umpan balik dari pembuat kode yang berbeda. Keempat belas umpan balik yang dimaksud antara lain, tidak ada yang pasak kunci, satu pasak kunci putih, satu pasak kunci berwarna, dua pasak kunci putih, satu pasak kunci berwarna dan satu pasak kunci putih, dua pasak kunci berwarna, tiga pasak kunci putih, satu pasak kunci berwarna dan pasak kunci putih, dua pasak kunci berwarna dan satu pasak kunci putih, tiga pasak kunci berwarna, empat pasak kunci putih, satu pasak kunci berwarna dan pasak kunci putih, dua pasak kunci berwarna dan dua pasak kunci putih, empat pasak kunci berwarna yang menandakan semua benar dan ronde berakhir yang dimenangkan oleh pemecah kode. Perhatikan bahwa umpan balik yang disebutkan hanya ada tiga belas karena pada tiga pasak berwarna dan satu pasak putih tidak dimungkinkan, karena jika sudah ada tiga pasak di posisi yang benar, maka pasak keempat dengan warna yang benar tetapi ditempatkan di posisi yang salah kemungkinan besar tidak ada. Jadi, sekarang hanya ada tiga belas kemungkinan pasak kunci yang dikeluarkan oleh pembuat kode.

Untuk kedepannya, kita akan mengacu enam kemungkinan warna menggunakan huruf A sampai F agar memudahkan dalam penyebutan. Sekarang, untuk mendapatkan umpan balik yang dapat memberi kita informasi paling berguna untuk menebak kombinasi warna yang benar, kita akan mencoba menebak empat kombinasi warna langkah pertama agar memudahkan kita dalam menebak kombinasi warna berikutnya. Dalam memilih empat pasak warna, ada 6^4= 1296 kemungkinan kombinasi warna. Namun, untuk variasi warna dan posisi, hanya ada lima kombinasi yang pada dasarnya berbeda, yakni AAAA (Keempat warna yang sama), AAAB (Ketiga warna sama dan satu warna berbeda), AABB (Ada dua warna sama dan ada dua warna lainnya yang sama), AABC (Dua warna sama dan dua warna lainnya berbeda), dan ABCD (Keempat warna berbeda). Inilah lima kemungkinan yang bisa dipilih sebagai langkah pertama. Setiap kali tebakan dibuat, umpan balik yang diberikan akan mengurangi jumlah solusi yang mungkin. Misalnya, jika pemecah kode menebak 'AAAA' dan umpan balik yang diberikan adalah tidak ada pasak kunci yang benar, berarti warna A tidak ada dalam kombinasi, sehingga membuat jumlah kemungkinan kombinasi warna berkurang menjadi hanya 5^4 = 625 kombinasi warna. Secara singkat, menunjukkan bahwa solusi yang terbaik memulai menebak warna adalah AABB karena didapatkan kombinasi warna berkurang menjadi 256 kemungkinan yang menjadikan kemungkinan menjadi sangat kecil dibandingkan kelima kombinasi lainnya. [5]

Untuk membuat solusi game *mastermind* menggunakan algoritma *backtracking*, kita akan menggunakan konvensi enam kemungkinan warna akan disimpan dalam susunan warna yang akan ditunjukkan oleh huruf/alfabet A hingga F dan untuk setiap tebakan, masukan akan dihitung sebagai skor. Setiap pasak berwarna akan diberi nilai 2 sedangkan pasak putih akan diberi nilai masing-masing nilai 1. Setelah itu, kita perlu menyesuaikan dulu kondisi permainannya. Jika menggunakan papan kosong sebagai awal dari proses penelusuran mundur, itu berarti akar pohon solusi adalah papan kosong, proses penelusuran mundur mungkin sedikit terlalu rumit. Oleh karena itu, sebagai langkah awal, kita perlu menentukan kombinasi warna mana yang akan digunakan sebagai awal proses runut mundur. Sesuai dengan pembahasan sebelumnya, maka akan digunakan pola AABB. Kombinasi warna yang ditandai dengan dua pasak putih (skor 2) akan dianggap sebagai kebalikannya (misalnya AABB akan dianggap sebagai BBAA), dan dianggap ditandai dengan dua pasak berwarna (skor 4).

Setelah akar pohon ditentukan, pohon tersebut kemudian diperluas dengan memeriksa setiap posisi yang memungkinkan (misalnya, AABB akan diperluas menjadi BABB, ABBB, AAAB, dan AABA), kemudian dengan mencoba setiap warna yang memungkinkan di satu posisi yang sama (AABB akan diperluas menjadi BABB, CABB, DABB, EABB, dan FABB). Kombinasi yang sudah dicek tidak akan dicek dua kali. Dalam memperluas pohon, posisi diubah dari kiri, sedangkan warna diubah berdasarkan urutannya dalam pemeriksaan. Warna yang terlibat dalam tebakan dengan skor 0 akan dibuang dari pemeriksaan. Kombinasi warna yang memberikan skor lebih sedikit akan secara otomatis dipangkas dari pohon solusi.

Untuk selanjutnya, kita akan tinjau proses untuk menemukan solusi jika kode yang akan ditemukan adalah **FCBE**. Pertama kali, kita akan menentukan akar pohonnya. Dalam hal ini, kemungkinan skor didapat gerakan pertama adalah AABB satu pasak berwarna dengan skor 2, CCDD satu pasak berwarna dengan skor 2, dan EEFF dua pasak putih dengan skor 2. Dari hasil tersebut kita dapat melihat bahwa setiap gerakan pertama memiliki skor yang sama. Tetapi perhatikan bahwa kemungkinan langkah pertama pada warna EEFF memiliki hasil dua pasak putih. Ini berarti bahwa kita dapat menganggap kombinasi tersebut sebagai FFEE, yang ditandai dengan dua pasak berwarna, dengan skor 4, menjadikannya langkah pertama dengan nilai terbaik. Maka kita dapat hitung sebagai akar pertama dan akan ditelusuri untuk selanjutnya.

Langkah kedua adalah memperluas akar berdasarkan posisi Dari akar yang dipilih, kita dapat mulai mengembangkannya berdasarkan posisi. Node baru yang pertama adalah AFEE yang ditandai dengan pasak putih dan pasak berwarna dengan skor 3. Ini terbukti lebih rendah dari simpul sebelumnya, yang artinya simpul dengan kombinasi AFEE akan dipangkas. Berpindah ke node baru berikutnya, FAEE, yang diberi skor 4, berarti skor yang didapatkan oleh node FAEE sama dengan skor saat ini. Sehingga node ini akan terus kita kembangkan, dengan syarat pada tahap selanjutnya skor harus lebih tinggi dari skor saat ini.

Langkah ketiga adalah memperluas simpul dengan warna. Dari node yang dipilih, kita bisa mulai mengembangkannya berdasarkan warna. Node baru pertama yang dihasilkan pada tahap ini adalah FBEE. Node ini ditandai dengan dua pasak berwarna dengan skor 4, yang tidak memenuhi kondisi baru yang diberikan. Node ini akan dipangkas dan melanjutkan ke node berikutnya, yaitu FCEE. Node ini mendapatkan skor total 6. Skor yang didapat lebih besar dari tiga kombinasi

warna sebelumnya. Karena skor yang didapat lebih besar dari skor-skor sebelumnya dan skor saat ini(skor 4), jadi node ini akan diperluas. Node berikutnya akan diperluas berdasarkan posisinya, dan kondisi tambahan tidak akan diterapkan.

Langkah keempat adalah memperluas node berdasarkan posisinya. Dari node FCEE, akan memeriksa setiap posisi. Node pertama yang akan dibangkitkan adalah ACEE.Skor dari kombinasi ACEE tidak akan mendapatkan skor yang lebih tinggi dari atau sama dengan skor yang telah diperoleh sebelumnya. Maka node ACEE dihapus dan dilanjutkan kepada node berikutnya, yaitu FAEE yang mendapatkan skor yang sama dengan ACEE, maka node FAEE dihapus dan tidak dilanjutkan. Di node berikutnya, FCAE, kita menemukan node yang ditandai dengan tiga pasak berwarna, dengan skor 6. Hal ini sama dengan skor saat ini. Ini akan menjadi node berikutnya untuk dilanjutkan ke tahap berikutnya.

Langkah kelima adalah memperluas root dengan warna. Pada langkah ini diberlakukan ketentuan tambahan bahwa skor harus lebih tinggi dari skor sebelumnya. Node saat ini, FCAE, akan diperluas di mana warna pada posisi ketiga akan berubah dengan setiap warna. Pada tahap ini solusi ditemukan pada uji coba pertama yaitu FCBE. Karena solusinya sudah ditemukan, pencarian akan dihentikan, dan node tidak akan diperluas lebih jauh.

IV. Kesimpulan

Dengan algoritma *backtracking dan mencari probabilitas suatu kejadian* memungkinkan kita untuk menelaah kejadian yang telah terjadi dan melihat peluang berikutnya. Ini sangat efisien apabila diterapkan pada jenis masalah yang tepat. Dengan sedikit penyesuaian, algoritma ini dapat menyelesaikan persoalan dalam dunia nyata, salah satunya adalah game *mastermind*.

References

[1] https://codebreaker-mastermind-superhirn.blogspot.com/2012/07/bulls-and-cows-mastermind-superhirn.html pada 9 Mei 2021

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Pohon-2020-Bag2.pdf diakses pada 9 Mei 2021

[3] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Brute-Force-(2021)-Bag1.pdf diakses 9 Mei 2021

[4] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian1.pdf diakses 9 Mei 2021

[5] https://programmingpraxis.com/2009/11/20/master-mind-part-2/ diakses pada 9 Mei 2021

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, 11 Mei 2021

Christian Gunawan, 13519199

PRANALA VIDEO

https://youtu.be/mcB082aqKFs