Penerapan Algoritme Depth First Search *dalam Menyelesaikan* Kakurasu Puzzle

Izharulhaq 13518092
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13518092@std.stei.itb.ac.id

Abstrak—Kakurasu puzzle merupakan puzzle logika yang dimainkan dalam sebuah tabel berbentuk persegi dan di setiap sisi persegi tersebut terdapat angka-angka. Angka-angka ini digunakan sebagai petunjuk untuk menentukan sel mana saja yang harus diwarnai (atau ditandai). Dalam makalah ini akan dijelaskan cara menyelesaikan Kakurasu puzzle dengan menggunakan algoritme *Depth First Search* (DFS).

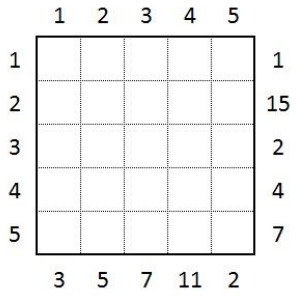
Kata kunci*—Kakurasu, *Depth First Search, Backtracking

I. Pendahuluan

Pada zaman sekarang ini, siapa yang tidak mengenal *puzzle*? permainan teka-teki yang memerlukan logika untuk diselesaikan ini sudah berhasil menarik perhatian banyak orang sehingga banyak sekali media seperti majalah dan koran yang disisipkan *puzzle* di dalamnya.

Hingga saat ini sudah banyak sekali jenis *puzzle* yang berhasil diciptakan, baik itu sesuatu yang orisinal atau merupakan turunan yang lebih dulu ada. Beberapa contoh dari jenis *puzzle* adalah *crossword*, sudoku, 15 puzzle, dan kakurasu. Dalam makalah ini jenis *puzzle* yang akan dibahas adalah kakurasu.

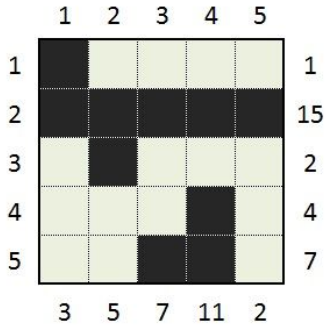
Kakurasu adalah sebuah *puzzle* yang berasal dari Jepang yang memerlukan kemampuan matematika dasar dalam memainkannya. Kakurasu berbentuk sebuah tabel persegi yang memiliki angka-angka pada bagian luar sisi-sisinya yang berfungsi sebagai petunjuk[1].



Gambar 1.1 - Kakurasu berukuran 5 × 5 yang masih kosong.
Sumber: http://curiouscheetah.com/Content/PuzzleImages/Kakurasu_initial.jpg

Sebelum membahas cara memainkan kakurasu, ada baiknya kita memahami maksud dari angka-angka pada kakurasu. Pada gambar 1 di atas, angka-angka pada sisi bagian atas dan sisi bagian kiri berturut-turut menyatakan nilai dari sel yang ada di bawahnya secara horizontal dan vertikal berturut-turut. Sebagai contoh, sel pada baris pertama kolom kedua memiliki nilai 2 secara horizontal dan memiliki nilai 1 secara vertikal. Sementara itu, angka-angka pada sisi bagian kanan dan bawah menyatakan *goal* dari setiap baris dan kolom berturut-turut. Sebagai contoh, 15 merupakan *goal* dari baris kedua dan 7 merupakan *goal* dari kolom ketiga.

Cara memainkan kakurasu adalah dengan menghitamkan sel-sel yang dirasa perlu sedemikian sehingga jumlah nilai dari seluruh sel yang berwarna hitam memenuhi *goal* setiap baris dan setiap kolom. Berikut adalah solusi dari kakurasu pada gambar 1.1.



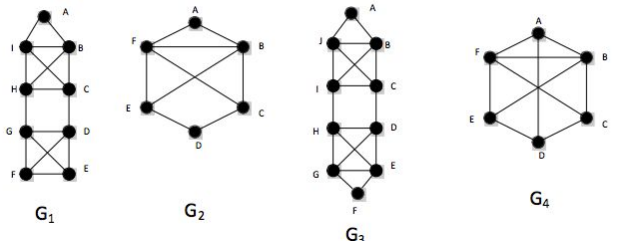
Gambar 1.2 - Solusi dari kakurasu pada gambar 1.
Sumber: http://curiouscheetah.com/Content/PuzzleImages/Kakurasu_solution.jpg

II. Teori Singkat

A. *Graf*

Graf merupakan struktur data diskrit yang biasa digunakan untuk menggambarkan objek-objek diskrit serta hubungan di antara objek-objek tersebut.

Graf, biasa dituliskan dengan G = (V, E), merupakan gabungan dari himpunan simpul (V) yang biasa digunakan untuk merepresentasikan objek diskrit serta himpunan sisi (E) yang biasa digunakan untuk menyatakan hubungan sepasang simpul.



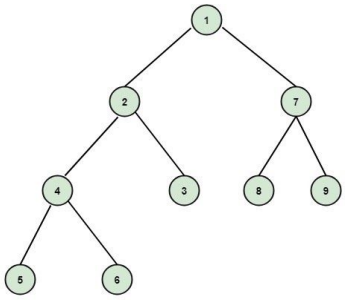
Gambar 1.3 - Contoh-contoh graf.
Sumber: https://i.stack.imgur.com/TERMr.png

Dalam graf terdapat konsep ketetanggaan. Maksudnya, jika terdapat suatu simpul X, maka semua simpul yang berhubungan langsung dengan X disebut sebagai tetangga dari X. Sebagai contoh, simpul C pada graf G_2 dari gambar 3 di atas memiliki tetangga simpul B, F, dan D. Jika suatu simpul tidak memiliki tetangga, maka simpul tersebut dinamakan simpul terpencil[2].

B. *Pohon*

Pada gambar 3, semua contoh graf yang disajikan setidaknya memiliki satu buah sirkuit. Artinya, setidaknya terdapat sebuah himpunan simpul yang sedemikian sehingga semua simpul pada himpunan tersebut dapat dikunjungi tepat sekali dan dan kembali lagi ke simpul yang pertama kali dikunjungi. Sebagai contoh, himpunan simpul {A, B, F}pada G_2 membentuk sebuah sirkuit karena dari simpul A simpul B dan F dapat dikunjungi tepat sekali dan kembali lagi ke simpul A.

Terdapat variasi lain dari graf yang dinamakan pohon atau biasa disebut *tree*. Pohon adalah graf yang tidak memiliki sirkuit di dalamnya. Berikut adalah contoh pohon.



Gambar 1.4 - Contoh pohon.
Sumber: https://static.javatpoint.com/tutorial/dms/images/discrete-mathematics-binary-trees3.jpg

Salah satu jenis dari pohon adalah yang sering digunakan pohon berakar (*rooted tree*). Pohon berakar adalah pohon yang salah satu simpulnya dianggap sebagai simpul awal (akar) dan setiap sisinya diberikan arah. Dalam penulisan pohon berakar, arah dari tiap sisi tidak perlu dinyatakan secara eksplisit. Gambar 1.4 merupakan contoh dari pohon berakar.

Dalam sebuah pohon berakar, unsur-unsur yang pasti ada adalah sebagai berikut:

1. *Root*. *Root* atau akar adalah simpul pada pohon yang menjadi simpul awal dari sebuah pohon. Akar dari pohon pada gambar 4 adalah simpul 1.
2. *Child/children*. Apabila terdapat beberapa simpul menjadi tetangga dari sebuah simpul, misal v, maka simpul-simpul tersebut adalah *children* atau anak dari simpul v. Biasanya, *children* dituliskan di bawah simpul v. Simpul 2 dan 7 merupakan *children* dari simpul 1.
3. *Parent*. *Parent* atau orangtua adalah simpul yang menjadi tetangga umum dari *children*. Biasanya, *parent* dituliskan di atas *children*. Simpul 1 merupakan *parent* dari simpul 2 dan 7.
4. *Path*. *Path* atau lintasan adalah banyaknya simpul yang dikunjungi dari suatu simpul ke simpul lain yang ada di bawahnya. *Path* dari simpul 2 ke simpul 6 adalah 2, 4, 6 dengan panjang *path* 2.
5. *Level*. *Level* atau aras adalah panjang path dari simpul akar ke simpul lainnya. *Level* dari simpul 2 dan 7 adalah 1, *level* dari simpul 5 dan 6 adalah 3.
6. *Depth*. *Depth* atau kedalaman adalah *level* tertinggi yang ada pada sebuah pohon. Depth dari pohon pada gambar 4 adalah 3.
7. *Degree*. *Degree* atau derajat merupakan banyaknya *children* yang dimiliki oleh suatu simpul. Simpul 1 memiliki *degree* 2. Sementara itu, simpul 6 memiliki *degree* 0.
8. *Sibling*. *Sibling* atau saudara adalah simpul-simpul yang memiliki level yang sama. *Sibling* dari simpul adalah simpul 7 dan begitu pula sebaliknya.
9. *Leaves*. *Leaves* atau daun adalah simpul-simpul yang tidak memiliki *children*. Simpul 5 dan 6 merupakan contoh dari *leaves.*
10. *Subtree*. *Subtree* atau upapohon adalah pohon lain yang berada dan menjadi bagian dari sebuah pohon yang lebih besar. Dalam pohon pada gambar 1.4, terdapat subtree yang simpul akarnya adalah simpul 2 [3].

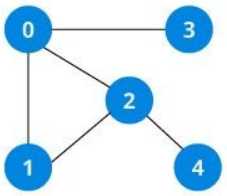
Salah satu contoh penggunaan pohon berakar adalah pohon pencarian ruang status. Dalam pohon pencarian ruang status, setiap simpul mewakilkan status yang mungkin dalam suatu permasalahan. Simpul akar dari pohon ruang status merupakan kondisi awal dari sebuah permasalahan dan *children* dari setiap simpul merupakan status-status yang mungkin diturunkan dari simpul tersebut.

C. *Algoritme* Depth First Search

Algoritme *Depth First Search* (DFS) merupakan salah satu jenis algortime dalam penelusuran graf tanpa informasi tambahan (*blind search*).

Cara kerja dari algoritme DFS adalah sebagai berikut.

1. Kunjungi salah satu simpul yang berperan sebagai simpul awal, misal v.
2. Kunjungi salah satu tetangga dari simpul awal, misal w.
3. Ulangi langkah pertama dengan menggunakan simpul w hingga seluruh simpul sudah pernah dikunjungi.
4. Jika suatu simpul tidak memiliki tetangga atau tetangganya sudah dikunjungi semua sementara belum semua simpul dikunjungi maka DFS akan *backtrack* ke simpul sebelumnya hingga didapat simpul yang tetangganya belum dikunjungi[4].



Gambar 1.5 - Contoh graf.
Sumber:
https://cdn.programiz.com/sites/tutorial2program/files/graph-dfs-step-0.jpg

Sebagai contoh, perhatikan gambar 4 di atas. Asumsikan DFS dimulai dari simpul 0 dan simpul yang memiliki angka lebih kecil terlebih dahulu dikunjungi. DFS akan mengunjungi simpul 0, simpul 1, simpul 2, simpul 4, *backtrack* hingga simpul 0 dan mengunjungi simpul 3.

D. *Fungsi Rekursif*

Fungsi rekursif adalah fungsi yang memanggil dirinya dalam pendefinisian fungsi tersebut. Fungsi rekursif terdiri dari dua bagian, yaitu:

1. Basis merupakan bagian yang secara eksplisit menyatakan nilai dari fungsi tersebut dan menjadi batas rekurensi.
2. Rekurens, merupakan bagian yang akan dijalankan ketika kondisi untuk menuju basis tidak terpenuhi. Pada bagian ini fungsi akan dipanggil terus-menerus hingga mencapai kondisi basis. Pada rekurens harus terdapat bagian yang dapat mengarahkan fungsi menuju kondisi basis. Jika tidak, maka akan terjadi pengulangan tak terhingga (*infinity loop*)[5].

Contoh umum yang biasa digunakan adalah fungsi faktorial.

$$n! = \begin{cases} 1 & n = 0 \\ n \times (n-1)! & n > 0 \end{cases}$$

Pada fungsi faktorial di atas, bagian atas merupakan basis dan bagian bawah adalah rekurens. Jika n > 0, maka fungsi faktorial akan terus dipanggil hingga didapat n = 0.

Fungsi rekursif juga dipakai dalam algoritme DFS. Dari penjelasan cara kerja DFS sebelumnya, kondisi basis dari DFS adalah tidak ada lagi simpul yang belum dikunjungi sementara kondisi rekurensnya adalah ketika masih ada simpul yang belum dikunjungi.

E. *Algoritme Runut-Balik*

Algoritme runut-balik (*backtrack*) merupakan perbaikan dari algoritme *exhaustive search* yang lebih efisien dan terstruktur. Berbeda dengan *exhaustive search* yang menelusuri seluruh kemungkinan yang ada, algoritme runut-balik hanya menelusuri kemungkinan-kemungkinan yang dapat menuju solusi.

Dalam penerapan algoritme runut-balik, unsur-unsur berikut dipastikan ada.

1. Solusi, merupakan solusi dari permasalahan yang sedang dipecahkan. Solusi dinyatakan dalam bentuk *tuple* yang berisikan status-status yang telah dilalui untuk menuju solusi.
2. Fungsi pembangkit, yaitu fungsi yang digunakan untuk membangkitkan status yang menjadi komponen dari kandidat solusi.
3. Fungsi pembatas, yaitu fungsi yang digunakan untuk mengevaluasi apakah kandidat solusi yang telah dicapai mengarah ke solusi atau tidak. Jika iya, maka status selanjutnya akan dibangkitkan. Jika tidak, maka kandidat solusi yang telah dicapai akan dibuang[6].

Sebagaimana yang telah dijelaskan sebelumnya, DFS memanfaatkan konsep *backtrack* dalam penelusurannya, yaitu ketika suatu simpul tidak memiliki tetangga yang belum dikunjungi maka DFS akan melakukan backtrack ke simpul sebelumnya yang memiliki tetangga yang belum dikunjungi.

III. Penyelesaian Kakurasu Menggunakan Depth First Search

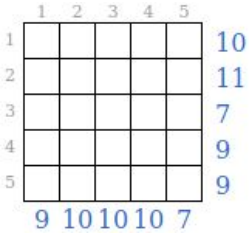
Dalam makalah ini metode penyelesaian kakurasu puzzle yang akan diterapkan adalah dengan menyelesaikan baris per baris. Akan tetapi, metode ini juga dapat digunakan dengan menyelesaikan kolom per kolom. Langkah-langkah yang akan diterapkan adalah sebagai berikut.

1. Membangkitkan solusi-solusi yang mungkin untuk baris pertama dengan batasan nilai yang ada sama dengan *goal* baris tersebut serta tidak melebihi *goal* dari setiap kolom. Jika terdapat lebih dari satu solusi, maka yang didahulukan adalah yang memiliki elemen paling sedikit. Jika sama, gunakan solusi yang memiliki angka terbesar lebih dahulu.
2. Ulangi langkah pertama untuk baris kedua dan seterusnya.
3. Jika pada suatu baris langkah pertama tidak menghasilkan solusi apapun, maka lakukan *backtrack*

ke baris sebelumnya dengan menggunakan solusi yang lain.

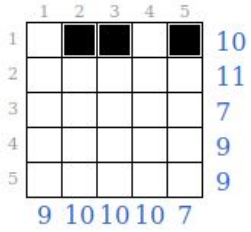
4. Jika semua baris sudah terisi dan goal pada setiap baris dan kolom terpenuhi maka dianggap selesai.

Sebagai contoh, kakurasu *puzzle* yang akan diselesaikan adalah kakurasu berukuran 5 × 5 yang dapat diperoleh dari https://www.puzzle-kakurasu.com/.



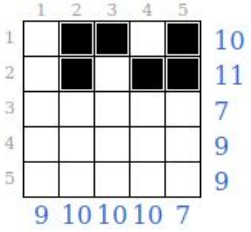
Gambar 3.1 - Kakurasu berukuran 5 × 5 yang masih kosong

Pada baris pertama, solusi-solusi yang dapat dipilih adalah [1, 2, 3, 4], [1, 4, 5], dan [2, 3, 5]. Pertama-tama, gunakan solusi [2, 3, 5] sehingga didapat



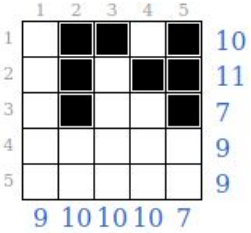
Gambar 3.2 - Baris pertama menggunakan solusi [2, 3, 5]

Pada baris kedua, solusi-solusi yang dapat dipilih adalah [1, 2, 3, 5] dan [2, 4, 5]. Gunakan solusi [2, 4, 5] terlebih dahulu sehingga didapat



Gambar 3.3 - Baris kedua menggunakan solusi [2, 4, 5]

Pada baris ketiga, solusi-solusi yang dapat dipilih adalah [1, 2, 4], [2, 5], dan [3, 4]. Gunakan solusi [2, 5] terlebih dahulu sehingga didapat

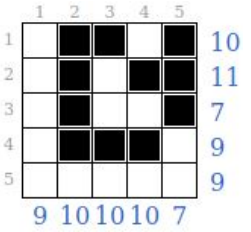


Gambar 3.4 - Baris ketiga menggunakan solusi [2, 5]

Pada baris keempat, solusi yang dapat dipilih hanyalah [2, 3, 4]. Solusi [4, 5] dan [1, 3, 5] tidak dibangkitkan karena:

1. Jika solusi [4, 5] digunakan, maka kolom kelima akan bernilai 10 yang tentu melebihi *goal* kolom kelima yaitu 7.
2. Jika solusi [1, 3, 5] digunakan, maka kolom kelima akan bernilai 10 yang tentu melebihi *goal* kolom kelima yaitu 7.

Oleh karena itu, gunakan solusi [2, 3, 4] sehingga didapat

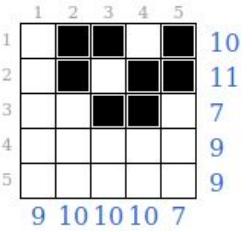


Gambar 3.5 - Baris keempat menggunakan solusi [2, 3, 4]

Pada baris kelima, tidak ada solusi yang mungkin digunakan. Solusi [1, 3, 5], [2, 3, 4], dan [4, 5] tidak dibangkitkan karena:

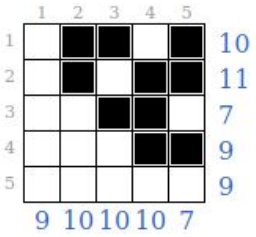
1. Jika solusi [1, 3, 5] digunakan, maka kolom kelima akan bernilai 11 yang tentu melebihi *goal* kolom kelima yaitu 7.
2. Jika solusi [2, 3, 4] digunakan, maka kolom kedua akan bernilai 15 dan kolom keempat akan bernilai 11 yang tentu melebihi *goal* kolom kedua dan keempat yaitu 10.
3. Jika solusi [4, 5] digunakan, maka kolom keempat dan kolom kelima akan bernilai 11 yang tentu melebihi *goal* kolom keempat dan kelima yaitu 10 dan 7.

Karena tidak terdapat solusi pada baris kelima, maka lakukan *backtrack* ke baris sebelumnya yaitu baris keempat. Akan tetapi, pada baris keempat hanya ada satu solusi yaitu [2, 3, 4] dan sudah digunakan. Oleh karena itu lakukan *backtrack* lagi ke baris ketiga. Pada baris ketiga salah satu solusi yang belum digunakan adalah [3, 4] sehingga dengan menggunakan solusi tersebut didapat



Gambar 3.6 - Baris ketiga menggunakan solusi [3, 4]

Pada baris keempat, kini solusi-solusi yang dapat dipilih adalah [1, 3, 5], [2, 3, 4], dan [4, 5]. Gunakan solusi [4, 5] terlebih dahulu sehingga didapat

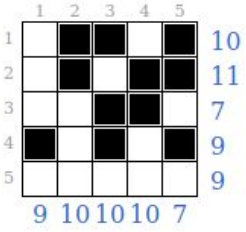


Gambar 3.7 - Baris keempat menggunakan solusi [4, 5]

Pada baris kelima, tidak ada solusi yang mungkin digunakan. Solusi [1, 3, 5], [2, 3, 4], dan [4, 5] tidak dibangkitkan karena:

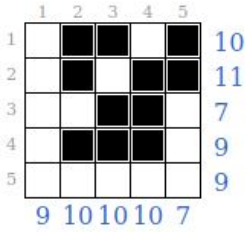
1. Jika solusi [1, 3, 5] digunakan, maka kolom kelima akan bernilai 12 yang tentu melebihi *goal* kolom kelima yaitu 7.
2. Jika solusi [2, 3, 4] digunakan, maka kolom keempat akan bernilai 14 yang tentu melebihi *goal* kolom keempat yaitu 10.
3. Jika solusi [4, 5] digunakan, maka kolom keempat dan kolom kelima akan bernilai 14 dan 12 secara berturut-turut yang tentu melebihi *goal* kolom keempat dan kelima yaitu 10 dan 7.

Karena tidak terdapat solusi pada baris kelima, maka lakukan *backtrack* ke baris sebelumnya yaitu baris keempat. Salah satu solusi yang belum digunakan adalah [1, 3, 5] sehingga dengan menggunakan solusi tersebut didapat



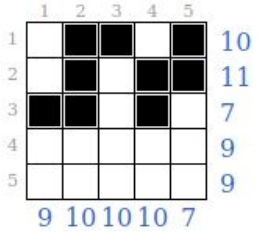
Gambar 3.8 - Baris keempat menggunakan solusi [1, 3, 5]

Pada baris kelima, lagi-lagi tidak ada solusi yang mungkin digunakan. Oleh karena itu, lakukan backtrack kembali ke baris keempat untuk menggunakan solusi terakhir yaitu [2, 3, 4] sehingga didapat



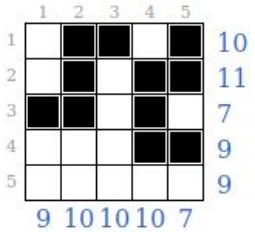
Gambar 3.9 - Baris keempat menggunakan solusi [2, 3, 4]

Pada baris kelima, lagi-lagi tidak ada solusi yang mungkin digunakan. Oleh karena itu, lakukan backtrack kembali ke baris keempat. Akan tetapi, semua solusi pada baris keempat sudah dicoba sehingga lakukan *backtrack* lagi ke baris ketiga. Solusi pada baris ketiga yang belum digunakan adalah [1, 2, 4] sehingga dengan menggunakan solusi tersebut didapat



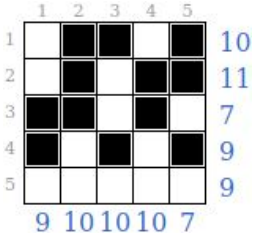
Gambar 3.10 - Baris ketiga menggunakan solusi [1, 2, 4]

Pada baris keempat, solusi yang mungkin adalah [1, 3, 5], [2, 3, 4], dan [4, 5]. Gunakan solusi [4, 5] terlebih dahulu sehingga didapat



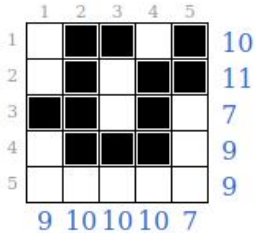
Gambar 3.11 - Baris keempat menggunakan solusi [4, 5]

Pada baris kelima, lagi-lagi tidak ada solusi yang mungkin digunakan. Oleh karena itu, lakukan *backtrack* kembali ke baris keempat untuk menggunakan solusi [1, 3, 5] sehingga didapat



Gambar 3.12 - Baris keempat menggunakan solusi [1, 3, 5]

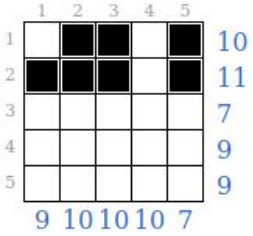
Pada baris kelima, lagi-lagi tidak ada solusi yang mungkin digunakan. Oleh karena itu, lakukan *backtrack* kembali ke baris keempat untuk menggunakan solusi [2, 3, 4] sehingga didapat



Gambar 3.13 - Baris keempat menggunakan solusi [2, 3, 4]

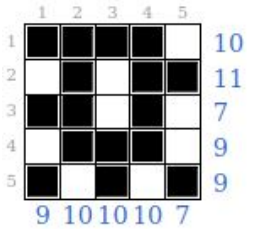
Pada baris kelima, lagi-lagi tidak ada solusi yang mungkin digunakan. Oleh karena itu, lakukan *backtrack* kembali ke baris keempat. Akan tetapi, semua solusi sudah pernah digunakan sehingga lakukan kembali *backtrack* ke baris

ketiga. Akan tetapi, pada baris ketiga semua solusi juga sudah digunakan sehingga lakukan kembali *backtrack* ke baris kedua. Solusi yang belum digunakan adalah [1, 2, 3, 5] sehingga didapat



Gambar 3.14 - Baris kedua menggunakan solusi [1, 2, 3, 5]

Ulangi langkah-langkah yang sudah dijelaskan sebelumnya hingga solusi ditemukan. Berikut adalah solusi yang dicari.



Gambar 3.15 - Solusi dari kakurasu pada gambar 6

IV. Analisis Algoritme

Dalam penjabaran di atas, kakurasu puzzle dapat direpresentasikan sebagai sebuah pohon ruang status dengan setiap simpul merepresentasikan solusi setiap baris pada *level* simpul tersebut serta baris-baris sebelumnya . Root dari pohon tersebut adalah kondisi ketika puzzle masih kosong (belum ada baris yang diisi). Sementara itu, children dari setiap simpul merupakan solusi setiap baris selanjutnya digabungkan dengan solusi baris yang sudah ada sebelumnya.

Dalam permasalahan ini, pohon ruang status memiliki kedalaman yang pasti yaitu sebanding dengan ukuran *puzzle* tersebut. Selain itu, solusi yang dicari pasti berada di simpul terbawah pada pohon ruang status sehingga kedalaman maksimum untuk mencapai solusi (d) dengan kedalaman maksimum pencarian (m) pada pohon ruang status ini adalah sama.

Sebagaimana yang telah dijelaskan sebelumnya, algoritme DFS menggunakan konsep algoritme *backtrack* dalam penelusurannya. Oleh karena itu, algoritme DFS tentu memiliki unsur-unsur dari algoritme *backtrack*. Unsur-unsur tersebut dalam permasalahan ini adalah sebagai berikut.

1. Unsur solusi dalam permasalahan ini adalah tuple berukuran *n* dengan *n* adalah ukuran *puzzle* yang setiap komponennya adalah kombinasi sel-sel yang dihitamkan dari baris pertama hingga baris ke-n.
2. Unsur fungsi pembangkit dalam permasalahan ini adalah fungsi pengulangan yang menghasilkan kandidat solusi yang mungkin pada setiap baris dalam *puzzle*.
3. Unsur fungsi pembatas dalam permasalahan ini adalah fungsi yang digunakan untuk mengecek apakah terdapat kandidat solusi yang mungkin untuk baris selanjutnya. Jika tidak terdapat kandidat solusi yang mungkin, maka tuple kandidat solusi yang sudah tercapai dianggap tidak mengarah ke solusi dan akan dibuang.

Berdasarkan penjelasan penggunaan algoritme DFS dalam menyelesaikan kakurasu *puzzle* di atas, dapat diketahui:

1. Algoritme ini pasti akan menuju ke solusi (*completeness*). Dalam persoalan yang dibahas, jumlah maksimal kombinasi yang ada terbatas dan tidak ada *state* yang berulang (*state* yang sama dihasilkan dari dua *state* yang berbeda) sehingga solusi pasti bisa dicapai.
2. Dalam permasalahan ini algoritme menghasilkan hasil yang optimal. Hal ini karena solusi pasti berada di kedalaman maksimum pohon ruang status.
3. Kompleksitas waktu algoritme ini adalah $O(b^d)$.
4. Kompleksitas ruang algoritme ini adalah O(bd).

V. Kesimpulan

Kakurasu puzzle dapat diselesaikan dengan menggunakan algoritme *depth first search* (DFS). Akan tetapi, algoritme ini hanya mampu digunakan untuk puzzle yang berukuran tidak terlalu besar karena jika puzzle berukuran sangat besar maka waktu yang diperlukan akan meningkat drastis karena kompleksitasnya yang merupakan fungsi polinomial.

Apabila dibandingkan dengan metode *exhaustive search*, algoritme ini akan jauh lebih mangkus karena exhaustive search akan membangkitkan seluruh kombinasi sel hitam yang mungkin sementara algoritme ini hanya membatasi pada kombinasi sel-sel yang mungkin menuju ke solusi.

Link Video Youtube

Video penerapan algoritme ini dalam kanal Youtube ada di https://youtu.be/DkV0nkD2Plo

Ucapan Terima Kasih

Penulis mengucapkan syukur kepada Allah Subhanahu wa ta'ala karena hanya atas rahmat-Nya penulis dapat menyelesaikan makalah ini. Selain itu, penulis juga mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, MT., Ibu Nur Ulfa Maulidevi ST., M.Sc., da Ibu Dr. Masayu Leylia Khodra, ST., MT. sebagai dosen pengajar IF2211 Strategi Algoritma yang telah memberikan ilmu-ilmu bermanfaat yang dapat membantu penyelesaian penulisan makalah ini.

Melalui tugas penulisan makalah ini juga penulis dapat mengaplikasikan dan mengembangkan ilmu yang telah didapat, terutama mengenai algoritme *Depth First Search*, dalam salah satu kegiatan dalam kehidupan sehari-hari yaitu menyelesaikan *puzzle*.

Daftar Pustaka

[1] Hartzer, Paul. "Kakurasu". http://curiouscheetah.com/Content/PuzzleImages/Kakurasu_initial.jpg, diakses pada Sabtu, 25 April 2020 pukul 20.54 WIB.

[2] Munir, Rinaldi. 2015. "Graf". http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2015-2016/Graf%20(2015).pdf, diakses pada Minggu, 26 April 2020 pukul 01.49 WIB.

[3] Munir, Rinaldi. 2013. "Pohon". http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2013-2014/Pohon%20(2013).pdf, diakses pada Minggu, 26 April 2020 pukul 07.38 WIB.

[4] Munir, Rinaldi. 2020. "Breadth/Depth First Search". http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/BFS-dan-DFS-(2020).pdf, diakses pada Minggu, 26 April 2020 pukul 02.07 WIB.

[5] Munir, Rinaldi. 2015. "Rekursi dan Relasi Rekurens". http://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2015-2016/Rekursi%20dan%20Relasi%20Rekurens%20(2015).pdf, diakses pada Minggu, 26 April 2020 pukul 02.41 WIB.

[6] Munir, Rinaldi. 2020. "Algoritma Runut-Balik (Backtracking)". http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Algoritma-Runut-balik-(2020).pdf, diakses pada Minggu, 26 April 2020 pukul 03.52 WIB.

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.

Tangerang Selatan, 24 April 2020

Ttd

Izharulhaq
13518092