Pemanfaatan Algoritma Runut-Balik dalam Menyelesaikan Puzzle NeurOn dalam Permainan Logical Cell

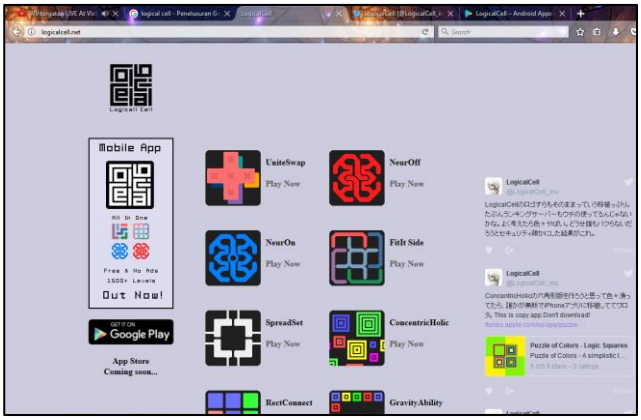
Adrian Mulyana Nugraha 13515075
Program Studi Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515075@std.stei.itb.ac.id

***Abstrak*--- Salah satu cara dalam mengasah kemampuan berpikir logika otak adalah dengan bermain permainan dengan genre *puzzle.* Dalam permainan *puzzle,* pemain harus mencari solusi dari suatu permasalahan dengan mencoba berbagai kemungkinan solusi yang ada. Dalam makalah ini, penulis akan menjelaskan bagaimana dengan algoritma Runut-balik seseorang dapat menyelesaikan permainan *puzzle*.**

***Kata Kunci*--- Puzzle, logika, Runut-balik, solusi**

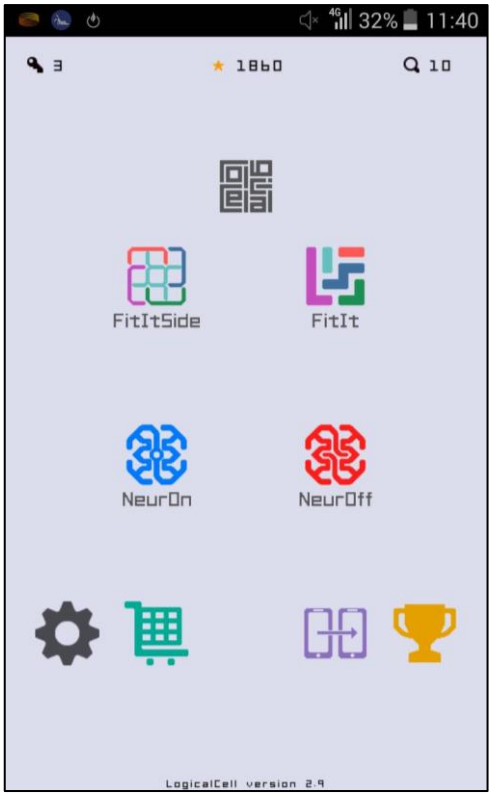
I. PENDAHULUAN

Logical Cell merupakan sebuah permainan ber-*genre puzzle*, yang dibuat oleh LogicalCell_INU, kelompok developer *indie* dari Jepang. Permainan ini pertama diluncurkan pada April 2014, dan sekarang tersedia pada komputer dan gawai Android. Tujuan dari permainan ini adalah menyelesaikan berbagai *puzzle* yang ada sebanyak mungkin.



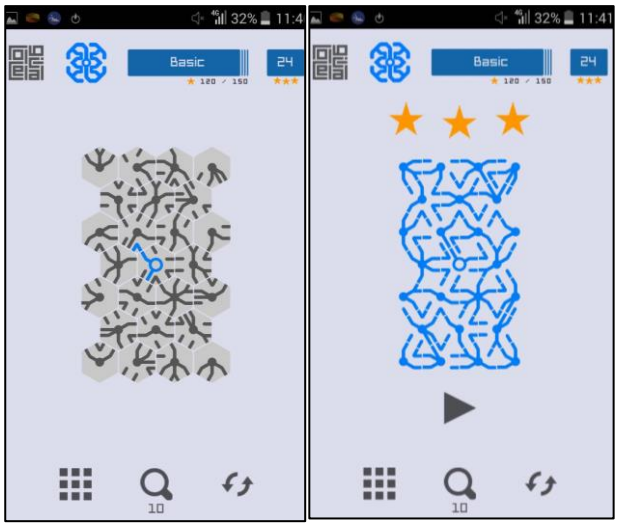
Gambar 1.1. Tampilan antarmuka Logical Cell pada komputer (sumber: logicalcell.net)

Ada berbagai mode permainan yang tersedia pada versi komputer, dan dari mode-mode tersebut, ada 4 mode permainan yang dimasukkan dalam versi Android, yakni FitItSide dan FitIt (mencocokkan bentuk pada sebuah lokasi), NeurOn (menghubungkan seluruh jalur), dan NeurOff (memisahkan seluruh jalur). Terdapat setidaknya 250 level dalam setiap mode, dan pengguna dapat menggunakan bantuan untuk menyelesaikan *puzzle* serta membeli level-level tambahan dalam setiap mode. Setiap level yang sukses akan memberikan 3 bintang. Setiap 50 bintang yang diraih akan memberikan 1 bantuan. Hingga saat ini, Logical Cell telah diunduh dan dimainkan oleh lebih dari 100 ribu orang di gawai Android, dan dimainkan oleh jutaan orang di komputer.



Gambar 1.2. Tampilan antarmuka Logical Cell di gawai Android (sumber: aplikasi Logical Cell)

Salah satu mode permainan yang akan penulis bahas adalah mode NeurOn. Terinspirasi dari ketersambungan pada sel otak (neuron), mode *puzzle* ini menantang pemain untuk menyambungkan seluruh jalur dalam *node* yang tersedia. Pemain hanya dapat memutar setiap *node* ke kanan, tapi tidak memindahkan. Tidak ada batas waktu dalam mode ini, dan level selesai ketika seluruh jalur dalam *node* berhasil tersambung. *Node* dapat berbentuk segitiga, segiempat, dan segienam. Tidak ada jalur yang bersifat siklik dalam *puzzle* ini. Seluruh jalur harus tersembung menjadi satu jalur, tidak boleh membuat jalur yang terpisah.



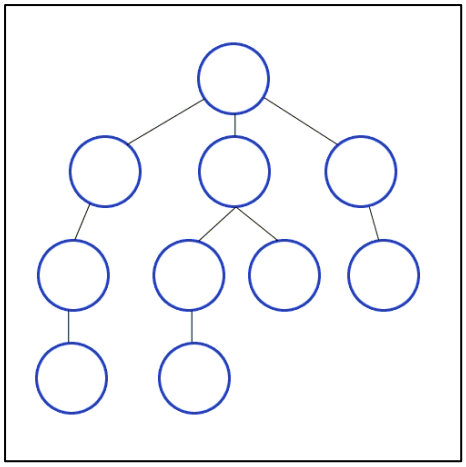
Gambar 1.3. *Puzzle* NeurOn yang belum selesai (kiri)
Gambar 1.4. *Puzzle* NeurOn yang sudah selesai (kanan)
(sumber: aplikasi Logical Cell)

II. LANDASAN TEORI

Strategi algoritma yang menjadi dasar dari makalah ini adalah algoritma Runut-balik (*Backtracking*).

Runut-balik merupakan algoritma pencarian solusi yang berbasis DFS (*Depth-First Search*). DFS sendiri merupakan algoritma pencarian solusi dengan penelusuran terhadap simpul-simpul pohon ruang status pada satu sisi terlebih dahulu hingga mencapai kedalaman maksimal, kemudian diulang kembali dengan sisi yang lain. Algoritma DFS pertama dikemukakan oleh Charles Pierre Trémaux, seorang matematikawan Perancis, pada abad ke-19 sebagai sebuah cara untuk mencari jalan keluar dari labirin.

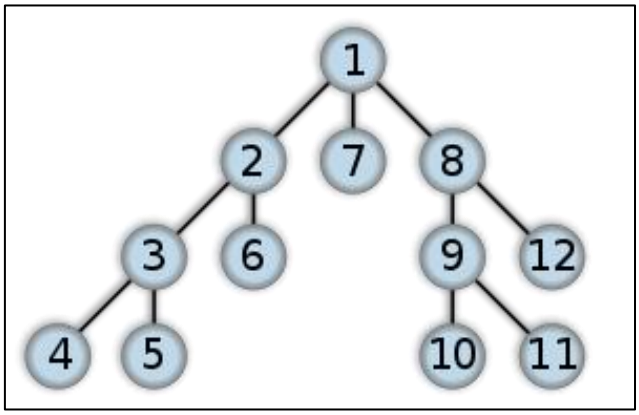
Sebelum menjelaskan lebih jauh, perlu dijelaskan mengenai pohon ruang status terlebih dahulu. Pohon ruang status merupakan sebuah pohon yang terdiri dari status persoalan, dimana akar dari pohon menyatakan status awal (*initial state*) dari persoalan, sedangkan setiap daun mewakili status persoalan. Untuk daun yang merupakan solusi disebut sebagai status tujuan (*goal state*).



Gambar 2.1. Pohon Ruang Status (sumber: Wikipedia)

Pada kasus terburuk, algoritma DFS memiliki kompleksitas waktu $O(n+s)$, dimana n merupakan jumlah simpul pada pohon dan s merupakan jumlah sisi. Kompleksitas komputasional dari DFS adalah P-complete.

Jika dibandingkan dengan BFS (*Breadth-First Search*), algoritma DFS memiliki waktu pencarian yang lebih cepat dibanding BFS. Tetapi, untuk tingkat kedalaman pohon yang sangat besar, DFS tidak menguntungkan karena dalam pencarian DFS bisa saja proses penelusuran kedalaman tidak mencapai titik akhir, sedangkan BFS lebih memiliki kemungkinan menemukan solusi.



Gambar 2.2. Penelusuran pohon menggunakan DFS (sumber: Wikipedia)

Algoritma Runut-balik sendiri lebih hemat dan mangkus dari DFS maupun BFS, karena hanya mempertimbangkan pencarian yang mengarah pada solusi. Pencarian yang cenderung menjauh dari solusi diabaikan, sehingga secara otomatis mengurangi jumlah penelusuran yang dilakukan dengan perbedaan yang cukup signifikan.

Algoritma ini dikemukakan oleh Derrick H. Lehmer pada tahun 1950, dan banyak digunakan dalam penyelesaian permainan *puzzle* (contoh: Sudoku, TTS), pemrograman logika, dan penyelesaian masalah kombinatorial (contoh: Knapsack).

Dalam algoritma Runut-balik, ada 3 elemen yang menjadi kunci dari algoritma:

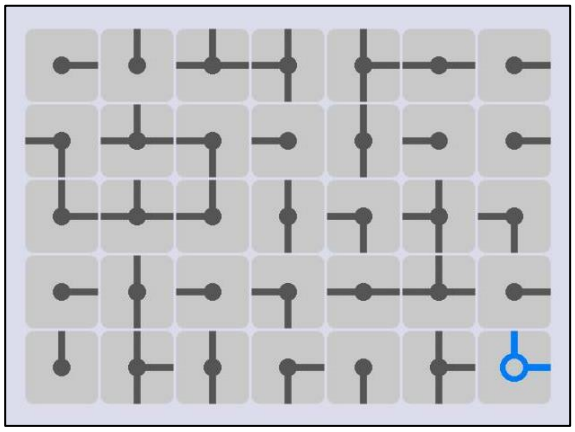
1. Solusi persoalan
   Solusi persoalan dinyatakan dengan *n-tuple*, yang menjelaskan setiap status pada pohon ruang status.
2. Fungsi pembangkit
   Fungsi pembangkit membentuk nilai dari komponen *tuple* solusi.
3. Fungsi pembatas
   Fungsi pembatas menentukan apakah lintasan yang ditempuh mendekati solusi atau tidak. Jika ya, maka penelusuran lintasan berikutnya akan dipertimbangkan. Jika tidak, lintasan tersebut "dimatikan".

Untuk pencarian solusi dengan Runut-balik, urutan pencarian adalah sebagai berikut:

1. Buat lintasan dari akar ke daun dengan prinsip DFS. Setiap simpul yang dibuat disebut sebagai simpul hidup. Simpul diberi nomor sesuai urutan pembuatan.
2. Panjang lintasan pada simpul hidup ditambah dengan pembentukan simpul anak, kemudian terapkan fungsi pembatas pada lintasan tersebut. Jika tidak mengarah pada solusi persoalan, simpul anak tersebut "dimatikan" (diabaikan dari pencarian dan panjang lintasannya tidak akan ditambah lagi)
3. Jika lintasan berakhir dengan simpul "mati", maka pencarian kembali dilakukan dengan membentuk simpul anak baru. Jika tidak ada lagi simpul anak yang dapat dibentuk, runut-balik ke simpul hidup terdekat. Simpul tersebut yang sekarang diperluas.
4. Pencarian berakhir ketika solusi ditemukan atau tidak ada lagi simpul hidup untuk dirunut-balik.

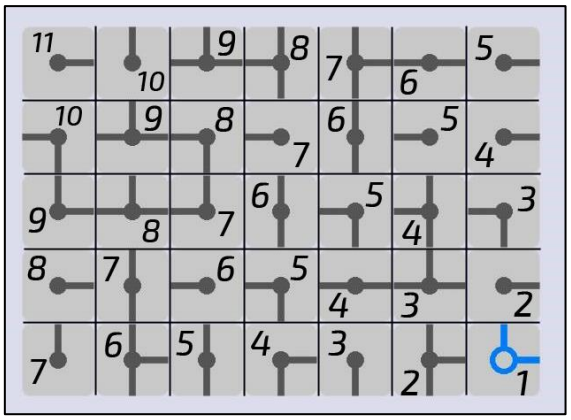
III. CARA PENYELESAIAN PUZZLE NEURON DENGAN BACKTRACKING

Salah satu algoritma yang dapat digunakan dalam menyelesaikan *puzzle* NeurOn adalah dengan DFS. Untuk itu, perlu dibuat pohon status dari permasalahan tersebut.



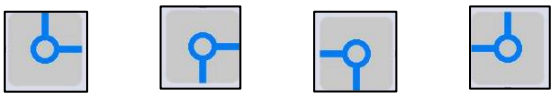
Gambar 3.1. Puzzle NeurOn (sumber: aplikasi Logical Cell)

Misalkan dalam persoalan diatas, status awal adalah posisi sebagai berikut. Simpul kedalaman 1 adalah *node* yang memiliki pusat sebuah lingkaran dan menyala biru (pojok kanan bawah). Simpul kedalaman 2 adalah *node-node* yang bersebelahan dengan *node* kedalaman 1. Simpul kedalaman 3 adalah *node-node* yang bersebelahan dengan *node* kedalaman 2, dan seterusnya. Dalam penggambaran *puzzle* tersebut, pohon ruang status akan menjadi sebagai berikut.



Gambar 3.2. Ilustrasi kedalaman pada pohon ruang status (sumber: aplikasi Logical Cell dengan ubahan)

Untuk status dari setiap node, pada status awal setiap node dianggap berada dalam status ke-1. Ada n status yang dapat diraih pada setiap *node*, dimana n melambangkan jumlah sisi pada *node*. Status ke-2 dibuat ketika *node* diputar sebesar $^{360}/_{n}$ derajat searah jarum jam (dalam hal ini 4, jadi putaran sebesar 90°). Status ke-3 dibuat dari status ke-2 yang diputar, dst. Dalam kasus ini, ada 35 *node*, maka secara keseluruhan ada 4^{35}, atau 1.1805×10^{21} ruang status dalam pohon (4 status, 35 *node*).



Gambar 3.3. Status 1-4 dari *node* pada tingkat kedalaman 1 (sumber: aplikasi Logical Cell)

Solusi ditemukan ketika setiap *node* terhubung satu sama lain, maka pada penelusuran perlu pengecekan apakah node tersebut terhubung dengan lintasan utama. Apabila node tersebut tidak terhubung, maka pencarian pada simpul tersebut langsung dihentikan, seluruh anak dari simpul tersebut diabaikan, dan sistem akan melakukan runut-balik.

Jika ada beberapa node pada kedalaman yang sama, maka status dinyatakan sebagai *n-tuple*, dengan n sebesar jumlah node pada kedalaman yang sama, dimulai dari paling atas. Sebagai contoh pada Gambar 3.2, ada 3 *node* pada kedalaman 9, maka status yang dibentuk pada kedalaman 9 adalah sebagai berikut:

$$x_9 = \{1..4\,, 1..4\,, 1..4\}$$

Secara keseluruhan, ruang status yang terbentuk adalah sebagai berikut:

$$x_{1,11} = \{1..4\}$$

$$x_{2,10} = \{1..4\,, 1..4\}$$

$$x_{3,9} = \{1..4\,, 1..4\,, 1..4\}$$

$$x_{4,8} = \{1..4\,, 1..4, 1..4\,, 1..4\}$$

$$x_{5,6,7} = \{1..4, 1..4, 1..4\,, 1..4\,, 1..4\}$$

Dan salah satu contoh solusi adalah sebagai berikut:

S = ({3}, {2,1}, {3,2,4}, {1,2,1,3}, {2,2,4,3,1}, {1,2,1,4,4}, {2,1,4,4,2}, {3,2,1,1}, {4,2,4}, {2,1}, {2})

Fungsi pembatas dari algoritma pencarian ini adalah apakah setiap *node* terhubung ke jalur utama. Namun, sebuah kasus khusus boleh dibuat, yakni tidak memprioritaskan keterhubungan terhadap *node* yang hanya memiliki 1 jalur (contohnya pada *node* simpul kedalaman 11). Karena sulitnya menemukan jalur yang hanya terhubung pada simpul tersebut, maka *node* tersebut dapat diabaikan terlebih dahulu. Pencarian kemudian diteruskan berdasarkan apakah *node* dengan 1 jalur tersebut terhubung dengan node pada kedalaman berikutnya.

Selain itu, fungsi pembatas kedua adalah apakah jumlah jalur yang terhubung sesuai jumlahnya. Pada contoh kasus ini tidak terlihat, karena setiap sisi hanya mempunyai 1 jalur keluar. Tetapi, pada level yang lebih sulit, ada maksimal 3 jalur yang keluar dari 1 sisi, sehingga perlu pengecekan jumlah jalur yang terhubung untuk memastikan keterhubungan benar-benar sesuai.

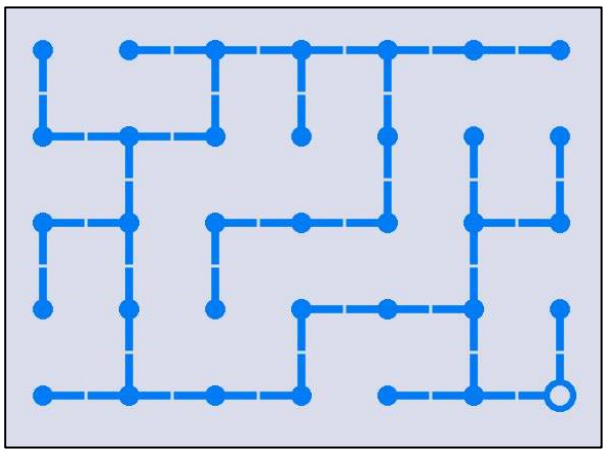
Langkah-langkah untuk penyelesaian dengan menggunakan algoritma ini adalah sebagai berikut:

- Buat pohon ruang solusi
- Tentukan fungsi pembatas dari algoritma
- Penelusuran runut-balik dimulai. Jika *node* terhubung dan jumlah jalur yang terhubung sesuai, maka pencarian dilanjut ke kedalaman berikutnya
- Jika tidak sesuai, maka lakukan pembentukan simpul anak baru atau runut-balik ke simpul hidup terakhir
- Pencarian dilakukan hingga menemukan solusi (pasti ada solusi)

Pada kasus ini, solusi yang didapat adalah sebagai berikut:

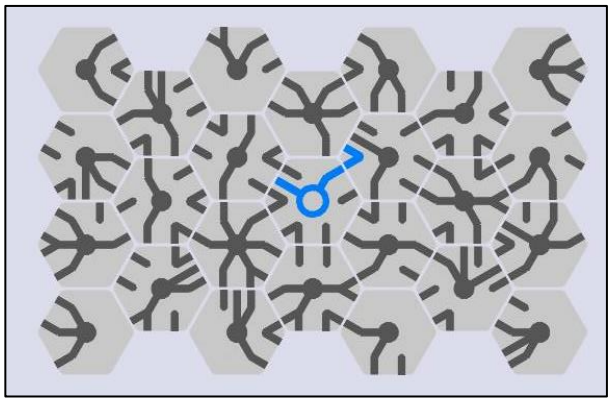
S = ({4}, {2,4}, {2,4,4}, {2,3,1,3}, {3,4,2,4,2}, {1,1,2,2,4}, {2,2,3,1,2}, {4,2,4,4}, {3,3,2}, {2,3}, {2})

Dengan bentuk akhir sebagai berikut:



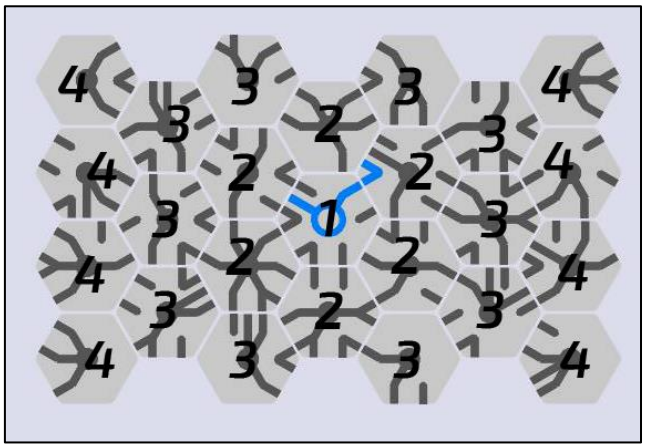
Gambar 3.4. Solusi dari gambar 3.1. (sumber: aplikasi Logical Cell)

Sebuah contoh yang menerapkan fungsi pembatas kedua, adalah sebagai berikut.



Gambar 3.5. Contoh kasus kedua

Jika dilihat, ada 3 jalur yang keluar dari salah satu sisi pada *node* kedalaman simpul 1 (sekali lagi, yang menyala biru). Tetapi, tidak ada *node* yang hanya memiliki 1 jalur keluar, sehingga kasus pertama dapat diabaikan. Serta pada kali ini, ada 6 sisi pada setiap *node*. Maka pohon ruang solusi yang terbentuk adalah sebagai berikut:



Gambar 3.6. Ilustrasi kedalaman pada pohon ruang status (sumber: aplikasi Logical Cell dengan ubahan)

Ada 25 *node* pada *puzzle* kali ini, maka total jumlah ruang status yang terbentuk sebanyak 6^{25}, atau 2.843 x 10^{19} status. Status ke-2 dari suatu *node* dibuat ketika *node* diputar sebesar 360/60 = 60^o, status ke-3 ketika status ke-2 diputar 60^o, dan seterusnya.

Secara keseluruhan, solusi yang terbentuk adalah sebagai berikut:

$$x_1 = \{1..6\}$$

$$x_2 = \{1..6, 1..6, 1..6, 1..6, 1..6, 1..6\}$$

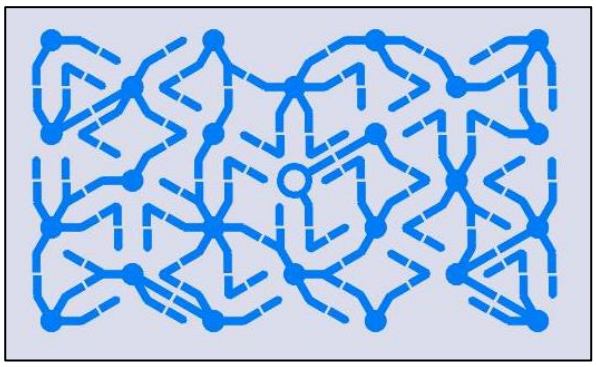
$$x_3 = \{1..6, 1..6, 1..6, 1..6, 1..6, 1..6, 1..6, 1..6, 1..6, 1..6, 1..6, 1..6\}$$

$$x_4 = \{1..6, 1..6, 1..6, 1..6, 1..6, 1..6, 1..6, 1..6\}$$

Dengan catatan, ruang status dalam solusi dimulai dari *node* paling atas diurutkan searah jarum jam. Dan, kali ini jumlah jalur yang keluar dari satu sisi harus diperhitungkan dan disamakan. Maka, solusi yang terbentuk dari persoalan ini adalah:

S = ({3}, {3,6,3,5,4,4}, {6,2,2,2,3,6,2,2,5,4}, {3,3,6,2,3,3,5,2})

Dengan bentuk akhir sebagai berikut:



Gambar 3.7. Solusi dari gambar 3.5.

Pseudocode dari algoritma yang digunakan adalah sebagai berikut:

```
procedure NeurOn (input/output g: GrafNeuron)
 {menyelesaikan puzzle NeurOn Logical Cell.
            masukan: puzzle yang masih teracak
            keluaran: puzzle yang telah selesai
 }
Deklarasi
     p : pohon_status
     s : status
     i : integer
     as : array of status
     g : GrafNeuron
     n : Node

     procedure CreatePohon(input g : GrafNeuron, output p :
pohon_status)
     {membuat pohon ruang status berdasarkan GrafNeuron
dari puzzle NeurOn}

     function IsConnected (input n1, n2 : Node) → boolean
     {mengecek apakah kedua node terhubung satu sama
lain}

     function CountSide (input n : Node) → integer
     {menghitung jumlah sisi pada node}

     function CountPath (input n : Node) → as
     {menghitung jumlah jalur pada setiap sisi}

     procedure CreateSolusi (input s : status, input a : an)
     {membuat solusi berdasarkan status yang dipilih}
Algoritma

  CreatePohon(g, p);
  for (i = 1, i <= CountSide(n), i++)
       if (IsConnected(s[i],s[i+1]) and CountPath(s[i]) =
CountPath (s[i+1])
            CreateSolusi(s[i],an)
       endif
endfor
```

IV. KESIMPULAN

Permainan Logical Cell merupakan sebuah permainan logika dari Jepang, dimana pemain harus menyelesaikan sebanyak mungkin *puzzle* dalam berbagai mode permainan. Salah satu mode yang dibahas pada makalah ini adalah mode NeurOn, dimana pemain harus menghubungkan setiap *node* pada permainan menjadi satu jalur utama.

Salah satu algoritma yang dapat digunakan untuk membantu menyelesaikan *puzzle* tersebut adalah algoritma Runut-balik (*Backtracking*). Algoritma tersebut membentuk pohon ruang status dari seluruh kemungkinan pada persoalan, tetapi hanya menelusuri status yang mendekati solusi, dan mengabaikan status dan seluruh anak dari status yang tidak mendekati solusi.

Dalam algoritma ini, status dari setiap *node* dilambangkan dengan *node* yang diputar, panjang lintasan maksimal adalah jarak *node* terjauh dari *node* simpul kedalaman 1, dan ukuran ruang status sebesar jumlah sisi dipangkat jumlah *node*.

Penelusuran dilakukan dengan mempertimbangkan 2 fungsi pembatas, yakni seluruh *node* pada kedalaman berikutnya bisa terhubung dengan *node* tersebut, dan jumlah jalur yang terhubung sesuai antara 2 *node*. Jika salah satu fungsi pembatas tersebut tidak dipenuhi, maka penelusuran terhadap lintasan dihentikan. Hal tersebut dilakukan hingga solusi ditemukan.

Penulis mengakui bahwa algoritma yang digunakan mungkin belum tentu dapat direalisasikan, karena sangat menghabiskan memori, terutama pada pembuatan pohon ruang solusi. Selain itu, perlu analisis terhadap komponen *node* dalam aplikasi Logical Cell tersebut, atau analisis secara visual dari layar. Karena itu, penulis minta maaf atas kekurangan dalam makalah ini, dan menerima saran dan kritik yang dapat digunakan untuk membangun makalah ini menjadi lebih baik lagi.

V. UCAPAN TERIMA KASIH

Penulis mengucapkan syukur kepada Tuhan Yang Maha Esa, karena makalah ini akhirnya dapat tersusun dengan baik. Penulis berterima kasih kepada Dr. Ir. Rinaldi Munir, yang telah memberikan materi dan bimbingan dalam kuliah Strategi Algoritma.

Penulis juga mengucapkan terima kasih kepada seluruh pihak yang telah memberikan dukungan baik secara fisik maupun moral sehingga penyusunan makalah ini dapat selesai tepat waktu.

DAFTAR PUSTAKA

[1] Munir, Rinaldi. 2009. *Diktat Kuliah IF2211 Strategi Algoritma.* Bandung:Penerbit Informatika

[2] https://twitter.com/LogicalCell_inu, diakses pada 15 Mei 2017 pk. 16.39

[3] https://www.ics.uci.edu/~eppstein/161/960215.html, diakses pada 15 Mei 2017 pk. 20.32

[4] Knuth, Donald E. 1968. *The Art of Computer Programming*. Amerika:Addison-Wesley.

[5] http://algorithms.tutorialhorizon.com/introduction-to-backtracking-programming, diakses pada 16 Mei 2017 pk. 22.37

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, 17 Mei 2017

Adrian Mulyana Nugraha

13515075