Penerapan Pohon dan Algoritma *Heuristic* dalam Menyelesaikan *Sliding Puzzle*

Rezan Achmad (13508104)

Program Studi Informatika Institut Teknologi Bandung
Jalan Ganesha 10 Bandung
email : rezanachmad@yahoo.com; if18104@students.if.itb.ac.id

ABSTRAK

***Sliding puzzle* merupakan salah satu jenis permainan *puzzle* yang cukup terkenal dan banyak disenangi oleh orang karena dalam memainkannya cukup memeras otak. Untuk menyelesaiakan *sliding puzzle* terdapat banyak cara salah satunya dengan menggunakan prinsip pohon divariasikan dengan algoritma *heuristic*. Pada makalah ini akan dijelaskan mengenai proses yang dilakukan untuk menyelesaikan *sliding puzzle*. Basisnya yaitu menggunakan pohon namun perlu penunjang yaitu algoritma heuristic agar lebih cepat untuk menemukan solusi. Semua kemungkian pergeseran puzzle didaftar disimpul pohon. Secara umum proses memperoleh solusi yaitu menentukan nilai heuristic semua daun yang ada, memilih daun yang memiliki nilai heuristic paling kecil untuk kemudian ditentukan anak-anaknya. Anak-anak yang baru terbentuk merupakan daun baru. Selanjutnya tentukan lagi nilai heuristic daun kemudian memilih yang terkecil untuk ditentukan lagi anak-anaknya. Proses ini dilakukan terus menerus hingga didapatkan daun yang memiliki nilai heuristic = 0. Pencarian solusi akan lebih cepat selesai jika memakai prinsip pohon disertai dengan algoritma *heuristic* daripada hanya menggunakan prinsip pohon saja.**

Kata kunci: *Sliding Puzzle*, Pohon, Algoritma *Heuristic*..

1. PENDAHULUAN

Sliding puzzle merupakan salah satu jenis permainan *puzzle* yang cukup memeras otak untuk menyelesaikannya. Pemain ditantang untuk berpikir kreatif bagaimana untuk membuat semua bagian puzzle terletak pada posisi sebenarnya. Cara memainkannya cukup mudah, pemain hanya menggeser *puzzle* satu demi satu sampai akhirnya semua *puzzle* terletak pada posisi sebenarnya. Permainan ini terlihat cukup sederhana namun untuk menempatkan semua *puzzle* pada tempat sebenarnya adalah kendala besar. Pemain harus mengerahkan segala kemampuan otaknya untuk membuat *puzzle* tersebut terletak pada posisi sebenarnya.

Gambar 1.1 *Sliding Puzzle*

Gambar 1.2 Gambar Sebenarnya

Contoh *sliding puzzle* yaitu terdapat pada gambar 1.1 Pemain diminta untuk menyelesaikan *puzzle* tersebut sesuai pada gambar 1.2. Pemain diminta untuk menggeser *puzzle* satu demi satu hingga terbentuk gambar seperti pada gambar 1.2. Memang, permaianan ini terlihat sangat sederhana. Tetapi, untuk menyelesaikannya dibutuhkan logika serta analisis yang kuat.

Sangat banyak kemungkinan pergeseran yang dilakukan untuk menyelesaikan puzzle ini. Hal ini yang membuat puzzle ini “susah” untuk diselesaikan. Berhubung karena *sliding puzzle* pada gambar 1.1 hanya berukuran 3 X 3, cukup mudah untuk menyelesaikan *sliding puzzle* ini. Bagaiamana dengan *sliding puzzle* yang berukuran 4 X 4,

5 X 5 dan seterusnya? Tentu tingkat kesusahannya semakin tinggi karena kombinasi pergeserannya pun semakin banyak.

Pertanyaan yang mungkin muncul yaitu adakah cara untuk memudahkan dalam penyelesaian *sliding puzzle* ini? Jawabannya tentu ada. Bahkan banyak variasi cara yang digunakan untuk menyelesaikan *sliding puzzle* ini. Variasi cara tersebut tergantung kemampuan analisis serta kretifitas pemain. Salah satu cara yang ada yaitu dengan menggunakan bantuan pohon *m-ary*.

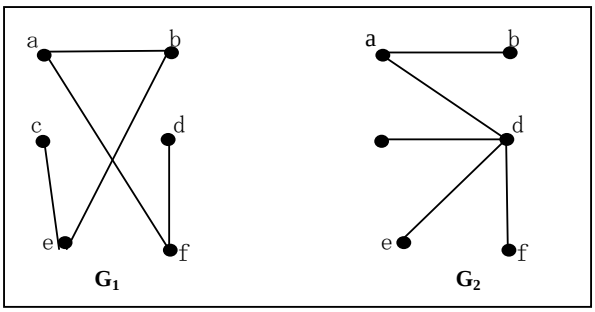
Perlu diketahui bahwa penerapan prinsip pohon memang sangat banyak digunakan dalam memecahkan masalah dalam bidang informatika. Hampir semua bidang dalam informatika memakai topologi pohon untuk menyelesaikan suatu masalah. Dalam dunia permainan pun pohon banyak digunakan untuk menyelesaikan permainan sampai tuntas. Selain dalam hal permaianan puzzle, pohon juga digunakan untuk menyelesaikan permaina sudoku, catur dan lan-lain.

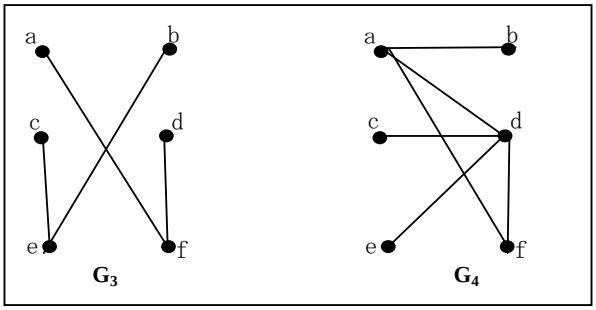
Sesuai kenyatan, pohon memiliki akar, cabang serta daun. Pada penyelesain *sliding puzzle* ini, akan digunakan struktur pohon yang memiliki akar, cabang dan daun. Kondisi awal puzzle diletakkan pada akar. Semua kemungkinan pergesean *puzzle* ditulis ke anak dari akar tersebut (cabang). Dibuat lagi anak untuk anak dari akar sebelumnya. Kemungkinan pergeseran dituliskan lagi disini hingga semua *puzzle* berada pada posisi sebenarnya. Solusi dari puzzle ini disebut daun. Penjelasan lebih rinci terdapat pada bab-bab selanjutnya. Bahkan terdapat cara lain yang dikombinasikan dengan prinsip pohon untuk menyelesaikan sliding puzzle ini salah satunya dengan memvariasikan dengan algoritma *heuristic*.

2. POHON

2.1 Definisi Pohon

Pohon adalah graf tidak berarah yang tidak mengandung sikuit. Ada dua sifat penting pada pohon yaitu terhubung dan tidak mengandung sirkuit. Pada gambar 2.1, hanya G1 dam G2 yang pohon, sedangkan G3 dan G4 bukan pohon. G3 bukan pohon karena ia mengandung sirkuit a,d,f,a sedangkan G4 bukan pohon karena ia tidak terhubung (anda jangan tertipu dengan persilangan dua buah sisi - dalam hal ini sisi (a,f) dan sisi (b,e)- karen titik silangnya bukan menyatakan simpul).





Gambar 2.1 G_1 dan G_2 adalah pohon, sedangkan G_3 dan G_4 bukan pohon.

2.2 Sifat-sifat Pohon

Sifat-sifat pohon dinyatakan dengan teorema 2.1 dibawah ini.

TEOREMA 2.1. Misalkan G = (V, E) adalah graf tidak berarah sederhana dan jumlah simpulnya n. Mkaa, semua pernyataan di bawah ini adalah ekivalen :

1. G adalah pohon.
2. Setiap pasang simpul di dalam G terhubung dengan lintasan tunggal.
3. G terhubung dan memiliki m = n – 1 buah sisi.
4. G tidaj mengandung sirkuit dan memiliki m = n -1 buah sisi.
5. G tidak mengandung sirkuit dan penambahan satu sisi pada graf akan membuat hanya satu sirkuit.
6. G terhubung dan semua sisinya adalah jembatan (jembatan adalah sisi yang bila dihapus menyebabkan graf terpecah du komponen).

2.3 Pohon Berakar dan Terminologinya

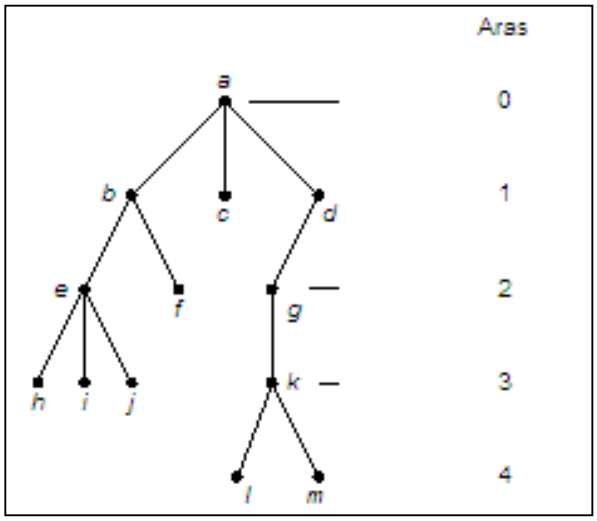
Pada kebanyakan aplikasi pohon, simpul tertentu diperlakukan sebagai akar (root). Sekali sebuah simpul ditetapkan sebagai akar, maka simpul-simpul lainnya dapat dicapai dari akar dengan memberi arah pada sisi pohon yang mengikutinya.

Akar mempunyai derajat-masuk sama dengan nol dan simpul-simpul lainnya berderajat-masuk sama dengan satu. Simpul yang mempunyai derajat-keluar sama dengan nol disebut daun atau simpul terminal. Simpul yang mempunyai derajat-keluar tidak sama dengan nol disebut simpul dalam atau simpul cabang. Setiap simpul di pohon dapat dicapai dari akar dengan sebuah lintasan tunggal (unik).

Anak (*child* atau *children*) dan Orangtua (*parent*)

Misakan Xx adalah sebuah simpul di dalam pohon berakar. Simpul y dikatakan anak dari simpul x jika ada sisi simpul x ke y. Dalam hal demikian, x disebut orangtua(parent). Pada Gambar 2.2 b,c, dan d adalah

anak-anak simpul a, dan a adalah orangtua dari anak-anak itu. E dan f adalah anak-anak simpul b, dan b adalah orangtua dari e dan f. G adalah anak simpul d, dan d adalah orangtua g. Simpul h,i,j,l dan m tidak mempunyai anak.



Gambar 2.2 Pohon Berakar

Lintasan (path)

Lintasan dari simpul v1 ke simpul vk adalah runtunan simpul-simpul v1, v2, .., vk sedemikian sehingga vi adalah orang tua dari vi+1 untuk 1<= i <=k. Dari pohon pada Gambar 2.2, lintasan dari a ke j adalah a, b, e, j, Panjang lintasan adalah jumlah sisi yang dilalui, yaitu k-1. Panjang lintasan dari a ke j adalah 3.

Keturunan (descendant) dan Leluhur (ancestor)

Jika terdapat lintasan dari simpul x ke simpul y di dalam pohon, maka x adalah leluhur dari simpul y, dan y adalah keturunan simpul x. Pada Gambar 2.2, b adalah leluhur simpul h, dan dengan demikian h adalah keuturan b.

Saudara kandung (sibling)

Simpul yang berorangtua sama adalah saudara kandung satu sama lain. Pada gambar 2.2, f adalah saudara kandung e. Tetapi, g bukan saudara kandung e, karena orangtua mereka berbeda.

Derajat (degree)

Ada perbedaaan definisi derajat pada pohon berakar dengan definisi derajat pada graf (termasuk pohon tidak berakar). Derajat sebuah simpul pada pohon berakar adalah jumlah upapohon (atau jumlah anak) pada simpul tersebut. Pada gambar 2.2, derajat a adalah 3, derajat b adalah 2, derajat d adalah satu dan derajat c adalah 0. Jadi, derajat yang dimaksudkan di sini adalah derajat-keluar.

Daun (*leaf*)

Simpul yang berderajat nol atau tidak mempunyai anak disebut daun. Simpul h, i, j, f, c, dan m adalah daun.

Simpul Dalam (internal nodes)

Simpul yang mempunyai anak disebut simpul dalam. Simpul d, e, g, dan k pada Gambar 2.2 adalah simpul dalam.

Aras (*level*) atau Tingkat

Akar mempunyai aras = 0, sedangkan aras simpul lainnya = 1 + panjang lintasan dari akar ke simpul tersebut. Beberapa litelatur memulai nomor aras dari 0, literatur lainnya dari 1. Sebagai konvensi, kita memulai penomoran aras dari 0.

Tinggi (*height*) atau kedalaman (*depth*)

Aras maksimum dari suatu pohon disebut tinggi atau kedalaman pohon tersebut. Atau, dapat juga dikatakan, tinggi pohon adalah panjang maksimum lintasan dari akar ke daun. Pohon pada gambar 2.2 mempunyai tinggi 4.

Pohon berakar yang setiap simpul cabangnya mempunyai paling banyak m buah anak disebut pohon m-ary.

Jika m=2, pohin disebut pohon biner (binary tree). Pohon m-ary dikatakan pohon penuh (*full*) atau pohon teratur jika semua simpul cabangnya mempunyai tepat m buah anak.

Pohon m-ary banyak digunakan diberbagai ilmu maupun dalam kehidupan sehari-hari. Dalam terapannya, pohon *m-ary* digunakan sebagai model yang merepresentasikan suatu struktur. Dua contoh penggunaan pohon m-ary yaitu penurunan kalimat (dalam bidang bahasa) dan direktori arsip di dalam komputer. Contoh penggunaan pohon *m-ary* lainnya adalah struktur organisasi, silsilah keluarga (dalam bidang genetika), struktur bab atau daftar isi didalam buku, bagan pertandingan antara beberapa tim sepakbola dan sebagainya.

Jumlah Daun pada Pohon *m-ary* Penuh

Pohon m-ary penuh adalah pohon yang setiap simpulnya tepat mempunyai m jumlah anak. Pada pohon m-ary penuh dengan tinggi h, jumlah daun adalah m^h. Perhatikan jika T buka pohon m-ary penuh, maka jumlah daun <= m^h.

Jumlah Seluruh Simpul pada Pohon m-ary Penuh

Pada pohon m-ary penuh dengan tinggi h,
aras 0 ⟶ jumlah simpul = $m^0 = 1$
aras 1 ⟶ jumlah simpul = $m^1 = m$
aras 2 ⟶ jumlah simpul = m^2
...
aras h ⟶ jumlah simpul = m^h

maka jumlah seluruh simpul adalah

$$S = m^0 + m^1 + m^2 + ... + m^h = \frac{m^{h+1} - 1}{m - 1}$$

3. ANALISIS PENERAPAN POHON DALAM MENYELESAIKAN *SLIDING PUZZLE*

Sesuai pada bab pendahuluan, solusi yang ditawarkan untuk menyelesaikan *sliding puzzle* ini ialah dengan menggunakan prinsip pohon. Pada bab sebelumnya, telah kita ketahui bahwa ada banyak bentuk pohon yang dapat dibuat. Untuk permasalahan *sliding puzzle* akan digunakan pohon *m-ary* yaitu pohon yang memliki anak paling banyak sejumlah *m*.

Gambar 1.1 merupakan gambar *sliding puzzle*. Kebetulan *puzzle* yang diselesaikan berupa papan tombol suatu alat elektronik jadi cukup mudah mengetahui posisi *puzzle* yang sebenarnya. Bagaimana jika gambar *sliding puzzle* itu berupa pemandangan? Atau berupa foto gedung-gedung tinggi yang memiliki banyak kaca? Tentu akan terasa sulit bagi kita jika objek *sliding puzzle* berupa gambar seperti itu.

Untuk mengatasi hal-hal seperti ini, sebelum objek-objek tertentu dibentuk menjadi *sliding puzzle*, terlebih dahulu tiap bagian *puzzle* diberi nomor. Setelah semua bagian diberi nomor, baru kita mengacak *puzzle* tersebut.

Cara diatas sangat berguna jika kita ingin membuat sebuah *game sliding puzzle*. Terlebih dahulu kita tentu melalukan pemberian nomor kepada tiap bagian objek gambar sebelum gambar tersebut dibentuk menjadi *sliding puzzle* kemudian diacak. Hal ini dilakukan supaya kita dapat mengenali posisi *sliding puzzle* melalui nomor saja. Ini terasa gampang dibandingkan kita mesti mengingat posisi melalui gambar.

Sekarang perhatikan gambar 3.1. Gambar 3.1 merupakan *sliding puzzle* yang telah jadi. Penulis sengaja membahas persoalan *sliding puzzle* dalam bentuk angka dari pada gambar karena seperti yang telah dikatakan sebelumnya bahwa ini hanya untuk mempermudah pembahasan dan pembaca akan lebih mudah untuk mengerti.

1	2	3
4	5	6
7	8	

Gambar 3.1 *Sliding Puzzle* yang telah jadi.

1		3
4	2	5
7	8	6

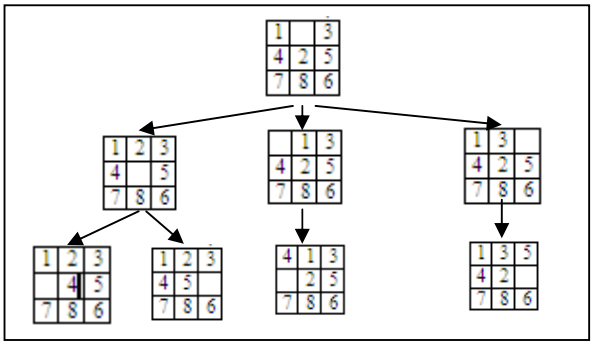
Gambar 3.2 *Sliding Puzzle* yang belum jadi.

Penulis akan melakukan pengacakan pada *Sliding Puzzle* pada Gambar 3.1. Hasil pengacakan dapat dilihat pada gambar 3.2. Sebenarnya, *sliding puzzle* diatas tidak rumit untuk diselesaikan. Anda bisa menebak-nebak langkah untuk membuat *sliding puzzle* pada gambar 3.2 menjadi seperti pada gambar 3.1. Penulis sengaja mengambil contoh yang sederhana untuk melakukan penerapan pohon pada penyelesaian *sliding puzzle* ini agar lebih mudah untuk membahasnya.

Telah disebutkan sebelumnya bahwa penyelesaian *sliding puzzle* ini menggunakan pohon m-ary. Pada *sliding puzzle* ini, akan digunakan pohon 4-ary karena maksimal kemungkinan arah pada *sliding puzzle* hanyalah empat yaitu atas, kanan, bawah dan kiri.

Pertama-tama, Gambar 3.2 dijadikan sebagai akar pada pohon. Untuk permulaan terdapat tiga macam gerak *puzzle* yang bisa dilakukan. Pertama, angka 3 bisa digeser ke sebelah kiri. Kedua, angaka 1 bisa digeser kesebelah kanan. Dan ketiga, angka 2 bisa digeser ke atas. Berarti, Gambar 3.2 akan memiliki tiga anak.

Setelah terebentuk tiga anak atau tiga simpul baru, tiga simpul baru ini masing-masing juga akan memiliki anak sesuai dengan jumlah kemungkinan pergerakan. Hal ini akan terus dilakukan sampai ditemukan sususan *puzzle* seperti pada gambar 3.1.



Gambar 3.3 Pohon yang berisi kemungkinan pergerakan *sliding puzzle*.

Gambar 3.3 merupakan susunan kemungkina pergerakan *sliding puzzle*. Telah dibahas sebelumnya bahwa simpul pada level 0 akan memiliki tiga anak. Kemudian tiga anak ini akan memiliki anak lagi dan seterusnya dilakukan seperti hingga kondisi *puzzle* yang diinginkan ditemukan. Perhatikan tiga simpul pada level satu. Tiga simpul ini merupakan anak dari simpul yang berada pada level nol. Sekarang akan dievaluasi anak yang dihasilkan pada simpul pertama di level satu. Simpul pertama pada level satu pada gambar 3.3 memiliki dua anak. Perhatikan bahwa ada yang aneh pada hal ini. Semestinya simpul pertama pada level satu tersebut memiliki tiga anak. Kemungkinan pergeserannya yaitu angka 4 digeser ke kanan, angka 5 digeser ke kiri, atau angka 2 digeser kebawah. Pada gambar 3.3, tidak terdapat angka 2 yang digeser ke bawah. Jika hal ini lakukan maka akan menyebabkan anaknya akan sama dengan orangtuanya. Hal ini tidak diperbolehkan karena membuat lama waktu evaluasi. Kondisi ini juga berlaku pada simpul-simpul yang lain. Jadi wajar jika anak yang dihasilkan adalah dua, bukan tiga.

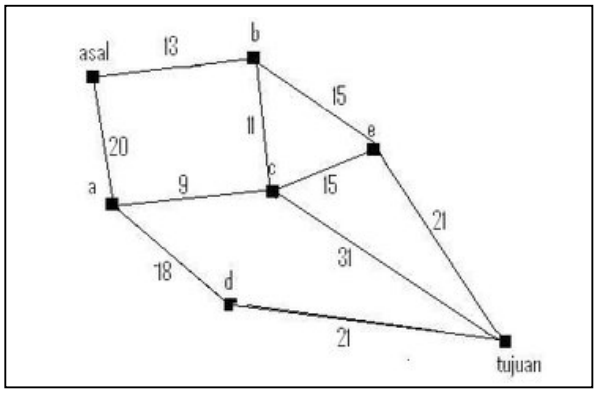
Pohon pada gambar 3.3 tidak menampung semua kemungkinan pergesaran *puzzle* bahkan tidak ada daun pun yang memiliki sosuli. Perlu diperhatikan bahwa solusi dari sliding *puzzle* ini pasti berupa sebuah daun karena pemeriksaan akan berhenti jika kondisi *puzzle* telah jadi. Penulis sengaja tidak melakukan evaluasi pada pohon hingga ditemukan kondisi *puzzle* yang sesuai. Alasannya adalah terlalu banyak kemungkinan pergerakan yang akan dievaluasi sehingga akan membutuhkan waktu yang lama untuk menemukan solusinya.

Telah diketahui sebelumnya bahwa setiap simpul akan dievaluasi atau dicari semua kemungkinan pergeserannya. Nah, hal ini yang membuat akan sangat banyak kemungkinan posisi *puzzle*. Perlu ada cara sehingga tidak perlu untuk mengevaluasi semua simpul. Salah tujuan untuk menemukan cara ini yaitu mempercepat waktu pencarian solusi serta menghemat memori atau banyaknya simpul yang terbentuk.

Salah satu alternatif cara yaitu memvariasikan prinsip pohon dengan algoritma *heuristic*. Sebelumnya akan dijelaskan mengenai algoritma heuristik.

Algoritma *heuristic* adalah algoritma yang setiap langkah penyelesaiannya selalu mencoba menghitung jarak dirinya dengan goal yang akan dicapai, sehingga algoritma tersebut dapat memutuskan langakah selanjutnya yang harus ditempuh.

Contoh sederhana dari algoritma ini adalah seorang agen dalam mencari jalur terpendek dari kota asal ke kota tujuan. Jika diantara kota tujuan dan asal terdapat beberapa kota misalnya : a,b,c,d dan e seperti pada gambar berikut:



Gambar 3.4 Graph dari Kota Asal ke Kota Tujuan.

Tidak ada yang aneh dengan graph yang merepresentasikan peta diatas. Perbedaannya pada heuristic adalah adanya jarak langsung antara tiap kota dengan kota tujuan sebagai berikut :

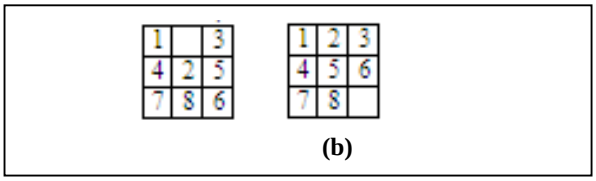
asal-tujuan = 40,
a-tujuan = 35,
b-tujuan = 31,
c-tujuan =25,
d-tujuan =15,
e-tujuan =14.

Lalu sang agen akan mengevaluasi heuristic didepannya yaitu a dan b (lihat gambar 3.4), dipilih b karena (*heuristic* b) + jarak(asal-b) = 44 lebih kecil daripada (*heuristic* a) + jarak(asal-a)=55.

Setelah itu evaluasi lagi *heuristic* didepannya yaitu c dan e, lakukan cara yang sama sehingga kota tujuan tercapai dengan langkah: asal-b-e-tujuan dengan cost total = 49.

Kekurangan dari heuristic ini adalah ketika misalnya e tidak memiliki akses lagi menuju tujuan, maka algoritma ini mengalami kegagalan. Selain itu algoritma *greedy* sendiri tidak selalu menghasilkan hasil optimal.

Dalam penyelesaian *sliding puzzle*, *heuristic* diatas diperbaiki dengan cara menyimpan hasil penjumlahan jarak + *heuristic* ke dalam simpul pohon.



Gambar 3.4 Perbandingan *Sliding Puzzle* yang telah jadi dan belum jadi.

Perhatikan gambar 3.4 diatas. Gambar 3.4 (b) merupakan kondisi awal *puzzle* atau kondisi jadi *puzzle*. Sekarang akan dicari nilai heuristic puzzle (a). Posisi yang salah pada puzzle (a) yaitu angka 2, 5, dan 6. Jarak angka 2 ke posisi sebenarnya adalah 1 kotak. Jarak angka 5 ke posisi sebenarnya adalah 1 kotak. Jarak angka 6 ke posisi sebenarnya adalah 1 kotak. Jadi, nilai *heuristic puzzle* (a) adalah 1+1+1 = 3.

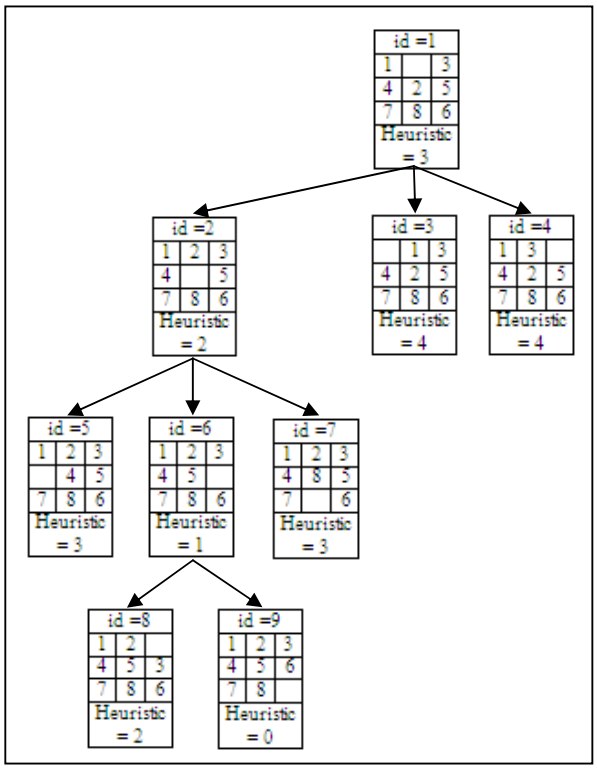
Algoritma untuk menyelesaikan *sliding puzzle* yaitu sebagai berikut :

1. Tentukan *heuristic* semua daun pada pohon.
2. Cari *heuristic* paling kecil pada semua daun.
3. Evaluasi daun yang memiliki *heuristic* paling kecil yaitu menentukan anak-anaknya.
4. Ulangi langkah 1 sampai 3 hingga ditemukan daun yang *heuristic*-nya sama dengan 0.

Sekarang perhatikan gambar 3.5. Awalnya, pohon hanya memiliki satu daun dan sekaligus akar (simpul yang memiliki id = 1). Sesuai dengan algoritma diatas, terlebih dahulu dicari *heuristic* semua daun yang ada pada pohon. Sekarang, hanya terdapat satu daun pada pohon, yaitu id = 1. Nilai *heuristic* id =1 adalah 3, sesuai hasil pada gambar 3.4. Kemudian dicari nilai heuristic yang paling kecil. Karena hanya terdapat satu daun, maka id =1 sekaligus menjadi daun yang terpilih untuk dievaluasi yaitu mencari anak-anaknya.

Anak-anak dari id=1 yaitu id=2, id=3, dan id=4. Sekarang terdapat 3 daun yaitu id=2, id-3, dan id=4. Id=1 bukan daun lagi karena telah memiliki anak. Sekarang cari *heuristic* semua daun. Sesuai dengan cara sebelumnya, diperoleh heuristic id=2 yaitu 2, id=3 yaitu 4 dan id=4 yaitu 4. *Heuristic* paling kecil terdapat pada id = 2.

Sekarang id=2 akan dievaluasi lagi karena merupakan elemen daun yang memiliki nilai heuristic paling kecil. Setelah dievaluasi, id=2 memiliki anak yaitu id = 5, id = 6, dan id = 7. Elemen daun pada pohon sekarang yaitu, id =3, id = 4, id = 5, id =6 dan id = 7. Nilai-nilai dari daun ini dapat dilihat pada gambar 3.5. Nilai *heuristic* paling kecil terdapat pada daun id = 6.



Gambar 3.5 Solusi* Sliding Puzzle *dengan Menggunakan Heuristic.

Hal yang sama tetap dilakukan yaitu melakukan evaluasi pada daun yang memiliki heristic paling kecil yaitu id = 6. Id=6 memilki anak id = 8 dan id = 9. Id=8 memiliki heuristic = 2 dan id = 9 memiliki *heuristic* = 0.

Evaluasi selesai dilakukan karena telah ditemukan daun yang memiliki *heuristic* = 0. Cara penyelesaian *sliding puzzle* ini yaitu dari id=1 ke id=2 ke id=6 dan ke id=9. Jika dalam bentuk pergesaraan yaitu 2 digeser ke atas, 5 digeser ke kiri, dan 6 digeser ke atas.

Sekarang bisa dibandingkan penyelesain *sliding puzzle* dengan hanya menggunakan pohon saja dan menggunakan pohon dengan variasi algoritma lain. Dengan variasi algoritma *heuristic* akan lebih efektif mencari solusi karena tidak semua simpul ditentukan anaknya tetapi hanya simpul yang memiliki *heuristic* paling kecil saja yang ditentukan anaknya.

IV. KESIMPULAN

Kesimpulan pada makalah ini yaitu :

1. Persoalan *sliding puzzle* dapat diselesaikan dengan menggunakan prinsip pohon serta algoritma *heuristic*.
2. Solusi *sliding puzzle* akan lebih cepat diperoleh jika digunakan prinsip pohon dengan variasi algoritma heuristic dibandingkan hanya menggunakan pohon saja.

V. REFERENSI

[1] Munir, Rinaldi, “Matematika Diskrit”, Informatika, 2009.

[2] http://torkasoft.multiply.com/journal/item/10. Akses : 09:40, 19 Desember 2009.