Penyelesaian Permainan Sliding Puzzle 3x3 Menggunakan Algoritma Greedy Dengan Dua Fungsi Heuristik

Akbar Gumbira - 13508106

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10, Bandung 40132, Indonesia
E-mail : if18106@students.if.itb.ac.id

ABSTRAK

Pada makalah ini, penulis membahas tentang penyelesaian Sliding Puzzle dengan ukuran 3x3. Sliding Puzzle merupakan salah satu permainan yang cukup terkenal. Permainan ini ditemukan oleh Sam Loyd pada sekitar tahun 1870. Dalam makalah ini, penulis mencoba untuk menggunakan salah satu algoritma yang cukup terkenal, yaitu algoritma Greedy, dengan menggunakan dua fungsi heuristik sekaligus dalam penyelesaian sliding puzzle ini. Fungsi heuristik ini tentunya memiliki prioritas yang berbeda. Prioritas pertama yang penulis ambil yaitu banyaknya grid puzzle yang berada di tempat yang salah, sedangkan prioritas yang kedua yaitu total keseluruhan jarak dari grid-grid yang tidak berada pada tempatnya yang sesuai, atau sering disebut dengan *manhattan distance*. Pada implementasinya, penulis tidak menggunakan struktur data pohon seperti kebanyakan solusi. Penulis menggunakan matriks untuk struktur data dari puzzle itu sendiri dan array of matriks untuk menyimpan kemunginan state puzzle berikutnya. Tujuan dari pembuatan makalah ini beserta implementasinya adalah penulis ingin menguji apakah dengan penggunaan Algoritma Greedy dengan dua fungsi heuristik sekaligus mampu mencapai solusi dari semua kemungkinan initial state puzzle yang ada.

Kata kunci: *Sliding puzzle*, *greedy*, Sam Loyd, heuristik, *manhattan distance*.

1. PENDAHULUAN

Sliding Puzzle merupakan salah satu jenis permainan puzzle dimana kita harus mencapai goal puzzle dari initial puzzle yang diberikan. Untuk mencapai goal puzzle, sliding puzzle ini menyediakan satu grid kosong agar grid-grid lain disekitarnya dapat digerakkan. Contoh initial state dari suatu puzzle yaitu seperti pada gambar 1.1. Sebagai catatan, angka 0 pada gambar tersebut menandakan pada grid tersebut tidak terdapat blok gambar (blok yang kosong).

Pada makalah beserta implementasinya ini, penulis menggunakan goal state seperti pada gambar 1.2 berikut :

1	8	2
0	4	3
7	6	5

Gambar 1.1 Contoh Initial Puzzle

1	2	3
4	5	6
7	8	0

Gambar 1.2 Goal dari Sliding Puzzle

Pada puzzle berukuran 3x3 ini jika kita mencoba semua kemungkinan pergeserannya, maka akan sangat susah untuk mencapai solusi. Untuk menyederhakan semua state puzzle yang mungkin, kita bisa menggunakan struktur

pohon untuk menelusuri semua kemungkinan dari state puzzle ini dengan algoritma-algoritma lain misalnya dengan DFS, BFS, dll.

Pada pembuatan implementasi ini, penulis ingin memenuhi rasa ingin tahunya untuk penyelesaian puzzle ini dengan suatu algoritma, yaitu algoritma Greedy, dengan menggunakan dua fungsi heuristik sekaligus. Karena penulis menggunakan Algoritma Greedy, maka penulis tidak menggunakan struktur data pohon. Penulis menggunakan matriks sebagai representasi data dari puzzle dan array of matriks untuk menyimpan state puzzle yang mungkin dari suatu puzzle. State puzzle disini maksudnya adalah puzzle-puzzle yang mungkin jika grid kosong dari puzzle digeser ke berbagai arah yang mungkin. Agar state puzzle selanjutnya tidak ada yang sama dari puzzle saat ini pada saat digenerate, untuk mengganti fungsionalitas dari pohon (karena pohon tentunya akan mudah menelusuri tiap state puzzle yang sudah dikunjungi), penulis menggunakan suatu atribut tambahan, yaitu sebuah status darimana puzzle itu diambil.

2. ALGORITMA GREEDY DAN FUNGSI HEURISTIK

2.1 Algoritma Greedy

Algoritma Greedy merupakan algoritma yang sederhana dan lempeng (*straightforward).* Secara harafiah greedy artinya rakus atau tamak.

Algoritma Greedy membentuk solusi langkah per langkah. Terdapat banyak pilihan yang perlu dieksplorasi pada setiap langkah solusi. Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan. Keputusan yang telah diambil pada suatu langkah tidak dapat diubah lagi pada langkah selanjutnya.

Pendekatan yang digunakan pada algoritma Greedy adalah membuat pilihan yang "tampaknya" memberikan perolah terbaik, yaitu dengan membuat pilihan optimum lokal pada setiap langkah dengan harapan bahwa sisanya mengarah ke solusi optimum global.

Berdasarkan pendekatan diatas, kita dapat mendefinisikan algoritma Greedy adalah algoritma yang memecahkan masalah langkah per langkah, pada setiap langkah :

- Mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan (prinsip "*take what you can get now*"
- Berharap bahwa dengan memilih optimum lokal pada setiap langkha akan berakhir dengan optimum global.

Pada setiap langkah di dalam algoritma Greedy kita baru memperoleh optimum lokal. Bila algoritma berakhir, kita berharap optimum lokal menjadi optimum global. Algoritma Greedy mengasumsikan bahwa optimum lokal menjadi optimum global.

Persoalan umum dalam konteks algoritma Greedy disusun oleh elemen-elemen sebagai berikut :

1. **Himpunan Kandidat, C.**
   Himpunan ini berisi elemen-elemen pembentuk solusi. Pada setiap langkah, satu buah kandidat diambil dari himpunannya. Dalam konteks sliding puzzle, himpunan kandidat merupakan himpunan puzzle-puzzle yang mungkin dibentuk dari initial state.
2. **Himpunan Solusi, S.**
   Himpunan ini berisi kandidat-kandidat yang terpilih sebagai solusi persoalan. Dengan kata lain, himpunan solusi merupakan himpunan bagian dari himpunan kandidat. Dalam konteks sliding puzzle, himpunan solusi merupakan state puzzle yang sesuai dengan goal puzzle.
3. **Fungsi Seleksi**
   Fungsi seleksi merupakan fungsi yang pada setiap langkah memilih kandidat yang paling memungkinkan mencapai solusi optimal. Kandidat yang sudah dipilih pada suatu langkah tidak pernah dipertimbangkan lagi pada langkah selanjutnya. Biasanya setiap kandidat, x, diassign sebuah nilai numerik dan fungsi seleksi memilih x yang mempunyai nilai terbesar atau memilih x yang mempunyai nilai terkecil. Pada konteks sliding puzzle, fungsi seleksi ini yaitu dua fungsi heuristik yang penulis gunakan.
4. **Fungsi Kelayakan**
   Fungsi kelayakan adalah fungsi yang memeriksa apakah suatu kandidat yang telah dipilih dapat memberikan solusi yang layak, yakni kandidat tersebut bersama-sama dengan himpunan solusi yang sudah terbentuk tidak melanggar kendala (*constraints*) yang ada. Kandidat yang layak dimasukkan ke dalam himpunan solusi, sedangkan kandidat yang tidak layak dibuang dan tidak pernah dipertimbangkan lagi. Fungsi kelayakan pada sliding puzzle ini yaitu fungsi status yang penulis gunakan. Artinya setiap state puzzle yang dibuat bukan merupakan puzzle yang telah dikunjungi sebelumnya.
5. **Fungsi Obyektif**
   Fungsi obyektif yaitu fungsi yang memaksimumkan atau meminimumkan nilai solusi. Fungsi obyektif pada sliding puzzle ini yaitu kedua fungsi heuristik yang penulis gunakan bernilai 0. Artinya puzzle tersebut telah memenuhi goal puzzle.

Dengan kata lain, persoalan optimasi yang diselesaikan dengan algoritm Greedy melibatkan pencarian sebuah himpunan bagian, S, dari himpunan kandidat, C, yang dalam hal ini, S harus memenuhi beberapa kriteria yang ditentukan, yaitu menyatakan suatu solusi dan S dioptimasi oleh fungsi obyektif.

2.2 Fungsi Heuristik

Misalkan f(n) didefinisikan sebagai suatu fungsi heuristik. Fungsi heuristik f(n) adalah perkiraan biaya termurah dari node n ke node tujuan.

Pada implementasi dari makalah ini, penulis menggunakan dua fungsi heuristik untuk mendapatkan solusi sliding puzzle. Dua fungsi heuristik tersebut yaitu :

1. Banyak grid yang berada pada tempat yang salah, misal untuk kemudahan selanjutnya fungsi heuristik ini kita namai dengan h(n).
2. Total keseluruhan dari jarak tiap grid relatif terhadap tempatnya yang sesuai (*manhattan distance)*. Untuk selanjutnya, fungsi heuristik ini kita namai dengan t(n).

Sebagai contoh, dari gambar 1.1 diatas, kita dapat menghitung kedua fungsi heuristik diatas, yaitu :

1. h(n) = 6, didapat dari grid angka 8, 2, 4, 3, 6, dan 5 yang menempati tempat yang salah.
2. t(n) = 9, didapat dari jarak dari posisi grid angka 8, 2, 4, 3, 6, dan 5 ke tempat yang seharusnya.

Untuk lebih jelasnya, berikut gambar dari grid-grid yang menempati tempat yang salah. Tanda merah menyatakan grid pada tempat yang salah :

1	**8**	**2**
0	**4**	**3**
7	**6**	**5**

Gambar 2.2.1 Grid pada tempat yang salah

3. IMPLEMENTASI SOLUSI PADA PROGRAM

Dalam implementasinya, penulis menggunakan Bahasa Java. Untuk menyimpan puzzle, digunakan matriks berukuran 3x3 dan untuk menyimpan state puzzle berikutnya dari puzzle ini, digunakan array of matriks berukuran 4 array. Ukuran array ini didasarkan pada kemungkinan pergeseran dari puzzle, yaitu Left, Right, Up dan Down. Left artinya grid puzzle yang berada di sebelah kanan grid kosong digeser ke kiri. Right artinya grid puzzle yang berada di sebelah kiri grid kosong digeser ke kanan. Up artinya grid puzzle yang berada di bawah grid kosong digeser ke atas. Down artinya grid puzzle yang berada di atas grid kosong digeser ke bawah.

Berikut langkah-langkah dalam implementasi solusi puzzle ini sesuai dengan algoritma Greedy:

1. Buat State Puzzle yang mungkin dari Initial State

Langkah pertama ini merupakan langkah pembuatan himpunan kandidat dari algoritma Greedy. Dari Puzzle masukan dibuat semua state puzzle yang mungkin berdasarkan grid kosong. Dalam hal ini, semua kemungkinan tersebut disimpan di array of matriks tadi, untuk selanjutnya array of matriks ini dinamai *state*. Penulis menandai *state* ini sebagai berikut :

- *State[0]* : Menyimpan puzzle yang merupakan puzzle sebelumnya yang dilakukan pergeseran Left.
- *State[1]* : Menyimpan puzzle yang merupakan puzzle sebelumnya yang dilakukan pergeseran Right.
- *State[2]* : Menyimpan puzzle yang merupakan puzzle sebelumnya yang dilakukan pergeseran Up.
- *State[3]* : Menyimpan puzzle yang merupakan puzzle sebelumnya yang dilakukan pergeseran Down.

Jika puzzle sebelumnya memang tidak mungkin untuk melakukan suatu pergeseran, maka *State* tersebut diisi dengan nilai NULL. Hal ini merupakan fungsi kelayakan yang penulis buat pada algoritma Greedy. Sebagai contoh pada gambar 3.1, Isi dari tiap *State* yaitu sebagai berikut :

1	8	2
0	4	3
7	6	5

Gambar 3.1 Contoh Initial Puzzle

1	8	2
4	0	3
7	6	5

Gambar 3.2 *State[0]* (Puzzle yang mengalami pergeseran Left)

1	8	2
7	4	3
0	6	5

Gambar 3.3 *State[2]* (Mengalami pergeseran Up)

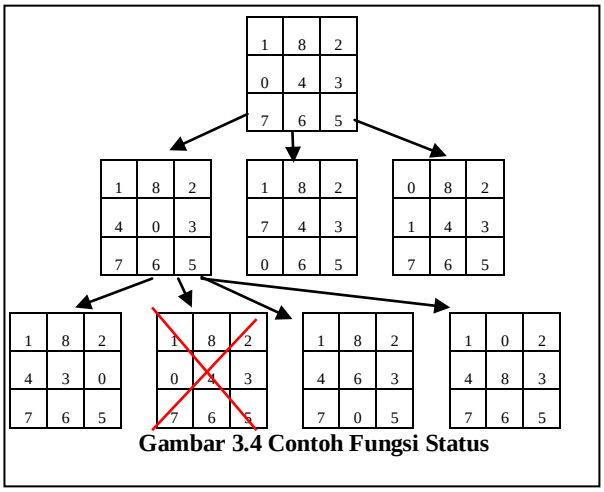
1	8	2
0	4	3
7	6	5

Gambar 3.4 *State[3]* (Mengalami pergeseran Down)

Pada contoh diatas, *State[1]* berisi NULL sebab dari initial puzzle kita tidak dapat melakukan pergeseran Right.

2. Buang State Puzzle yang memiliki status "berlawanan" dengan status puzzle sebelumnya.

Selain pengisian nilai NULL pada state puzzle yang tidak mungkin dibentuk, fungsi kelayakan yang penulis buat juga akan membuang state puzzle yang memiliki status "berlawanan" dengan status puzzle sebelumnya. Pada implementasi ini, dikarenakan penulis tidak menggunakan struktur data pohon, agar *State* yang terpilih nantinya bukan merupakan *State* sebelumnya, maka dibutuhkan suatu status yang merepresentasikan *State* mana yang diambil pada langkah sebelumnya. Agar mudah, penulis memberikan status ini sama dengan indeks dari *State* yang terpilih. Sebagai ilustrasi terdapat pada gambar berikut :



Gambar 3.4 Contoh Fungsi Status

Pada gambar 3.4 diatas, puzzle yang terpilih yaitu Puzzle yang mengalami pergeseran ke kiri (*State[0]*). Oleh karena itu, puzzle diberi status 0 (sama dengan indeks statusnya). Karena puzzle memiliki status 0 (Left), maka kita tidak boleh mengikutsertakan *State* Puzzle yang mengalami pergeseran ke kanan (status 1). Dari sini State Puzzle yang mengalami pergeseran Right kita isi dengan NULL (pada gambar disilang dengan warna merah) karena *State* ini sama dengan Puzzle sebelumnya.

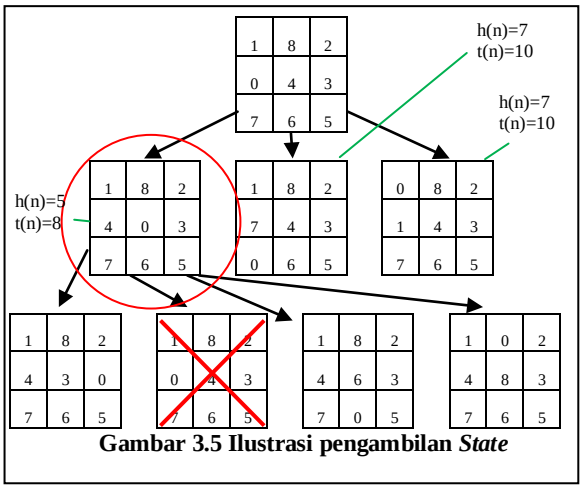
Penghapusan *State* Puzzle ini berlawanan dengan status sebelumnya. Misal jika status puzzle saat ini yaitu 0, maka *State[1]* harus kita hapus (diisi NULL). Jika status puzzle saat ini yaitu 1, maka *State[0]* harus kita hapus (diisi NULL). Hal ini berlaku juga dengan status 2 dan 3.

3. Aplikasikan Fungsi Heuristik pada State Puzzle Tersisa

Langkah ini merupakan implementasi dari fungsi seleksi algiritma Greedy. Dari semua kemungkinan puzzle yang ada dengan menggunakan fungsi heuristik yang didefinisikan, kita pilih *State* mana yang memiliki biaya paling sedikit. Pada program ini, penulis menggunakan dua fungsi heuristik. Fungsi heuristik yang pertama yaitu banyak grid yang menempati tempat yang salah, selanjutnya akan dinamai dengan h(n). Sedangkan fungsi heuristik yang kedua yaitu total keseluruhan jarak tiap grid yang menempati tempat yang salah terhadap posisi grid yang benar, atau sering disebut dengan *manhattan distance*. Fungsi heuristik yang kedua ini selanjutnya akan dinamai dengan t(n). Fungsi heuristik ini memiliki prioritas yang berbeda, dengan

h(n) memiliki prioritas yang lebih tinggi dibandingkan dengan t(n). Artinya, pertama kali akan ditelusuri *State* mana yang memiliki h(n) yang lebih kecil. Jika terdapat *State* yang memiliki h(n) sama, maka akan ditelusuri t(n) yang lebih kecil. Jika ternyata t(n) nya pun sama, maka penulis menetapkan untuk mengambil *State* yang terakhir ditelusuri dan membuang *State* lain yang tidak diambil.

Berikut ilustrasi pengambilan *State* ini :



Gambar 3.5 Ilustrasi pengambilan *State*

Dari cara pengambilan puzzle ini, kita dapat memberikan dugaan terkait dengan algoritma Greedy ini. Dugaan ini yaitu bahwa dimungkinkan solusi tidak dapat dicapai karena solusi mungkin muncul dari *State* yang bukan diambil. Penulis mencoba untuk meminimalisir State yang memiliki fungsi heuristik yang sama ini dengan menggunakan dua fungsi heuristik sekaligus. Namun kenyataanya masih tetap dimungkinkan beberapa *State* dengan h(n) minimal dan t(n) minimal yang sama, yang mengakibatkan tidak ditemukannya solusi atau solusi yang dihasilkan tidak optimal.

4. **Perbaharui Status Pada Puzzle yang terpilih sesuai indeksnya**

   Setelah puzzle terpilih berdasarkan fungsi heuristik yang memberikan cost minimal, status pengambilan *State* ini harus kita perbaharui sesuai dengan indeks *State* yang diambil.

5. **Ulangi langkah 1-4 sampai banyak grid yang salah tempat berjumlah 0.**

   Untuk mendapatkan solusi puzzle, maka langkah-langkah 1-4 sebelumnya harus diulangi sampai ditemukan puzzle dengan h(n) bernilai 0. Karena dimungkinan tidak dicapai sebuah solusi yang memenuhi goal puzzle, maka penulis membatasi perulangan sampai dengan 5000 kali.

4. HASIL PENGUJIAN

Dari hasil pengujian pada program yang penulis buat terdapat initial puzzle yang tidak memiliki solusi. Contoh puzzle ini yaitu :

1	2	3
4	5	6
0	8	7

Gambar 4.1 Contoh Puzzle yang tidak dapat diselesaikan

Pada program yang dibuat, penulis tidak memberikan suatu exception pada puzzle ini ataupun *State* yang memberikan puzzle ini. Namun, puzzle yang tidak terselaikan ini nantinya akan diberhentikan pencarian solusinya setelah mencapai 5000 langkah.

Contoh beberapa puzzle yang sudah penulis coba dan berhasil diselesaikan dengan pendekatan solusi yang penulis buat yaitu :

1	2	3
4	5	8
6	7	0

1	8	2
0	4	3
7	6	5

1	2	3
4	0	8
7	6	5

1	2	3
4	6	0
7	5	8

1	0	3
4	2	5
7	8	6

1	2	3
4	6	8
7	0	5

Gambar 4.2 Contoh Puzzle yang dapat diselesaikan

5. KESIMPULAN

Dari semua yang telah dipaparkan sebelumnya, ada beberapa kesimpulan yang dapat ditarik dari makalah ini, yaitu :

1. Algoritma Greedy ternyata belum tentu mampu memberikan solusi dari semua kemungkinan initial puzzle yang ada. Jika ditemukan solusi, maka solusi ini belum tentu merupakan solusi yang optimal. Hal ini membuktikan sifat dari Algoritma Greedy yaitu bahwa solusi yang didapat dimungkinkan tidak optimal.
2. Fungsi heuristik mampu menyederhanakan kemungkinan puzzle yang mungkin berdasarkan cost minimal. Penulis menggunakan dua fungsi heuristik sekaligus dengan tujuan meminimalisir banyak *State* puzzle yang memiliki fungsi heuristik yang sama. Namun, pada kenyataanya masih dimungkinan *State* yang memiliki dua fungsi heuristik ini dengan nilai yang sama. Jika terdapat kemungkinan seperti ini, *State* yang akhirnya penulis ambil memungkinkan tidak memiliki solusi, karena pencapaian solusi malah mungkin berasal dari *State* yang tidak diambil.

REFERENSI

[1] Munir, Rinaldi. *Diktat Kuliah IF3051 – Strategi Algoritma*. Bandung : Program Studi Teknik Informatika, Institut Teknologi Bandung. 2009.

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, 6 Desember 2010

Ttd

Akbar Gumbira
13508106