Pembentukan pohon pencarian solusi dan perbandingan masing-masing algoritma pembentuknya dalam simulasi N-Puzzle

Windarto Harimurti – NIM : 13503089

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if13089@students.if.itb.ac.id

Abstrak

Makalah ini membahas tentang studi terhadap pohon dan graf serta implementasinya dalam pohon pencarian solusi pada simulasi N-puzzle. simulasi n-puzzle merupakan sebuah simulasi dimana diberikan sebuah state awal dan program akan menggerakkan salah satu blok puzzle yang kosong untuk menggantikan salah satu blok puzzle yang ada di kiri, kanan, atas atau bawah dari blok puzzle yang kosong tersebut. Pergerakan puzzle kosong itu akan terus dilakukan sampai ditemukan sebuah solusi dari simulasi tersebut.

Ketika akan melakukan suatu pencarian solusi terhadap suatu permasalahan tertentu dalam dunia pemrograman diperlukan suatu struktur khusus yang mampu menampung berbagai langkah-langkah berikutnya yang akan mungkin akan terjadi ketika diberikan sebuah state awal atau state perantara ketika suatu persoalan itu belum mencapai finish/state akhir. Berdasarkan langkah-langkah yang mungkin akan terjadi itu kemudian ditentukan satu langkah yang akan dijadikan langkah berikutnya dari pencarian solusi itu. Struktur tersebut dinamakan pohon pencarian solusi, dimana selama mencari solusi yang mungkin, sebuah state akan melahirkan berbagai state anak yang mungkin terjadi , dan pohon pencarian solusi akan menentukan state anak yang akan dipilih sebagai state berikutnya. Sebuah solusi dari pohon pencarian solusi akan membentuk sebuah lintasan graf berarah dimana titik awal merupakan state awal dan titik akhir merupakan state akhir. Pohon pencarian solusi ini digunakan juga oleh berbagai aplikasi simulasi N-Puzzle, pada N-Puzzle terdapat sebuah state awal yang akan digunakan sebagai akar pohon dan membentuk berbagai kemungkinan solusi yang ada sebagai node anak, kemudian akan memasukkan satu node anak ke himpunan solusi, langkah tersebut dilakukan berulang-ulang sampai ditemukan state final yang ditentukan oleh aplikasi.

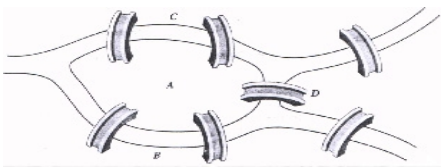
Pembentukan pohon solusi sebetulnya memiliki banyak algoritma untuk pemecahan persoalannya, namun pada makalah ini dibatasi pada 2 algoritma saja yaitu Algoritma Breadth First Search(BFS) dan Depth First Search(DFS), dimana 2 algoritma itu memiliki karakteristik masing-masing dalam pembentukan pohon pencarian solusi.

Keyword: *pohon, graf, pohon pencarian solusi, N-Puzzle, Breadth First Search, Depth First Search*

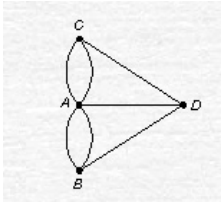
1. Pendahuluan Graf dan Pohon

Pada pembahasan tentang pohon pencarian solusi sebaiknya harus memahami dulu konsep pohon dan konsep graf. Graf digunakan untuk merepresentasikan objek-objek dan hubungan-hubungan diantara objek tersebut. Definisi teknis dari Graf adalah kumpulan himpunan simpul yang tidak boleh kosong dan kumpulan himpunan sisi. Simpul direpresentasikan sebagai objek-objek sedangkan sisi direpresentasikan sebagai hubungan antar objek tersebut.

Awal mula dari konsep graf sendiri adalah permasalahan jembatan koenigsberg, seperti yang diperlihatkan pada gambar di bawah.



Permasalahan yang diajukan adalah mampukah seseorang melintasi seluruh jembatan yang berada pada gambar dan kembali ke tempat asal. Permasalahan ini dipecahkan oleh seorang matematikawan euler yang merepresentasikan jembatan koenigsberg menjadi sebuah graf, dimana simpul merupakan representasi daratan dan sisi merupakan representasi dari jembatan, seperti di bawah

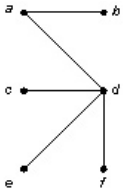


dalam solusinya euler mengatakan bahwa tidak mungkin bisa melalui seluruh jembatan dan kembali ke tempat semula jika derajat setiap simpul tidaklah genap. Derajat merupakan jumlah sisi yang terhubung dengan suatu simpul.

Dalam setiap graf yang tiap simpul memiliki satu atau lebih sisi, selalu terdapat lintasan diantara simpul-simpul tersebut. Lintasan didefinisikan sebagai sebuah kumpulan simpul dimana terdapat satu simpul bertindak sebagai simpul awal dan satu simpul sebagai simpul akhir dan diantara simpul awal dan akhir itu terdapat himpunan sisi dan simpul yang menghubungkan simpul awal dan simpul akhir.

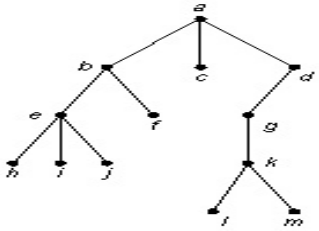
Sebagai contoh pada graf jembatan koenigsberg di atas B – D – A – C merupakan sebuah lintasan. Lintasan memiliki bentuk khusus yang dinamakan sirkuit, yaitu sebuah lintasan yang memiliki simpul awal dan simpul akhir sama, sebagai contoh dari gambar di atas yaitu B – A – C – D – B.

Sebuah pohon merupakan bentuk khusus dari suatu graf, dimana pohon adalah sebuah graf yang tidak memiliki sirkuit namun semua simpul saling terhubung satu sama lain, terhubung di sini artinya adalah selalu terdapat lintasan yang menghubungkan setiap simpul pada pohon. Berdasarkan definisi pohon maka bentuk di bawah ini termasuk pohon.



Pada pohon setiap simpul umumnya hanya akan disebut dengan nama node, untuk membedakan istilah dengan istilah yang digunakan pada graf.

Pohon seperti digambarkan di atas akan sulit digunakan pada pohon pencarian solusi, karena pohon pencarian solusi membutuhkan satu buah node yang akan digunakan sebagai node awal. Oleh karena itu pohon pencarian solusi menggunakan bentuk khusus dari pohon yaitu pohon berakar, dimana definisi pohon berakar sama dengan definisi pohon namun ditambahkan dengan satu node pada pohon dianggap sebagai akar dan semua sisi dibuat berarah. Pohon berakar dapat dilihat pada gambar di bawah.

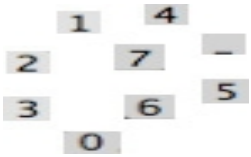


Pada pohon berakar terdapat beberapa terminologi yang sering digunakan yaitu parent dan child, yang dipandang dari node yang digunakan, parent adalah simpul yg berada di atas node yang digunakan, sedangkan child merupakan simpul yang terletak di bawah node yang digunakan. Sebagai contoh pada gambar jika menggunakan simpul b, simpul itu memiliki parent simpul a, dan memiliki 2 child taitu simpul b dan simpul f. simpul yang tidak memiliki parent dinamakan akar sedangkan simpul yang tidak memiliki child dinamakan daun. Pada gambar di atas, akar adalah simpul a, sedangkan simpul h,i,j,fc,l,m adalah simpul daun.

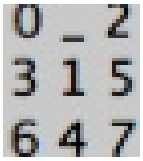
2. Pengenalan simulasi N-Puzzle

Simulasi N-puzzle merupakan sebuah simulasi yang dibuat berdasarkan permainan N-puzzle. Permainan n-puzzle dapat dijelaskan sebagai berikut :

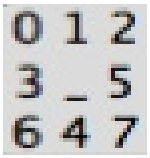
diberikan balok dengan jumlah n, dimana pada tiap-tiap balok dinomori penomoran dari nomor 0 sampai n-2 dan satu buah balok yang diberi tanda “blank” atau kosong. Pada contoh di bawah menggunakan 9 buah balok yang diawali dari 0 sampai 7 dan satu buah balok kosong.



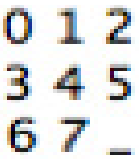
Kemudian masing-masing balok akan diletakkan membentuk persegi dimana urutannya bebas. seperti yang diperlihatkan pada gambar di bawah.



Pada permainan n-puzzle ini balok yang berisi blank ini harus dipindah-pindahkan. Pemindahan bisa dilakukan dengan memindahkan ke atas, bawah, kiri atau kanan. Seperti pada contoh di atas blok “blank” dapat dipindahkan ke kanan, kiri atau bawah. Jika dipindahkan ke bawah maka akan menghasilkan gambar seperti :



blok “blank” harus terus dipindah-pindahkan agar solusi akhir didapat, solusi akhir dari permainan ini biasanya adalah balok-balok yang ada pada permainan terurut mulai dari nomor blok paling kecil di ujung kiri atas sampai blok blank di ujung kanan bawah seperti yang diperlihatkan pada gambar di bawah ini.



Simulasi n-puzzle ini dilakukan dirancang berdasarkan aturan-aturan yang sudah disebut di atas seperti hanya balok kosong yang dapat dipindah-pindahkan, susunan yang bebas. Perbedaannya hanya pada simulasi ini proses pergerakkan balok “blank” dilakukan oleh komputer, serta state akhir yang bisa ditentukan sendiri oleh user yang menggunakan simulasi ini

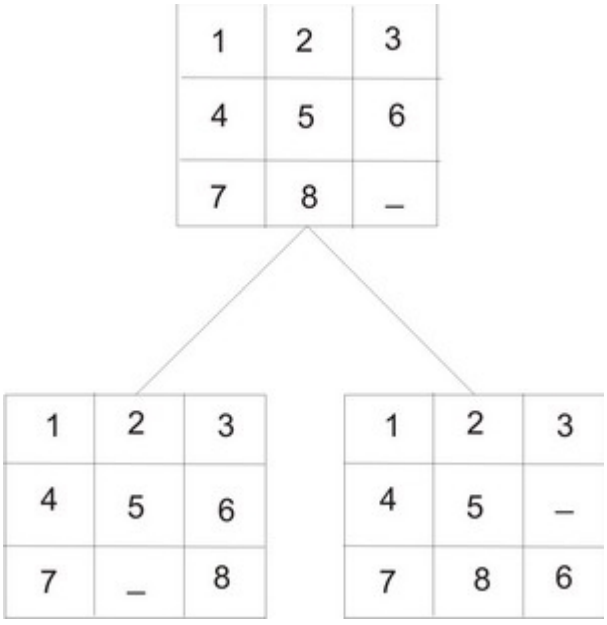
Proses pada simulasi ini dimulai dengan memasukkan berapa angka n yang akan digunakan oleh user, kemudian user juga menentukan state akhir dari pergerakan balok. Setelah itu program akan memberikan posisi blok angka-angka secara acak kemudian program akan menampilkan animasi pergerakkan balok "kosong" sampai posisi balok sesuai dengan state akhir yang diberikan oleh user.

3. Struktur Pohon pada pencarian solusi

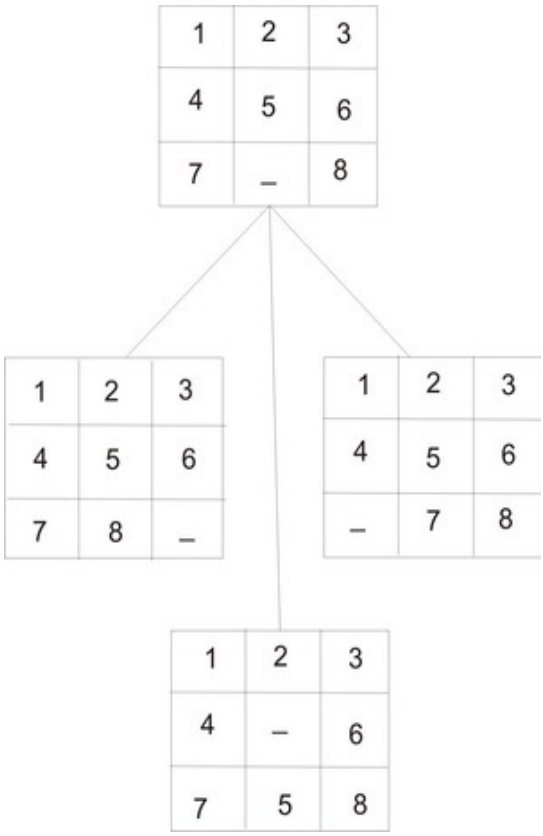
Pada pohon pencarian solusi struktur pohon yang dibentuk memiliki kesamaan dengan pohon berakar yang sudah dibahas pada bab pendahuluan di atas. Yang berbeda pada pohon pencarian solusi ini adalah representasi nodenya. Node dari pohon pencarian solusi pada permainan n-puzzle ini adalah state dari permainan n-puzzle itu sendiri, sebagai contoh adalah jika diberikan state awal seperti ini

1	2	3
4	5	6
7	8	_

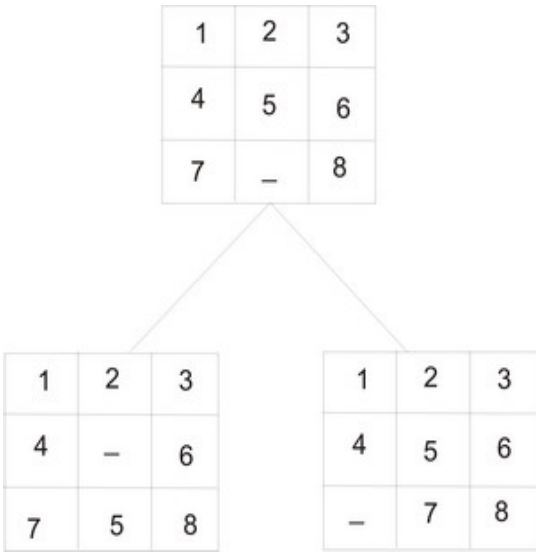
dapat dilihat pada state di atas blok blank dapat ditukar dengan blok no 6 dan blok no 8, sehingga dapat dibentuk anak-anak pohon dari state awal sebagai berikut :



struktur pohon di atas memiliki kelemahan yaitu pergerakan blok "blank" dapat saja kembali ke state awal seperti yang terjadi pada jika kita membuat anak-anak pohon dari state yang di sebelah kiri.



Dapat dilihat pada anak pohon paling kiri memiliki kesamaan state pohon pembentuk parentnya. Jika kejadian ini terjadi maka akan membentuk pohon solusi yang sama dalam satu pohon solusi atau rekursif. Oleh karena itu dalam membentuk suatu anak pohon, sebaiknya anak pohon harus diperiksa dulu apakah anak pohon itu memiliki kesamaan state dengan salah satu node yang ada pada pohon. Jika terdapat kesamaan, maka anak pohon itu tidak boleh dimasukkan dalam pohon pencarian solusi, jika tidak maka dapat ditambahkan pada pohon pencarian solusi. Jika aturan tersebut diimplementasikan pada pembentukan pohon solusi, maka pembentukan pohon di atas, hanya akan menghasilkan 2 anak pohon seperti yang terlihat di bawah ini :



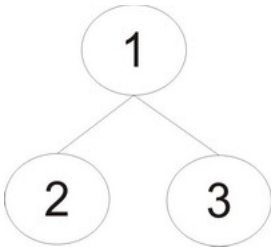
4. Jenis Algoritma dalam pembentukan pohon solusi

Dalam abstraksi sudah dikatakan bahwa algoritma pembentukkan pohon solusi banyak sekali, namun yang dibahas disini adalah pembentukkan pohon solusi dengan algoritma Breadth First Search(BFS) dan Depth First Search(DFS)

- Breadth First Search

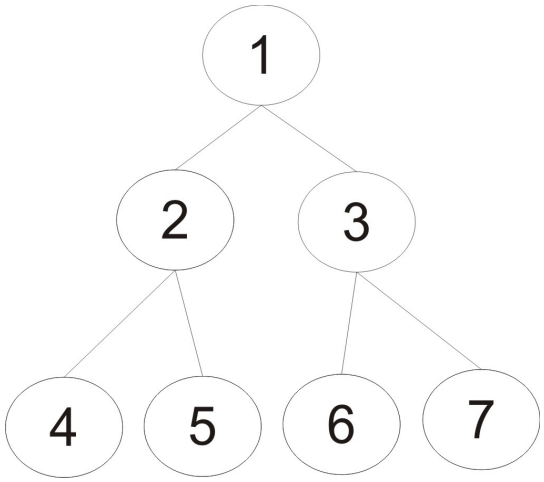
pada algoritma ini diberikan sebuah state awal, kemudian pohon solusi yang dibentuk adalah membentuk semua state pohon anak terlebih dahulu, kemudian dari setiap state pohon anak dibentuk state pohon anak kembali. Penggambaran masing-masing tahap pada pembentukan pohon solusi dapat dilihat di bawah ini.
Tahap 1 : diberikan state awal

Tahap 2 : pembentukan state pohon anak



pembentukan state pohon anak juga dimulai dari pohon anak kiri kemudian baru anak pohon kanan, sesuai dengan penomoran pada penggambaran di atas.

Tahap 3 : pembentukan state pohon anak dari anak pohon state awal



urutan angka merupakan urutan pembentukan anak pohon.

Langkah-langkah yang dilakukan dalam membuat pohon solusi dengan algoritma BFS adalah:

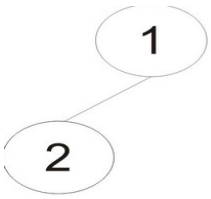
1. masukkan simpul akar ke dalam antrian Q. jika simpul akar adalah simpul solusi, maka solusi telah ditemukan, maka simulasi berhenti.
2. Jika antrian Q kosong maka tidak ada solusi, maka solusi berhenti
3. ambil simpul dari kepala(*head*) antrian, bangkitkan semua anak-anaknya, jika tidak memiliki anak maka akan kembali ke langkah 2. Tempatkan semua anak dari kepala(*head*) antrian di belakang antrian Q, sesuai dengan urutan kelahirannya.
4. Jika suatu simpul anak v adalah simpul solusi, maka solusi telah ditemukan, kalau tidak kembali ke langkah 2.

- Depth First Search

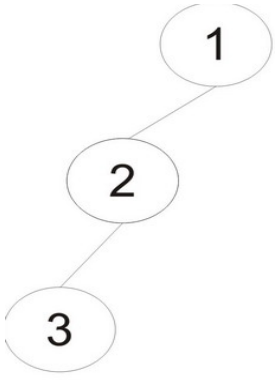
Pada algoritma ini diberikan sebuah state awal, kemudian pohon solusi yang dibentuk adalah membentuk satu anak pohon terlebih dahulu kemudian dari satu anak pohon itu akan dibentuk satu anak pohon lagi. Pembentukan pohon anak secara mendalam, terus dilakukan sampai batas tertentu, dimana batas tertentu tersebut ditentukan oleh program solusi. Penggambaran masing-masing tahap dapat dilihat di bawah ini.

Tahap 1 : diberikan state awal

Tahap 2 : pembentukan satu state pohon anak



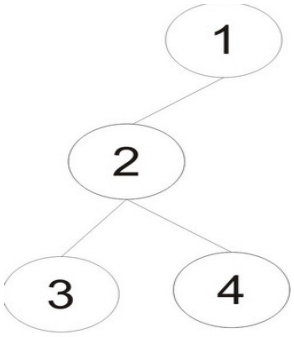
Tahap 3 : pembentukan state pohon anak dari anak pohon state awal



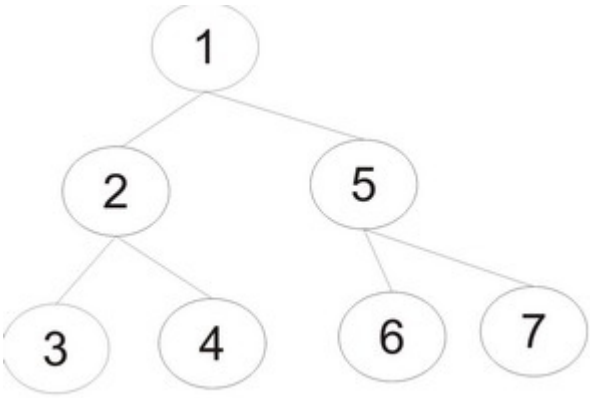
urutan angka merupakan urutan pembentukan anak

pohon, pembentukan anak pohon lainnya ditentukan oleh batas kedalaman dari pembentukan algoritma DFS. Ketika sudah mencapai batas kedalaman tersebut, maka pohon anak akan melakukan *backtrack* ke pohon *parent.* Pohon parent tersebut akan membentuk pohon anak yang lain, jika ada, jika tidak ada maka, akan melakukan *backtrack* lagi ke pohon *parent*nya. Pola tersebut berulang sampai tidak ada lagi anak pohon yang tidak dapat dibentuk.

Sebagai pada pohon di atas jika batas kedalaman 2 maka setelah terbentuk pohon 3 terbentuk dan ditambahkan sebagai anak pohon 2, maka simulasi akan melakukan *backtrack* ke pohon 2 dan akan membentuk pohon 4 sebagai anak pohon 2.



pola tersebut akan kembali berulang sampai seluruh simpul daun pada pohon terbentuk dan tidak ada lagi simpul dalam yang dapat dibentuk, sehingga pohon contoh di atas akan memiliki bentuk seperti gambar di bawah



urutan angka merupakan urutan kelahiran simpul pohon tersebut.

Langkah-langkah yang harus dilakukan dalam pembentukan pohon solusi dengan algoritma DFS adalah :

1. masukkan simpul akar ke dalam antrian Q, jika simpul akar merupakan simpul tujuan maka solusi telah ditemukan, simulasi berhenti
2. jika antrian Q kosong maka tidak ada solusi.
3. Ambil simpul dari kepala antrian Q jika kedalaman simpul yang diambil sama dengan batas kedalaman maksimum kembali ke langkah 2.
4. bangkitkan satu anak simpul yang diambil, jika simpul yang diambil tidak mempunyai anak lagi, kembali ke langkah 2. tempatkan semua anak pada awal antrian Q
5. jika anak dari simpul yang diambil merupakan simpul tujuan, berarti solusi telah ditemukan, kalau tidak sama maka kembali ke langkah 2

struktur data yang dipakai dalam langkah-langkah pembentukan pohon solusi baik dengan algoritma BFS dan DFS memang sebuah antrian, karena dalam pembentukan anak pohon sangat dibutuhkan akses terhadap *parent,* dimana jika menggunakan struktur data rekursif pohon, program tidak bisa mengakses parent dari node tersebut.

5. Implementasi algoritma pencarian solusi pada permainan N-Puzzle

Dalam permainan N-puzzle pergerakan sebuah blok blank harus diatur terlebih dahulu prioritasnya, hal ini harus dilakukan karena pembentukan state pohon anak tergantung dari pergerakan blok ini. Jika prioritas tidak ditentukan maka pembentukan state pohon anak sulit ditentukan, terutama dalam algoritma DFS, karena algoritma DFS membentuk satu anak pohon hingga kedalaman tertentu, oleh karena itu harus ada pergerakan blok yang diutamakan. Pada umumnya pergerakan blok blank ini memiliki prioritas atas,bawah,kiri dan kanan. Prioritas disini berarti jika blok blank mampu bergerak ke atas, maka blok blank akan selalu bergerak ke atas, jika tidak bisa maka baru bisa bergerak ke bawah, demikian juga dengan pergerakan ke kiri atau kanan.

Pergerakan blok blank ini dibatasi juga oleh posisi dari blok blank itu sendiri Jika blok blank tersebut berada pada tembok sebelah kiri seperti yang terlihat pada gambar di bawah ini.

1	2	3
4	8	6
7	5	–

Maka pergerakkan blok blank di atas hanya bisa dilakukan dengan blok di atasnya atau blok di sebelah kirinya. Demikian juga jika blok blok blank memiliki batas di sebelah kanan atau di atasnya. Sebuah blok blank dapat bergerak bebas jika di sebelah kiri, kanan, atas, bawahnya tidak dibatasi oleh tembok pembatas. Seperti yang ditampilkan pada gambar di bawah ini

1	2	3
4	–	6
7	5	8

- Algoritma BFS

Breadth First Search (BFS) pada simulasi ini menggunakan 2 struktur data. Struktur data yang pertama adalah struktur data Matriks dan yang kedua adalah struktur data Queue. Struktur data matriks digunakan untuk merepresentasikan Tampilan N-Puzzle di layar, sedangkan struktur data Queue digunakan untuk menampung state matriks yang kelak akan berubah-ubah.

Algoritma BFS membuat semua kemungkinan pohon anak dan memasukkannya ke ke antrian untuk diproses, oleh karena itu pada algoritma BFS di simulasi ini terdapat 16 buah kondisi melahirkan state pohon anak dari sebuah state yang diberikan yang dilakukan berdasarkan posisi blok blank. 16 buah kemungkinan pembuatan state pohon anak itu adalah :

1. atas, bawah, kiri, kanan

pada posisi ini sebuah state yang diberikan dapat melahirkan 4 buah state anak. Dimana pada masing-masing state anak memiliki posisi blok blank diatas, dibawah dikiri dan dikanan dari posisi blok blank yang dimiliki oleh state yang diberikan

2. atas, bawah, kiri

pada posisi ini sebuah state yang diberikan dapat melahirkan 3 buah state anak. Dimana pada masing-masing state anak memiliki posisi blok blank diatas, dikiri dan dikanan dari posisi blok blank yang dimiliki oleh state yang diberikan

3. atas, bawah, kanan

pada posisi ini sebuah state yang diberikan dapat melahirkan 3 buah state anak. Dimana pada masing-masing state anak memiliki posisi blok blank diatas, dibawah dan dikanan dari posisi blok blank yang dimiliki oleh state yang diberikan

4. atas, kiri, kanan

pada posisi ini sebuah state yang diberikan dapat melahirkan 3 buah state anak. Dimana pada masing-masing state anak memiliki posisi blok blank diatas dikiri dan dikanan dari posisi blok blank yang dimiliki oleh state yang diberikan

5. atas, bawah

pada posisi ini sebuah state yang diberikan dapat melahirkan 2 buah state anak. Dimana pada masing-masing state anak memiliki posisi blok blank diatas dan dibawah dari posisi blok blank yang dimiliki oleh state yang diberikan

6. atas, kiri

pada posisi ini sebuah state yang diberikan dapat melahirkan 2 buah state anak. Dimana pada masing-masing state anak memiliki posisi blok blank diatas dan dikiri dari posisi blok blank yang dimiliki oleh state yang diberikan

7. atas, kanan

pada posisi ini sebuah state yang diberikan dapat melahirkan 2 buah state anak. Dimana pada masing-masing state anak memiliki posisi blok blank diatas dan dikanan dari posisi blok blank yang dimiliki oleh state yang diberikan

8. bawah, kiri, kanan

pada posisi ini sebuah state yang diberikan dapat melahirkan 3 buah state anak. Dimana pada masing-masing state anak memiliki posisi blok blank dibawah dikiri dan dikanan dari posisi blok blank yang dimiliki oleh state yang diberikan

9. bawah, kanan

pada posisi ini sebuah state yang diberikan dapat melahirkan 4 buah state anak. Dimana pada masing-masing state anak memiliki posisi blok blank dibawah dan dikanan dari posisi blok blank yang dimiliki oleh state yang diberikan

10. bawah, kiri

pada posisi ini sebuah state yang diberikan dapat melahirkan 2 buah state anak. Dimana pada masing-masing state anak memiliki posisi blok blank dibawah dan dikiri dari posisi blok blank yang dimiliki oleh state yang diberikan

11. atas

pada posisi ini sebuah state yang diberikan dapat melahirkan 1 buah state anak. Dimana pada masing-masing state anak memiliki posisi blok blank diatas dari posisi blok blank yang dimiliki oleh state yang diberikan

12. bawah

pada posisi ini sebuah state yang diberikan dapat melahirkan 1 buah state anak. Dimana pada masing-masing state anak memiliki posisi blok blank dibawah dari posisi blok blank yang dimiliki oleh state yang diberikan

13. kiri

pada posisi ini sebuah state yang diberikan dapat melahirkan 1 buah state anak. Dimana pada masing-masing state anak memiliki posisi blok blank dikiri dari posisi blok blank yang dimiliki oleh state yang diberikan

14. kanan

pada posisi ini sebuah state yang diberikan dapat melahirkan 1 buah state anak. Dimana pada masing-masing state anak memiliki posisi blok blank dikanan dari posisi blok blank yang dimiliki oleh state yang diberikan

15. kiri, kanan

pada posisi ini sebuah state yang diberikan dapat melahirkan 2 buah state anak. Dimana pada masing-masing state anak memiliki posisi blok blank dikiri dan dikanan dari posisi blok blank yang dimiliki oleh state yang diberikan

16. tidak melahirkan state pohon anak.

pada posisi ini sebuah state yang diberikan dapat melahirkan 0 buah state anak. Dimana disini berarti semua state pohon solusi sudah dimasukkan ke dalam pohon.

Program Utama untuk simulasi N-Puzzle dengan algoritma BFS ini dapat dituliskan dengan pseudo-code seperti yang terlihat di bawah

```
int main(){
   /*kamus*/
   Queue Q;
   MovMat Saw, Sakh;
   /*algoritma*/
   InputStateAwal(&Saw);
   /*prosedur untuk memasukkan state awal*/
   InputStateAkhir(&Sakh);
   /*prosedur untuk memasukkan*/
   /*state akhir yang diinginkan*/
   AddElmtQueue(Saw);
   /*prosedur memasukkan state awal ke Queue*/
   /*dan menjadi head dari queue*/
   if (IsEqual(Saw, Sakh)){
     /*do nothing*/
   }
   else{
       do {
         CheckMoveBlank(Q.Head);
         /*prosedur yang memeriksa bisa*/
         /*kemana saja blok blank bisa bergerak*/
         MoveBlank(Q.Head);
         /*prosedur yang membentuk semua anak pohon dari*/
         /*Q.head berdasarkan penjelasan yg terdapat di atas*/
         /*dan meletakkannya pada belakang antrian*/
         GetNext(Q);
         /*mengambil elemen berikutnya dari queue*/
         /*dan menjadikannya sebagai head*/
         }while(IsEqual(Q.Head,Sakh));
   return 0;
}
```

- Algoritma DFS

Depth First Search pada simulasi ini menggunakan 2 struktur data. Struktur data yang pertama adalah struktur data Matriks dan yang kedua adalah struktur data Queue. Struktur data matriks digunakan untuk merepresentasikan Tampilan N-Puzzle di layar, sedangkan struktur data Queue digunakan untuk menampung state matriks yang kelak akan berubah-ubah.

Algoritma DFS hanya membuat pohon solusi secara secara mendalam, berbeda dengan BFS yang melahirkan semua kemungkinan pergerakkan dari blok blank, DFS hanya melahirkan satu anak pohon berdasarkan prioritas dari algoritma itu sendiri. Pada makalah ini prioritas itu adalah atas, bawah, kiri, kanan. Jadi selama pergerakan blok blank ke atas masih memungkinkan, maka pohon pencarian solusi akan selalu melahirkan state pohon anak dimana state anak akan memiliki blok blank diatas blok blank milik *parent*.

Terdapat kemampuan lebih pada algoritma DFS, yaitu kemampuan untuk melakukan *backtrack*, kemampuan ini berguna ketika suatu pohon sudah tidak mempunyai kemungkinan untuk membentuk anak pohon lagi atau anak pohon sudah mencapai kedalaman yang ditentukan oleh program.

Algoritma DFS ini menggunakan beberapa proses validasi. Proses validasi yang pertama adalah validasi apakah kedalaman pohon sudah mencapai batas yang diberikan oleh simulasi. Validasi ini memiliki 2 kemungkinan, jika true maka akan membentuk anak pohon dari state yang diberikan yang diberikan, jika false maka akan bactrack kembali ke pohon parent dari state yang diberikan.

Proses validasi yang kedua memeriksa jenis pergerakan blok blank yang akan dilakukan. Validasi ini memiliki 4 kemungkinan, pemeriksaan ini dilakukan secara kondisioanal. Hasil dari validasi ini ada 4 yaitu :

1. membentuk pohon anak dengan menggerakan blok *blank* ke atas

membentuk pohon anak dengan mengerakan blok blank ke atas, jika tidak bisa maka akan bergerak ke kondisi berikutnya.

2. membentuk pohon anak dengan menggerakan blok *blank* ke bawah

membentuk pohon anak dengan mengerakan blok *blank* ke bawah, jika tidak bisa maka akan bergerak ke kondisi berikutnya.

3. membentuk pohon anak dengan menggerakan blok *blank* ke kiri

membentuk pohon anak dengan mengerakan blok *blank* ke kiri, jika tidak bisa maka akan bergerak ke kondisi berikutnya.

4. membentuk pohon anak dengan menggerakan blok *blank* ke kanan

membentuk pohon anak dengan mengerakan blok blank ke atas, jika tidak bisa maka akan bergerak ke kondisi berikutnya.

5. tidak membentuk pohon anak

tidak membentuk pohon anak dan akan melakukan backtrack kembali ke pohon parent.

Program Utama untuk simulasi N-Puzzle dengan algoritma DFS ini dapat dituliskan dengan pseudo-code seperti yang terlihat di bawah

```
int main(){
   /*kamus*/
   Queue Q;
   MovMat Saw, Sakh;
   /*algoritma*/
   InputStateAwal(&Saw);
   /*prosedur untuk memasukkan state awal*/
   InputStateAkhir(&Sakh);
   /*prosedur untuk memasukkan*/
   /*state akhir yang diinginkan*/
   AddElmtQueue(Saw);
   /*prosedur memasukkan state awal ke Queue*/
   /*dan menjadi head dari queue*/
   if (IsEqual(Saw, Sakh)){
     /*do nothing*/
   }
   else{
       do {
         CheckMoveBlank(Q.Head);
         /*prosedur yang memeriksa bisa*/
         /*kemana saja blok blank bisa bergerak*/
         MoveBlank(Q.Head);
        /*prosedur yang membentuk semua anak pohon dari*/
         /*Q.head dan meletakkannya pada depan antrian*/
         GetHead(Q);
         /*mengambil elemen head dari queue*/
         }while(IsEqual(Q.Head,Sakh));
   return 0;
}
```

6. Contoh Pemecahan Persoalan

Diberikan sebuah permainan n-puzzle dengan N = 9 puzzle dengan state awal seperti yang diperlihatkan pada gambar di bawah

5	7	3
	2	4
1	6	8

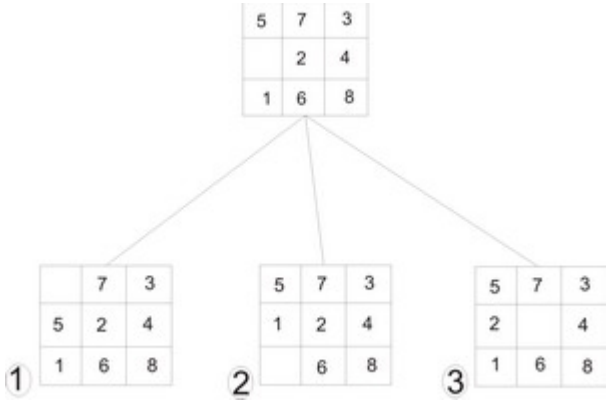
dan ketika diminta memasukkan state akhir,memasukkan kombinasi angka seperti yang diperlihatkan di bawah.

5	7	3
1	2	4
6		8

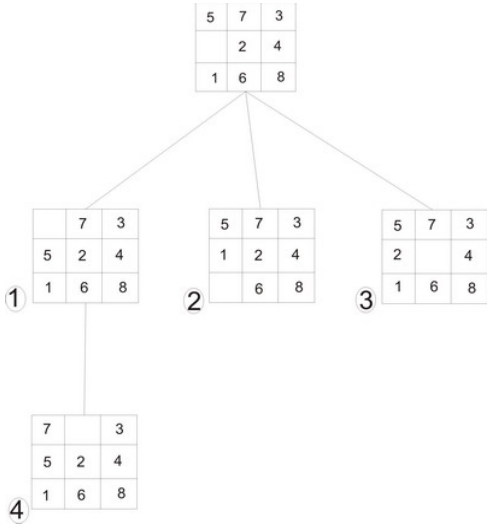
Penyelesaian permainan tersebut akan menggunakan 2 buah algoritma seperti yang sudah dijelaskan di atas yaitu BFS dan DFS

Breadth First Search

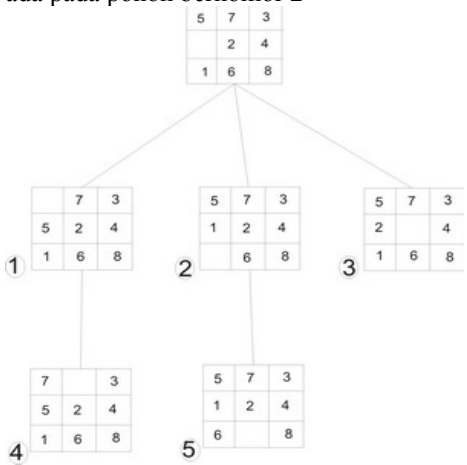
dengan menggunakan algoritma BFS, pertama-tama akan diperiksa apakah state awal sama dengan state akhir, karena tidak sama maka dari state awal akan dilahirkan 3 buah state anak yaitu hasil pergerakan ke atas, bawah dan kanan.



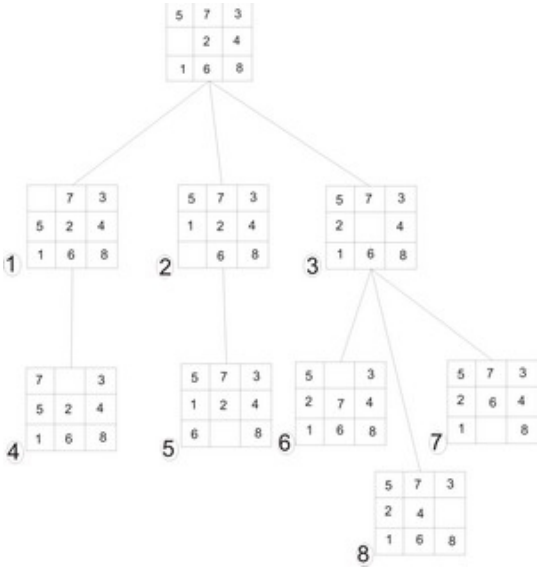
Pohon hasil bentukan yaitu no 1, 2 dan 3 akan dimasukkan sebagai antrian belakang dengan keterurutan yang sama. Berdasarkan implementasi algoritma yang sudah dijelaskan pada bab sebelumnya, maka kali ini state anak yang bernomor 1 akan menjadi head berikutnya dan akan diperiksa terlebih dahulu, apakah sama dengan state akhir, karena ternyata tidak sama maka akan dibentuk pohon anak dari state pohon nomor 1 itu.



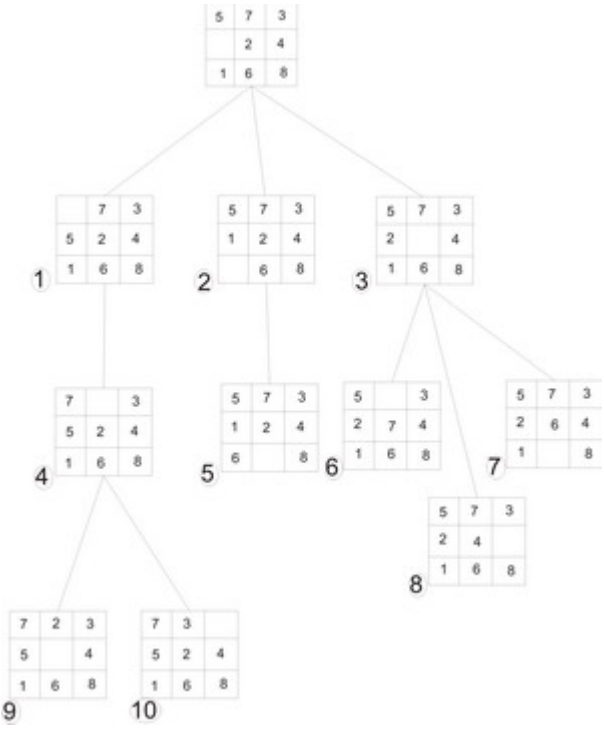
Pohon 4 akan dimasukkan sebagai antrian belakang. Kemudian head akan bergeser ke pohon 2, dimana akan diperiksa apakah pohon 2 merupakan state akhir karena tidak maka akan membentuk pohon anak state yang ada pada pohon bernomor 2



pohon 5 akan dimasukkan sebagai antrian belakang. head kemudian bergeser ke pohon anak no 3, kemudian pemeriksaan pun terjadi lagi apakah pohon nomor 3 sama dengan state akhir, karena tidak akan dibentuk pohon-pohon anak dari pohon anak no 3



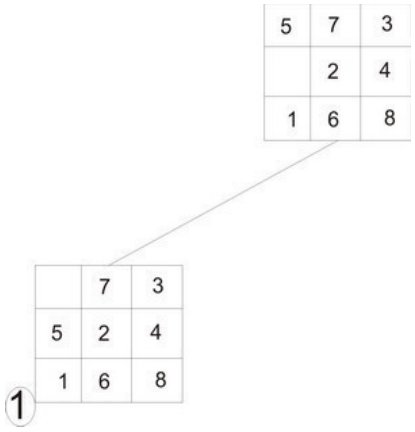
pohon 6,7,8 akan dimasukkan sebagai antrian belakang dengan keterurutan yang sama. Kemudian head bergerak kembali menuju elemen berikutnya, yaitu pohon anak yang bernomor 4, terjadi pemeriksaan apakah pohon 4 sama dengan state akhir, karena tidak sama maka dibentuk pohon anak dari pohon no 4 itu



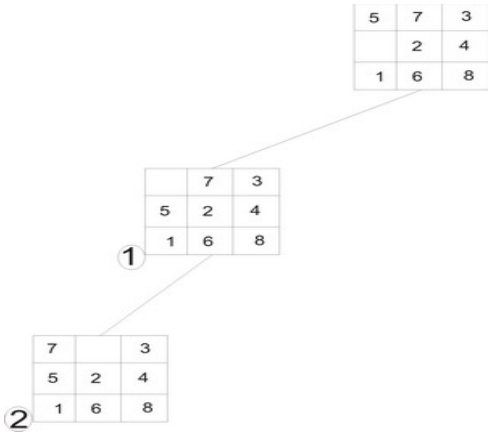
pohon 9 dan 10 akan dimasukkan ke dalam antrian bagian belakang dengan keterurutan yang sama. Kemudian pointer pohon anak bergerak kembali ke pohon 5 apakah pohon 5 sama dengan state akhir, ternyata state 5 sama dengan state akhir sehingga pencarian solusi pun dihentikan dengan pohon solusi yang dibentuk sama dengan gambar di atas. Solusi yang didapatkan berupa sebuah lintasan antar node yaitu node State Awal-2-5.

Depth First Search

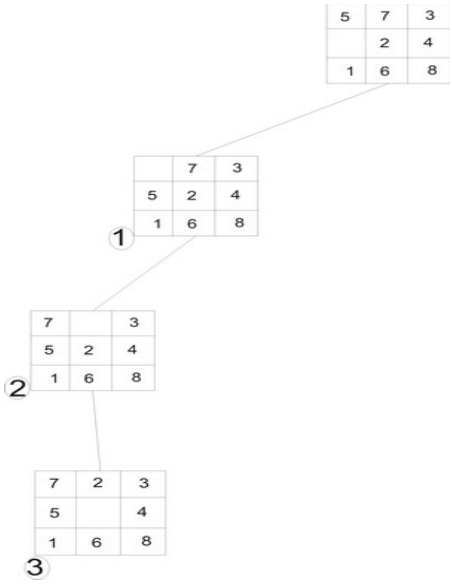
Dengan algoritma DFS dalam simulasi ini memiliki batas kedalaman 3, pertama-tama akan diperiksa apakah state awal sama dengan state akhir, karena tidak sama maka akan dibentuk pohon anak dari state awal dengan prioritas yang disebutkan di atas.



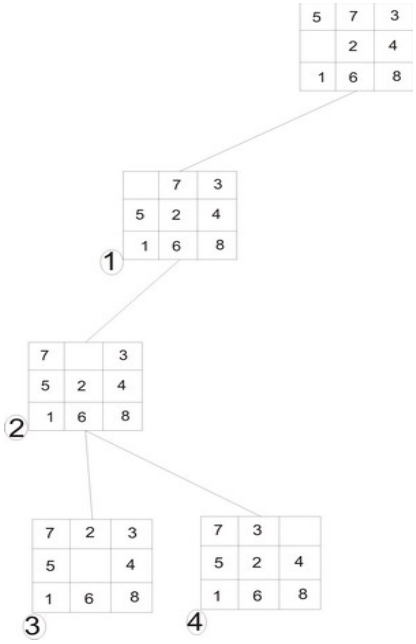
Pohon no 1 akan dimasukkan ke dalam awal antrian dan akan diperiksa apakah pohon no 1 merupakan state akhir dan apakah kedalaman dari pohon 1 sudah mencapai batas maksimum, karena bukan state akhir dan belum mencapai batas maksimum, maka akan melahirkan anak-anak pohon dari state pohon no 1 itu



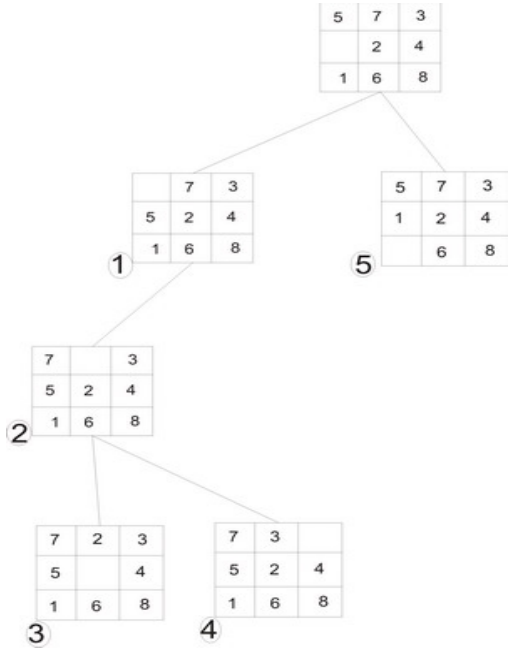
pohon 2 akan dimasukkan ke dalam head antrian kemudian akan diperiksa apakah sudah sama dengan state akhir atau sudah mencapai batas maksimum, karena belum mencapai state akhir dan belum mencapai batas, maka akan dibentuk anak pohon dari state pohon no 2.



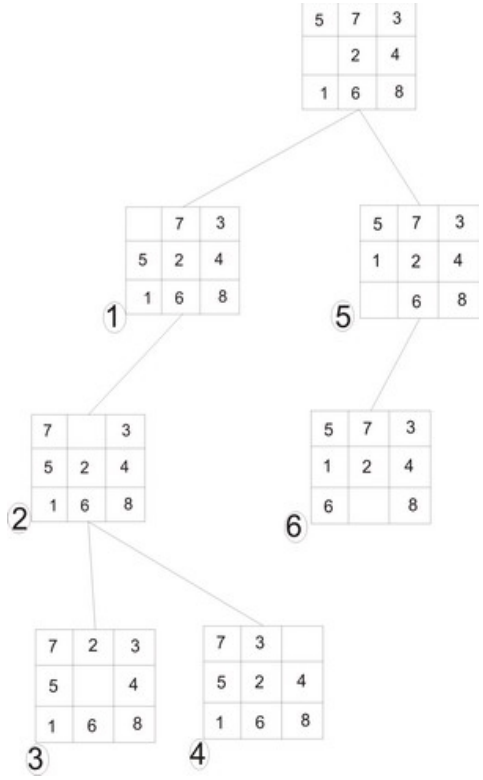
pohon nomor 3 ini akan dimasukkan ke dalam head antrian dan diperiksa apakah sudah sama dengan state akhir dan mencapai batas kedalaman, karena ternyata sudah mencapai batas kedalam maksimum dan bukan state akhir, maka akan melakukan *bactrack* kembali ke pohon 2 dan melakukan pembentukan anak pohon yang masih mungkin dibuat dari pohon 2.



pohon 4 akan dimasukkan ke dalam head antrian dan diperiksa apakah sudah mencapai batas maksimum dan sama dengan state akhir, karena sudah mencapai batas maksimum, maka akan melakukan *backtrack* kembali ke pohon 2 dan akan membentuk anak pohon yang masih mungkin dari pohon 2, karena tidak ada maka akan melakukan *backtrack* kembali ke pohon 1dan membentuk anak pohon yang mungkin dari pohon 1, karena kembali tidak ada maka akan melakukan *backtrack* ke state awal, dan membentuk anak-anak pohon yang masih mungkin dari state awal.



pohon 5 akan dimasukkan ke dalam head antrian dan diperiksa apakah sama dengan state akhir dan batas kedalaman pohon itu sudah maksimum, karena belum keduanya maka dilakukan pembentukan anak pohon dari pohon 5



pohon 6 kemudian dimasukkan ke dalam antrian dan diperiksa apakah sama dengan state akhir dan sudah mencapai kedalaman maksimum, karena sudah sama dengan state akhir maka pencarian solusi pun dihentikan dengan bentuk pohon pencarian solusi yang digambarkan seperti di atas. Solusi yang didapatkan berupa lintasan antar simpul, yaitu State Awal-5-6.

7. Kesimpulan

- Pada permainan atau simulasi yang terdiri dari langkah-langkah yang pasti menuju suatu *state final* ternyata dapat diimplementasikan sebuah pohon pencarian solusi, dimana akar dari pohon itu adalah sebuah state awal dan node-node dari pohon itu adalah berbagai kemungkinan state yang mungkin terjadi dari state awal, dan salah satu daun dari pohon pencarian solusi itu merupakan sebuah *state final* yang diinginkan.
- Algoritma yang dapat digunakan untuk pembentukan pohon pencarian solusi adalah algoritma BFS dan DFS, algoritma BFS melakukan pencarian secara meluas sedangkan DFS melakukan pencarian secara mendalam.
- Masing-masing implementasi algoritma pembentukan pohon solusi yaitu BFS dan DFS memiliki kelemahan dan kekuatan. Kekuatan BFS adalah karena melakukan pencarian secara meluas, maka dapat dipastikan suatu solusi final dapat ditemukan pada simpul pohon dengan

kedalaman berapapun, namun kelemahan dari BFS ini adalah menghabiskan resources yang ada pada komputer, karena melakukan pencarian secara meluas atau melahirkan sebuah anak pohon tanpa memastikan terlebih dahulu apakah anak pohon itu merupakan solusi final atau bukan.Kekuatan DFS adalah algoritma ini hemat sumber daya karena hanya melahirkan satu anak pohon dan kemudian memeriksanya apakah sudah sama dengan state final, namun kelemahan dari algoritma ini juga ada yaitu algoritma ini memiliki batas kedalaman yang tertentu dalam pohon pencarian solusi, sehingga terdapat kemungkinan solusi final tidak didapat, karena keterbatasan dalam pembentukan pohon solusi pada kedalaman tertentu.

- Terdapat perbedaan yang nyata pada algoritma secara teoritis dan implementasinya. Pada algoritma teoritis pemeriksaan solusi dalam suatu state dilakukan juga terhadap bentukan anak-anak pohon dari state tersebut, sedangkan pada implementasi algoritmanya pemeriksaan solusi dilakukan hanya pada state itu berada. Hal itu mengakibatkan suatu efek simulasi ini memboroskan memory dari komputer karena pemeriksaan hanya dilakukan pada state itu saja.

8. Daftar Pustaka

1. Munir , Rinaldi. 2004, Bahan Kuliah IF2151 Matematika Diskrit, Departemen Teknik Informatika, Institut Teknologi Bandung
2. Munir , Rinaldi. 2004, Bahan Kuliah IF2251 Strategi Algoritmik, Departemen Teknik Informatika, Institut Teknologi Bandung
3. Soiland, Stain. 2005, Graphing the N-Puzzle, *www.soiland.no/blog/tiles* Tanggal Akses : 2 Januari 2007
4. Schwartz, David. 2003, Graphs Review, *www.cs.cornell.edu/Courses/cs211/2006fa/Sections/S11/graphreview.pdf* Tanggal Akses : 2 Januari 2007