

KELOMPOK X

llllllll

8 PUZZLE INFORMED SEARCH

IMPLEMENTASI A*

KECERDASAN BUATAN F



KELOMPOK X

Line 1
Line 2
Line 3

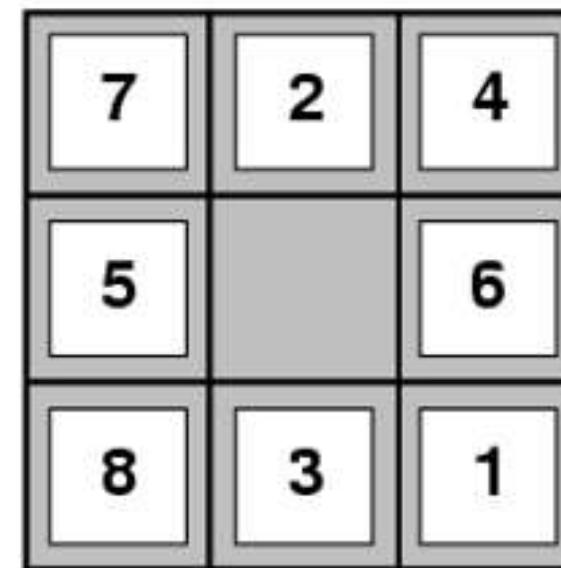
NADIF MUSTAFA / 5025211127
FREDERICK YONATANS / 5025211121
NIZAM HAKIM / 5025211209

KECERDASAN BUATAN F

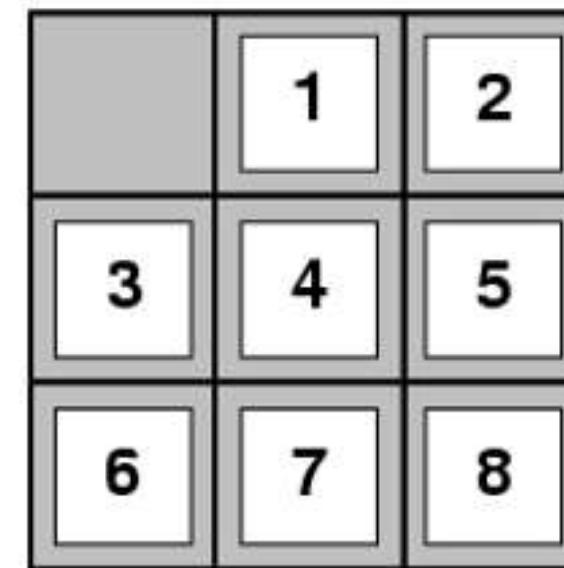


8 PUZZLE PROBLEM

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total Manhattan distance
(i.e., no. of squares from desired location of each tile)



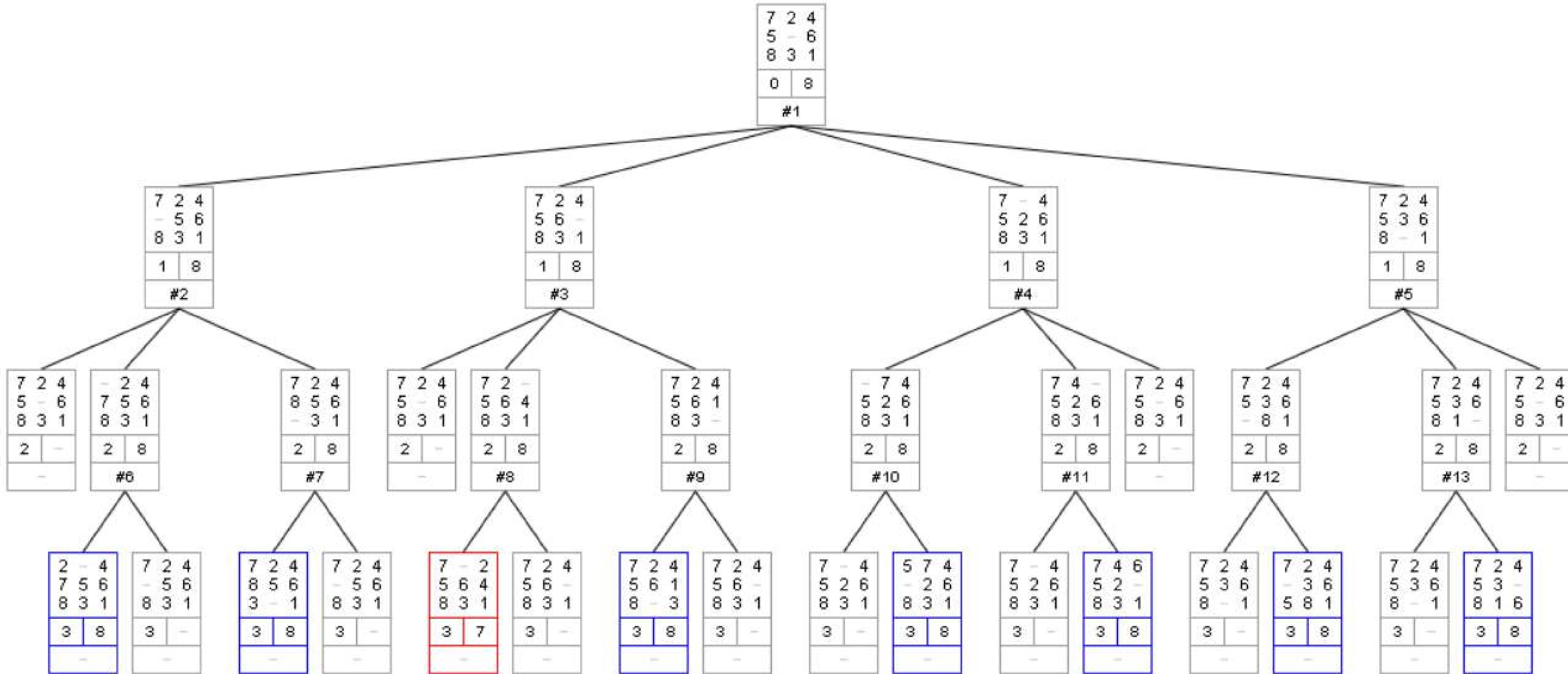
Start State

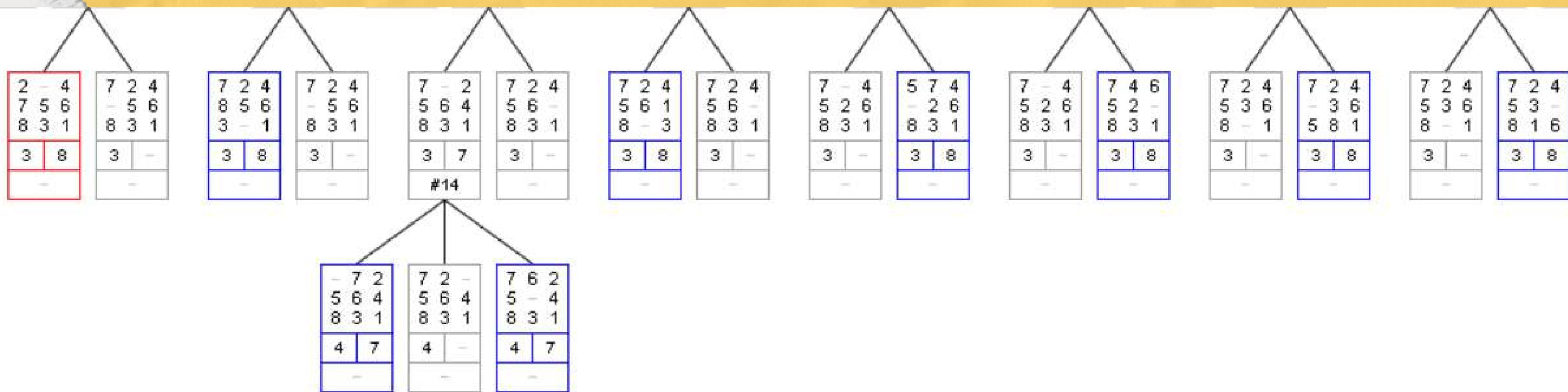


Goal State

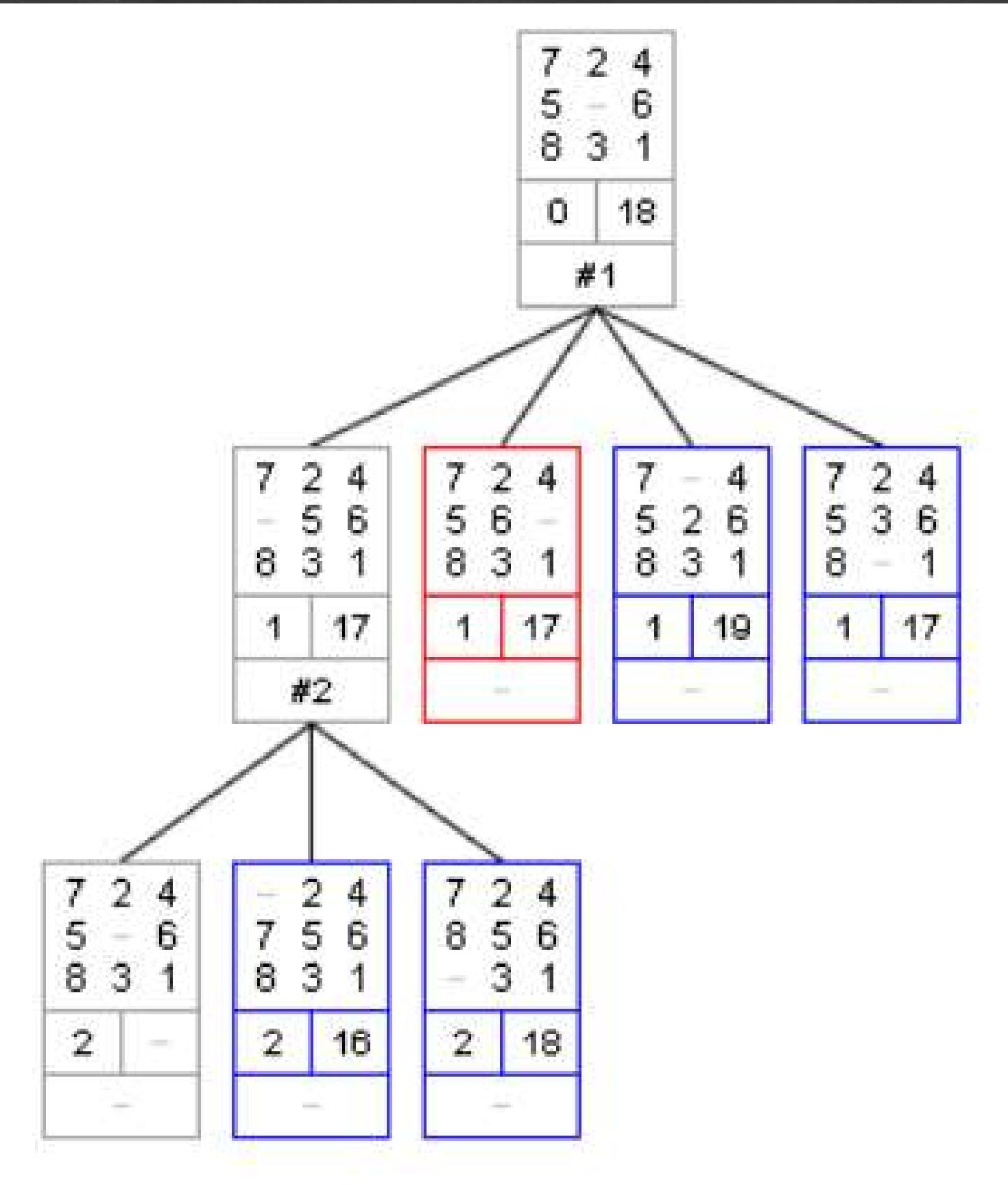
H1: MISPLACED TILES

numu





H2: MANHATTAN DISTANCE



PENJELASAN KODE UNTUK H1 DAN H2



olee

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 #define pqueue priority_queue
4 #define endl '\n'
5
6 typedef struct
7 {
8     string str;
9     int dist, Tdist;
10 }
11 state;
12
13 class cmp
14 {
15 public:
16     bool operator()(state a, state b)
17     {
18         if(b.Tdist == a.Tdist)
19             return b.dist < a.dist;
20         return b.Tdist < a.Tdist;
21     }
22 };
23
```

Berikut adalah penjelasan dari kedua atribut tersebut:

- Tdist: adalah nilai $f(n)$ pada algoritma A* yaitu biaya total yang sudah ditempuh (nilai $g(n)$) ditambah estimasi biaya yang diperlukan untuk mencapai titik tujuan dari simpul saat ini (nilai $h(n)$). Semakin kecil nilai Tdist, maka semakin dekat simpul saat ini dengan solusi yang diinginkan.
- dist: adalah jarak dari simpul saat ini ke simpul tujuan. Nilai dist digunakan sebagai nilai heuristik ($h(n)$) pada algoritma A*. Semakin kecil nilai dist, maka semakin dekat simpul saat ini dengan simpul tujuan.

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 #define pqueue priority_queue
4 #define endl '\n'
5
6 typedef struct
7 {
8     string str;
9     int dist, Tdist;
10 }
11 state;
12
13 class cmp
14 {
15 public:
16     bool operator()(state a, state b)
17     {
18         if(b.Tdist == a.Tdist)
19             return b.dist < a.dist;
20         return b.Tdist < a.Tdist;
21     }
22 };
23
```

Kode yang diberikan adalah sebuah class cmp yang digunakan sebagai komparator pada penggunaan priority queue dalam implementasi algoritma A* (A star).

Komparator ini membandingkan dua objek state, yaitu objek a dan objek b berdasarkan dua nilai atribut yaitu Tdist dan dist.

Jika nilai atribut Tdist pada objek b sama dengan nilai atribut Tdist pada objek a, maka komparator akan membandingkan nilai atribut dist pada kedua objek tersebut. Jika nilai atribut Tdist pada objek b lebih kecil dari nilai atribut Tdist pada objek a, maka objek b dianggap lebih kecil dan akan diletakkan di posisi lebih awal pada priority queue.

```
23
24 map <string, int> Sdist, heur;
25 map <string, string> prev;
26 set <string> vis;
27 int loop;
28
29 int getPos(int idx, int dest)
30 {
31     int row = idx / 3;
32     int col = idx % 3;
33
34     if(dest == 0)
35         return (col == 0 ? -1 : 3 * row + col - 1); // Left
36     else if(dest == 1)
37         return (col == 2 ? -1 : 3 * row + col + 1); // Right
38     else if(dest == 2)
39         return (row == 0 ? -1 : 3 * (row - 1) + col); // Top
40     else
41         return (row == 2 ? -1 : 3 * (row + 1) + col); // Bottom
42 }
43
```

Kode yang diberikan adalah sebuah fungsi `getPos()` yang digunakan pada implementasi algoritma A* (A star) untuk mencari posisi (indeks) kotak kosong pada papan permainan puzzle 8.

Fungsi ini menerima dua parameter yaitu `idx` yang merepresentasikan indeks kotak yang sedang diproses pada saat ini dan `dest` yang merepresentasikan arah perpindahan kotak kosong (0 untuk kiri, 1 untuk kanan, 2 untuk atas, dan 3 untuk bawah).

Pertama, fungsi ini akan menghitung posisi baris dan kolom dari kotak pada indeks `idx` dengan menggunakan operasi pembagian dan modulus.

Jika `dest` adalah 0 (kiri), maka fungsi akan mengecek apakah kotak tersebut berada di kolom paling kiri atau tidak. Jika tidak, maka fungsi akan mengembalikan indeks kotak di sebelah kiri kotak tersebut. Jika ya, maka fungsi akan mengembalikan nilai -1, menandakan bahwa perpindahan tidak mungkin dilakukan.

```
23
24 map <string, int> Sdist, heur;
25 map <string, string> prev;
26 set <string> vis;
27 int loop;
28
29 int getPos(int idx, int dest)
30 {
31     int row = idx / 3;
32     int col = idx % 3;
33
34     if(dest == 0)
35         return (col == 0 ? -1 : 3 * row + col - 1); // Left
36     else if(dest == 1)
37         return (col == 2 ? -1 : 3 * row + col + 1); // Right
38     else if(dest == 2)
39         return (row == 0 ? -1 : 3 * (row - 1) + col); // Top
40     else
41         return (row == 2 ? -1 : 3 * (row + 1) + col); // Bottom
42 }
43
```

Jika dest adalah 1 (kanan), maka fungsi akan mengecek apakah kotak tersebut berada di kolom paling kanan atau tidak. Jika tidak, maka fungsi akan mengembalikan indeks kotak di sebelah kanan kotak tersebut. Jika ya, maka fungsi akan mengembalikan nilai -1.

Jika dest adalah 2 (atas), maka fungsi akan mengecek apakah kotak tersebut berada di baris paling atas atau tidak. Jika tidak, maka fungsi akan mengembalikan indeks kotak di atas kotak tersebut. Jika ya, maka fungsi akan mengembalikan nilai -1.

Jika dest adalah 3 (bawah), maka fungsi akan mengecek apakah kotak tersebut berada di baris paling bawah atau tidak. Jika tidak, maka fungsi akan mengembalikan indeks kotak di bawah kotak tersebut. Jika ya, maka fungsi akan mengembalikan nilai -1.

```
44 int getHeur(string cur, string finish)
45 {
46     if(heur.count(cur))
47         return heur[cur];
48     int res = 0;
49
50     for(int i = 0; i < 9; ++i){
51         if(finish[i] != '0')
52             res += (finish[i] != cur[i]);
53     }
54     return heur[cur] = res;
55 }
```

Kode yang diberikan adalah sebuah fungsi `getHeur()` yang digunakan pada implementasi algoritma A* (A star) untuk menghitung nilai heuristik ($h(n)$) dari sebuah simpul pada pohon pencarian.

Fungsi ini menerima dua parameter yaitu `cur` yang merepresentasikan konfigurasi papan permainan puzzle 8 pada simpul saat ini dan `finish` yang merepresentasikan konfigurasi papan permainan puzzle 8 pada simpul tujuan.

Pertama, fungsi ini akan mengecek apakah nilai heuristik untuk simpul saat ini sudah pernah dihitung atau belum dengan menggunakan fungsi `count()` pada objek map `heur`. Jika sudah pernah dihitung, maka nilai heuristik tersebut akan langsung dikembalikan.

```
44 int getHeur(string cur, string finish)
45 {
46     if(heur.count(cur))
47         return heur[cur];
48     int res = 0;
49
50     for(int i = 0; i < 9; ++i){
51         if(finish[i] != '0')
52             res += (finish[i] != cur[i]);
53     }
54     return heur[cur] = res;
55 }
```

Jika belum pernah dihitung, maka fungsi ini akan menghitung nilai heuristik dengan cara menghitung banyaknya kotak yang tidak berada pada posisi yang benar (tidak sama dengan posisi pada simpul tujuan). Pada setiap iterasi perulangan, fungsi akan membandingkan nilai dari kotak pada posisi yang sama antara simpul saat ini dan simpul tujuan. Jika nilai tersebut tidak sama, maka fungsi akan menambahkan 1 pada variabel res yang merepresentasikan jumlah kotak yang tidak berada pada posisi yang benar.

Terakhir, nilai heuristik untuk simpul saat ini akan disimpan pada objek map heur agar tidak perlu dihitung kembali pada saat yang sama. Fungsi akan mengembalikan nilai heuristik yang sudah dihitung.

```
57 void printPuzzle(string str)
58 {
59     for(int i = 0; i <= 2; ++i){
60         for(int j = 0; j <= 2; ++j)
61             cout << str[3*i+j] << " ";
62         cout << endl;
63     }
64     cout << endl;
65 }
66 }
```

Kode yang diberikan adalah sebuah fungsi `printPuzzle()` yang digunakan untuk mencetak konfigurasi papan permainan puzzle 8 ke layar.

Fungsi ini menerima satu parameter yaitu `str` yang merepresentasikan konfigurasi papan permainan puzzle 8 dalam bentuk string. Konfigurasi papan permainan ini disusun dalam bentuk matriks 3x3.

Pada setiap iterasi perulangan, fungsi ini akan mencetak isi kotak pada posisi yang sesuai pada matriks 3x3. Setiap baris akan dicetak pada baris yang berbeda pada layar, kemudian fungsi akan mencetak baris kosong untuk memisahkan antara satu konfigurasi dengan konfigurasi yang lain.

```

67 void aStar(string start, string finish)
68 {
69     pqueue<state, vector<state>, cmp> pq;
70     Sdist[start] = 0;
71
72     state init = {start, 0, getHeur(start, finish)};
73     pq.push(init);
74
75     while(!pq.empty())
76     {
77         state curState = pq.top();
78         pq.pop();
79
80         if(vis.count(curState.str))
81             continue;
82         vis.insert(curState.str);
83
84         printPuzzle(curState.str);
85         cout << "g(n) : " << curState.dist << endl;
86         cout << "h(n) : " << getHeur(curState.str, finish) << endl << endl;
87         cout << string(25, '-') << endl << endl;
88
89         if(curState.str == finish)
90             break;
91         int kosong = -1;
92

```

```

93     for(int i = 0; i < 9; ++i){
94         if(curState.str[i] == '0'){
95             kosong = i;
96             break;
97         }
98     }
99     for(int i = 0; i < 4; ++i)
100    {
101        string nextStr = curState.str;
102        int nextPos = getPos(kosong, i);
103        int nextHeur = 0;
104
105        if(nextPos != -1)
106        {
107            swap(nextStr[kosong], nextStr[nextPos]);
108            nextHeur = getHeur(nextStr, finish);
109
110            if(!Sdist.count(nextStr) || curState.dist + 1 < Sdist[nextStr])
111            {
112                Sdist[nextStr] = curState.dist + 1;
113                prev[nextStr] = curState.str;
114
115                state nextState = curState;
116                nextState.str = nextStr;
117                nextState.dist++;
118                nextState.Tdist = nextState.dist + heur[nextStr];
119
120                pq.push(nextState);
121            }
122        }
123    }
124 } loop++;
125
126
127

```

Kode di atas adalah implementasi dari algoritma A* untuk menyelesaikan puzzle 8. Fungsi ini menerima dua parameter, yaitu string start dan string finish yang merepresentasikan konfigurasi awal dan konfigurasi tujuan dari papan permainan puzzle 8.

Pertama-tama, fungsi ini menginisialisasi jarak yang ditempuh dari konfigurasi awal (start) dengan nol. Kemudian, membuat sebuah state dengan konfigurasi awal, jarak yang ditempuh dari awal (dist), dan heuristik dari konfigurasi awal ke konfigurasi tujuan (Tdist), lalu menyimpannya pada priority queue.

```

67 void aStar(string start, string finish)
68 {
69     pqueue<state, vector<state>, cmp> pq;
70     Sdist[start] = 0;
71
72     state init = {start, 0, getHeur(start, finish)};
73     pq.push(init);
74
75     while(!pq.empty())
76     {
77         state curState = pq.top();
78         pq.pop();
79
80         if(vis.count(curState.str))
81             continue;
82         vis.insert(curState.str);
83
84         printPuzzle(curState.str);
85         cout << "g(n) : " << curState.dist << endl;
86         cout << "h(n) : " << getHeur(curState.str, finish) << endl << endl;
87         cout << string(25, '-') << endl << endl;
88
89         if(curState.str == finish)
90             break;
91         int kosong = -1;
92
93         for(int i = 0; i < 9; ++i){
94             if(curState.str[i] == '0'){
95                 kosong = i;
96                 break;
97             }
98         }
99         for(int i = 0; i < 4; ++i)
100        {
101             string nextStr = curState.str;
102             int nextPos = getPos(kosong, i);
103             int nextHeur = 0;
104
105             if(nextPos != -1)
106             {
107                 swap(nextStr[kosong], nextStr[nextPos]);
108                 nextHeur = getHeur(nextStr, finish);
109
110                 if(!Sdist.count(nextStr) or curState.dist + 1 < Sdist[nextStr])
111                 {
112                     Sdist[nextStr] = curState.dist + 1;
113                     prev[nextStr] = curState.str;
114
115                     state nextState = curState;
116                     nextState.str = nextStr;
117                     nextState.dist++;
118                     nextState.Tdist = nextState.dist + heur[nextStr];
119
120                     pq.push(nextState);
121                 }
122             }
123         }
124     }
125     loop++;
126 }

```

Selanjutnya, fungsi ini akan melakukan perulangan selama priority queue tidak kosong. Pada setiap iterasi, state pada bagian atas priority queue diambil dan dihapus dari priority queue. Kemudian, dilakukan pengecekan apakah konfigurasi tersebut sudah pernah dikunjungi atau belum. Jika sudah, maka akan dilewati dan lanjut ke iterasi selanjutnya. Jika belum, konfigurasi tersebut akan ditandai sebagai sudah dikunjungi dan dicetak ke layar menggunakan fungsi `printPuzzle()`. Setelah itu, fungsi akan mencetak nilai $g(n)$ (jarak yang ditempuh dari konfigurasi awal ke konfigurasi saat ini) dan $h(n)$ (heuristik dari konfigurasi saat ini ke konfigurasi tujuan) ke layar.

Jika konfigurasi saat ini sudah sama dengan konfigurasi tujuan, maka proses pencarian akan dihentikan dan fungsi akan keluar dari perulangan. Jika belum, fungsi akan mencari posisi kotak kosong pada konfigurasi saat ini dan mencoba untuk memindahkan kotak ke posisi yang mungkin (kiri, kanan, atas, dan bawah).

```

67 void aStar(string start, string finish)
68 {
69     pqueue<state, vector<state>, cmp> pq;
70     Sdist[start] = 0;
71
72     state init = {start, 0, getHeur(start, finish)};
73     pq.push(init);
74
75     while(!pq.empty())
76     {
77         state curState = pq.top();
78         pq.pop();
79
80         if(vis.count(curState.str))
81             continue;
82         vis.insert(curState.str);
83
84         printPuzzle(curState.str);
85         cout << "g(n) : " << curState.dist << endl;
86         cout << "h(n) : " << getHeur(curState.str, finish) << endl << endl;
87         cout << string(25, '-') << endl << endl;
88
89         if(curState.str == finish)
90             break;
91         int kosong = -1;
92

```

```

93     for(int i = 0; i < 9; ++i){
94         if(curState.str[i] == '0'){
95             kosong = i;
96             break;
97         }
98     }
99     for(int i = 0; i < 4; ++i)
100    {
101        string nextStr = curState.str;
102        int nextPos = getPos(kosong, i);
103        int nextHeur = 0;
104
105        if(nextPos != -1)
106        {
107            swap(nextStr[kosong], nextStr[nextPos]);
108            nextHeur = getHeur(nextStr, finish);
109
110            if(!Sdist.count(nextStr) or curState.dist + 1 < Sdist[nextStr])
111            {
112                Sdist[nextStr] = curState.dist + 1;
113                prev[nextStr] = curState.str;
114
115                state nextState = curState;
116                nextState.str = nextStr;
117                nextState.dist++;
118                nextState.Tdist = nextState.dist + heur[nextStr];
119
120                pq.push(nextState);
121            }
122        }
123    }
124 }
125 }
126 }
127

```

Pada setiap kemungkinan posisi, fungsi akan membuat sebuah string baru yang merepresentasikan konfigurasi yang mungkin tersebut. Kemudian, fungsi akan menghitung heuristik dari konfigurasi tersebut ke konfigurasi tujuan. Jika konfigurasi tersebut belum pernah dikunjungi atau jarak yang ditempuh dari konfigurasi awal lebih pendek daripada jarak yang sudah pernah disimpan pada map Sdist, maka konfigurasi tersebut akan disimpan pada map Sdist, konfigurasi sebelumnya (yang membawa ke konfigurasi ini) akan disimpan pada map prev, dan state baru dengan konfigurasi tersebut, jarak yang ditempuh, dan heuristik dari konfigurasi tersebut ke konfigurasi tujuan akan disimpan pada priority queue.

Pada akhirnya, fungsi akan mencetak jumlah iterasi yang dibutuhkan untuk menyelesaikan puzzle 8.

```
128 void backtrack(string cur, int depth)
129 {
130     if(depth > 0)
131         backtrack(prev[cur], depth - 1);
132     cout << "Step " << depth << endl;
133     printPuzzle(cur);
134 }
135
```

Code backtrack merupakan bagian dari solusi untuk problem A* search. Fungsi ini akan dipanggil setelah pencarian A* selesai dan goal state berhasil ditemukan. Tujuannya adalah untuk menampilkan langkah-langkah yang diambil dari initial state sampai goal state.

Fungsi backtrack menerima dua parameter yaitu cur dan depth. cur merepresentasikan current state saat ini yang akan dicetak, dan depth merepresentasikan kedalaman dari state saat ini dalam search tree (yaitu jarak langkah dari initial state).

```
128 void backtrack(string cur, int depth)
129 {
130     if(depth > 0)
131         backtrack(prev[cur], depth - 1);
132     cout << "Step " << depth << endl;
133     printPuzzle(cur);
134 }
135
```

Pada dasarnya, backtrack akan melakukan rekursi dengan memanggil dirinya sendiri hingga mencapai kedalaman state yang paling awal ($\text{depth} = 0$). Ketika kedalaman state yang paling awal sudah dicapai, fungsi akan mencetak langkah-langkah yang diambil dari initial state sampai goal state dengan urutan yang benar (sesuai dengan urutan pencarian A*).

Setelah mencapai kedalaman state yang paling awal ($\text{depth} = 0$), fungsi akan mencetak langkah terakhir dan goal state. Setiap langkah akan dicetak dengan menggunakan fungsi `printPuzzle` untuk menampilkan state dalam bentuk grid 3x3.

```
136 int main()
137 {
138     ios_base::sync_with_stdio(false);
139     cin.tie(NULL);
140     cout.tie(NULL);
141
142     string start = "724506831";
143     string finish = "012345678";
144     aStar(start, finish);
145
146     cout << "Total Distance : " << Sdist[finish] << endl;
147     cout << "Total Loop : " << loop << endl << endl;
148     backtrack(finish, 26);
149     return 0;
150 }
```

```
44 int manhattan(int p1, int p2)
45 {
46     int r1 = p1 / 3,
47         c1 = p1 % 3;
48     int r2 = p2 / 3,
49         c2 = p2 % 3;
50     return abs(r1 - r2) + abs(c1 - c2);
51 }
52
53 int getHeur(string cur, string finish)
54 {
55     if(heur.count(cur))
56         return heur[cur];
57     int res = 0;
58
59     for(int i = 0; i < 9; ++i)
60     {
61         if(finish[i] == '0')
62             continue;
63         for(int j = 0; j < 9; ++j)
64         {
65             if(finish[i] == cur[j]){
66                 res += manhattan(i, j);
67                 break;
68             }
69         }
70     }
71     return heur[cur] = res;
72 }
```

Fungsi "manhattan" menghitung jarak Manhattan antara dua posisi pada kotak 3x3. Fungsi menerima dua parameter, yaitu p1 dan p2, yang merupakan posisi dalam array 1 dimensi dari elemen pada kotak. Fungsi ini akan mengembalikan nilai jarak Manhattan antara kedua posisi tersebut, yaitu jumlah selisih jarak baris dan kolom antara kedua posisi tersebut.

```
44 int manhattan(int p1, int p2)
45 {
46     int r1 = p1 / 3,
47         c1 = p1 % 3;
48     int r2 = p2 / 3,
49         c2 = p2 % 3;
50     return abs(r1 - r2) + abs(c1 - c2);
51 }
52
53 int getHeur(string cur, string finish)
54 {
55     if(heur.count(cur))
56         return heur[cur];
57     int res = 0;
58
59     for(int i = 0; i < 9; ++i)
60     {
61         if(finish[i] == '0')
62             continue;
63         for(int j = 0; j < 9; ++j)
64         {
65             if(finish[i] == cur[j]){
66                 res += manhattan(i, j);
67                 break;
68             }
69         }
70     }
71     return heur[cur] = res;
72 }
```

Fungsi ini menghitung heuristik atau estimasi jarak dari posisi awal ke posisi akhir pada permasalahan puzzle 8 angka dengan menggunakan metode Manhattan Distance.

Pertama-tama, fungsi ini memeriksa apakah heuristik untuk keadaan (state) saat ini (cur) sudah pernah dihitung sebelumnya. Jika iya, maka fungsi mengembalikan nilai heuristik tersebut.

Jika belum, fungsi menghitung heuristik baru dengan melakukan loop pada kedua state (cur dan finish). Pada setiap iterasi, fungsi memeriksa apakah digit yang berada pada posisi i pada state akhir (finish) bukanlah angka 0 (kosong). Jika iya, fungsi melakukan loop kembali pada kedua state untuk mencari posisi j di mana digit tersebut berada pada state saat ini (cur). Kemudian, fungsi menggunakan metode Manhattan Distance pada posisi i dan j untuk menghitung jarak antara kedua posisi tersebut. Hasil dari setiap iterasi dijumlahkan pada variabel res.

Setelah loop selesai, nilai heuristik res disimpan pada map heur untuk state cur, kemudian fungsi mengembalikan nilai res tersebut.

H1

1 0 2
3 4 5
6 7 8

g(n) : 25
h(n) : 1

0 1 2
3 4 5
6 7 8

g(n) : 26
h(n) : 0

Total Distance : 26
Total Loop : 44988

H2

1 0 2
3 4 5
6 7 8

g(n) : 25
h(n) : 1

0 1 2
3 4 5
6 7 8

g(n) : 26
h(n) : 0

Total Distance : 26
Total Loop : 4085

H1 VS H2



TERIMA KASIH