TUGAS

Penyelesaian 8-Puzzle menggunakan Metode A Star (A*)

WIDHI SULISTYO

18/433802/PPA/05618

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS GADJAH MADA

YOGYAKARTA

Penjelasan Program :

1. Keterangan file :

Nama File	Keterangan
Program_utama.c	Merupakan file utama/ program utama (Implementasi A star)
file_list.h	Menjelaskan struktur data untuk menyimpan daftar node serta operasinya
file_node.h	Menjelaskan struktur data dan mendefinisikan operasi Node.
file_state.h	Menjelaskan struktur data untuk mewakili keadaan permainan. Operasi "state" juga didefinisikan di sini.
file_Inputoutput.h	Fungsi input / output

a. State (state.h: struct State)

Permainan ini merupakan turunan dari semua konfigurasi papan 3 x 3, angka dimulai dari 0 hingga 8 yang akan dimasukan dalam 9 blok .

```
typedef struct State {
    Move action;
    char board[3][3];      //menyiapkan papan berbentuk array 3 x 3
} State;
```

Fungsi	Penjelasan
`State* createState(State *s, Move m);`	Membuat dan mengembalikan state baru yang berasal dari argumen, jika m adalah tindakan yang valid dari s. Jika tidak, NULL dikembalikan.
`void destroyState(State **s);`	Cukup deallocates state yang dibuat oleh createState (). Dan set * s ke NULL.
`int manhattanDist(State *c, State * g);`	Mengevaluasi jarak manhattan (nilai heuristik) dari konfigurasi state yang ditentukan c berdasarkan keadaan tujuan g.

2. Operators (state.h: struct Move)

Operator adalah semua tindakan yang tersedia sesuai dengan mekanisme permainan, dalam permainan puzzle ini tindakan yang mungkin dilakukan adalah ke atas, bawah, kiri, dan kanan.

```
typedef enum Move {
    UP, DOWN, LEFT, RIGHT,
    NOT_APPLICABLE
} Move;
```

3. Nodes (node.h: struct Node)

Sebuah simpul merupakan pohon pencarian. Ini berisi informasi tentang induknya, anak, state, cost heuristik, dan kedalaman.

```
typedef struct Node Node;
struct Node {
    unsigned int depth; //kedalaman node dari root. Untuk pencarian A
*
    unsigned int hCost; //biaya heuristik dari node
    State *state;
    Node *parent;
    NodeList *children;
};
```

Catatan :

Dalam permainan ini, biaya jalan node akan seberapa jauh dari akar pohon; yaitu, semakin dalam simpul, semakin mahal harganya. Karena dasar untuk menetapkan biaya jalur ke suatu titik berkorelasi dengan kedalamannya.

Fungsi	Penjelasan
`Node* createNode(int d, int h, State*s, Node *p);`	Membuat dan mengembalikan node baru dengan kedalaman d, nilai heuristik h, state s, dan node induk p.
`void destroyTree(Node *n);`	Membatalkan seluruh subtree node dari node root n.
`NodeList* getChildren(Node *p, State *g);`	Memperluas p node dan mengembalikan daftar node turunan dengan h-cost terhitung berdasarkan pada status tujuan g.
`int totalCost(Node *n);`	Mengevaluasi dan mengembalikan biaya total node dengan menambahkan nilai heuristik ke biaya jalurnya.

4. Linked List Container (list.h: struct NodeList)

   Struktur ini berfungsi sebagai pembungkus untuk daftar ListNodes. Digunakan untuk membawa daftar tipe Node. Ini berisi jumlah node dalam daftar, serta pointer ke kepala dan simpul ekor.

5. Linked List Nodes (list.h: struct ListNode)

   Struktur ini adalah bagian dari NodeList. Ini memiliki pointer ke ListNode sebelumnya dan berikutnya dalam daftar, dan dapat menyimpan paling banyak satu Node.

```
typedef struct ListNode {
    Node *currNode;
    struct ListNode *prevNode;
    struct ListNode *nextNode;
} ListNode;
```

Fungsi	Penjelasan
`char pushNode(Node *n, NodeList** l);`	Menambahkan simpul pohon permainan n ke daftar utama l. Mengembalikan 1 keberhasilan; 0 pada kegagalan.
`Node* popNode(NodeList** l);`	Menghapus simpul di ekor daftar l dan mengembalikan pointer ke sana.
`void pushList(NodeList **t, NodeList*l);`	Menambahkan daftar simpul t, ke daftar l pada ekor yang terakhir.
`void pushListInOrder(NodeList **t, NodeList *l);`	Sama seperti pushList () kecuali bahwa node yang akan ditambahkan ke l diurutkan berdasarkan biaya f-node.

```
struct NodeList {
    unsigned int nodeCount;
    ListNode *head;
    ListNode *tail;
};
```

6. Solution Path (list.h: struct SolutionPath)

   Struktur ini membuat daftar tindakan untuk dieksekusi untuk mencapai tujuan, yaitu jalur solusi.

   Implementasinya menggunakan daftar yang terhubung tunggal.

```
typedef struct SolutionPath {
    Move action;
    struct SolutionPath *next;
} SolutionPath;
```

Fungsi	Penjelasan
`void destroySolution(SolutionPath **l);`	Membatalkan memori yang dialokasikan daftar l.

7. Global Counters

Seperti yang diinstruksikan, program harus dapat mengumpulkan dan mencetak urutan langkah yang terkait dengan solusi dan statistik berikut:

- Jumlah total node diperluas
- Jumlah total simpul yang dihasilkan
- Panjang jalur solusi (jumlah gerakan)
- Urutan gerakan yang sesuai dengan solusi (misalnya kiri, kanan, atas, bawah)

Oleh karena itu, selain urutan langkah, kami memiliki variabel global berikut dengan tujuan berikut:

Variabel	Keterangan
`unsigned int nodesExpanded`	Rekam jumlah node saat ini diperluas
`unsigned int nodesGenerated`	Rekam jumlah node yang dihasilkan saat ini
`unsigned int solutionLength`	Menghitung jumlah sisi di jalur solusi

Selain itu, untuk mengukur kinerja algoritma pencarian A * untuk perbandingan, tambahan runtime variabel global tipe ganda diatur untuk mencatat waktu eksekusi algoritma dalam milidetik. Di sisi lain, untuk mendapatkan ruang yang dikonsumsi oleh masing-masing algoritma untuk input yang diberikan, nodeExpanded dikalikan dengan ukuran struct Node.

Hasil Percobaan :

```
------------------------------------------------------------------------
--
Perintah:
    Masukan status awal dan status akhir dari 8-puzzle
    Bilangan integer 0-8
------------------------------------------------------------------------
--
KONDISI AWAL:
    POSISI[0][0]: 2
    POSISI[0][1]: 3
    POSISI[0][2]: 1
    POSISI[1][0]: 5
    POSISI[1][1]: 4
    POSISI[1][2]: 6
    POSISI[2][0]: 8
    POSISI[2][1]: 7
    POSISI[2][2]: 0


KONDISI AKHIR / GOAL:
    POSISI[0][0]: 1
    POSISI[0][1]: 2
    POSISI[0][2]: 3
    POSISI[1][0]: 4
    POSISI[1][1]: 5
    POSISI[1][2]: 6
    POSISI[2][0]: 7
    POSISI[2][1]: 8
    POSISI[2][2]: 0

KONDISI AWAL:
+---+---+---+
| 2 | 3 | 1 |
+---+---+---+
| 5 | 4 | 6 |
+---+---+---+
| 8 | 7 | 0 |
+---+---+---+
GOAL / KONDISI AKHIR:
+---+---+---+
| 1 | 2 | 3 |
+---+---+---+
| 4 | 5 | 6 |
+---+---+---+
| 7 | 8 | 0 |
+---+---+---+
```

```
-------------------------- A* ALGORITMA --------------------------
SOLUSI: (Relative untuk ruang karakter)
1. Geser NAIK
2. Geser NAIK
3. Geser KIRI
4. Geser TURUN
5. Geser TURUN
6. Geser KIRI
7. Geser NAIK
8. Geser KANAN
9. Geser KANAN
10. Geser TURUN
11. Geser KIRI
12. Geser KIRI
13. Geser NAIK
14. Geser KANAN
15. Geser NAIK
16. Geser KIRI
17. Geser TURUN
18. Geser KANAN
19. Geser TURUN
20. Geser KANAN
DETAIL:
 - Panjang Solusi : 20
 - Nodes expanded  : 1531
 - Nodes generated : 2528
 - Waktu           : 0.008 milliseconds
 - Penggunaan Memory    : 80896 bytes

--------------------------------
Process exited after 25.4 seconds with return value 0
Press any key to continue . . .
```