

BỘ MÔN KHOA HỌC MÁY TÍNH

CT332: TRÍ TUỆ NHÂN TẠO

BÀI THỰC HÀNH SỐ 2

TÌM KIẾM HEURISTIC

I. Mục tiêu

- ✓ Biểu diễn bài toán trên không gian trạng thái và áp dụng các luật của heuristic để chọn những nhánh nào có nhiều khả năng nhất đến một giải pháp chấp nhận được – áp dụng heuristic trong bài toán 8 puzzel
- ✓ Ngôn ngữ sử dụng: C.

II. Nội dung

1. Mô tả bài toán

Cho một trạng thái bắt đầu gồm 8 ô số như hình bên dưới. Để bài sẽ cho ta biết trạng thái bắt đầu và kết thúc. Trạng thái bắt đầu và kết thúc là tùy theo mỗi bài toán. Trong bài toán 8 puzzel thì sẽ có $n = 8$, puzzle sẽ chia thành $\sqrt{n + 1}$ hàng, $\sqrt{n + 1}$ cột. Gồm b ô và một ô trống có thể di chuyển được. Di chuyển ô trống để đạt được trạng thái mục tiêu. Biết rằng chúng ta có thể thực hiện các thao tác / hành động sau đây để di chuyển các ô số.

2	8	1
4		3
7	6	5

Initial State

1	2	3
8		4
7	6	5

Goal State

Hành động 1. Di chuyển ô trống sang trái một ô. Nghĩa là đổi giá trị vị trí ô trống với ô bên trái nó. Còn với trường hợp ô trống nằm sát vị trí biên bên trái thì không thực hiện hành động.

Ví dụ: Với vị trí ô trống như hình bên dưới sau khi thực hiện hành động ta sẽ được

<table border="1"><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>8</td><td></td><td>4</td></tr><tr><td>7</td><td>6</td><td>5</td></tr></table>	1	2	3	8		4	7	6	5	<table border="1"><tr><td>1</td><td>2</td><td>3</td></tr><tr><td></td><td>8</td><td>4</td></tr><tr><td>7</td><td>6</td><td>5</td></tr></table>	1	2	3		8	4	7	6	5
1	2	3																	
8		4																	
7	6	5																	
1	2	3																	
	8	4																	
7	6	5																	
Trước khi thực hiện hành động	Sau khi thực hiện hành động																		

Ví dụ: Với vị trí ô trống như hình bên dưới thì việc di chuyển sang trái là không thực hiện được, vì nó đang ở vị trí biên.

	2	3
8	1	4
7	6	5

1	2	3
	8	4
7	6	5

1	2	3
8	7	4
	6	5

Hành động 2. Di chuyển ô trống sang phải một ô. Nghĩa là đổi giá trị vị trí ô trống với ô bên phải nó. Còn với trường hợp ô trống nằm sát vị trí biên bên phải thì không thực hiện hành động.

Ví dụ: Với vị trí ô trống như hình bên dưới sau khi thực hiện hành động ta sẽ được

1	2	3
8		4
7	6	5

1	2	3
8	4	
7	6	5

Trước khi thực hiện hành động

Sau khi thực hiện hành động

Ví dụ: Với vị trí ô trống như hình bên dưới thì việc di chuyển sang phải là không thực hiện được, vì nó đang ở vị trí biên.

1	2	
8	3	4
7	6	5

1	2	3
8	4	
7	6	5

1	2	3
8	5	4
7	6	

Hành động 3. Di chuyển ô trống lên trên một ô. Nghĩa là đổi giá trị vị trí ô trống với ô phía trên nó. Còn với trường hợp ô trống nằm sát vị trí biên trên thì không thực hiện hành động.

1	2	3
8		4
7	6	5

1		3
8	2	4
7	6	5

Trước khi thực hiện hành động

Sau khi thực hiện hành động

Ví dụ: Với vị trí ô trống như hình bên dưới thì việc di chuyển lên trên là không thực hiện được, vì nó đang ở vị trí biên.

	2	3
8	1	4
7	6	5

1		3
8	2	4
7	6	5

1	2	
8	3	4
7	6	5

Hành động 4. Di chuyển ô trống xuống dưới một ô. Nghĩa là đổi giá trị vị trí ô trống với ô phía dưới nó. Còn với trường hợp ô trống nằm sát vị trí biên dưới thì không thực hiện hành động.

Ví dụ: Với vị trí ô trống như hình bên dưới sau khi thực hiện hành động ta sẽ được

<table border="1"> <tr><td>1</td><td>2</td><td>3</td></tr> <tr><td>8</td><td></td><td>4</td></tr> <tr><td>7</td><td>6</td><td>5</td></tr> </table>	1	2	3	8		4	7	6	5	<table border="1"> <tr><td>1</td><td>2</td><td>3</td></tr> <tr><td>8</td><td>6</td><td>4</td></tr> <tr><td>7</td><td></td><td>5</td></tr> </table>	1	2	3	8	6	4	7		5
1	2	3																	
8		4																	
7	6	5																	
1	2	3																	
8	6	4																	
7		5																	
Trước khi thực hiện hành động	Sau khi thực hiện hành động																		

Ví dụ: Với vị trí ô trống như hình bên dưới thì việc di chuyển xuống dưới là không thực hiện được, vì nó đang ở vị trí biên.

1	2	3
8	7	4
	6	5

1	2	3
8	6	4
7		5

1	2	3
8	5	4
7	6	

2. Phân tích giải thuật A*

A* là một giải thuật máy tính được sử dụng rộng rãi để tìm đường đi, di chuyển của đồ thị. Nó sẽ lưu lại các nút, bỏ qua các nút đã truy cập tiết kiệm lượng lớn thời gian. Và một danh sách chứa các nút còn lại để tìm kiếm và chọn ra nút tối ưu nhất. Với bài toán sẽ sử dụng hai danh sách là Open_A_Star và Close_A_Star. Open_A_Star chứa các nút sinh ra từ trạng thái cha, không có trong Open_A_Star và Close_A_Star hoặc đã tồn tại trong Open_A_Star nhưng có g tốt hơn hoặc đã tồn tại trong Close_A_Star nhưng có g tốt hơn. Còn Close_A_Star sẽ chứa đã duyệt qua rồi.

Mỗi nút sẽ có một con trỏ tới cha nó để có thể truy xuất đến cha và in ra kết quả đường đi sau này. Nếu nút đó là nút gốc thì cha là NULL. Ngoài ra nút còn chứa trạng thái, hàm f, g, h. Trong đó h là số ô sai vị trí so với trạng thái mục tiêu (goal), còn g là số lượng nút đã đi qua

để đạt được trạng thái hiện tại, và $f = g + h$. no_function sẽ chứa hành động đã thực hiện để có được nó.

```
//Khai bao cau truc Node de dung cay tim kiem
typedef struct Node{
    State state;
    struct Node* parent; // Nut cha
    int no_function;
    int f; //f=g+h
    int g; //Chi phi tu trang thai bat dau (start) den trang thai hien tai
    int h; //Chi phi tu trang thai hien tai den trang thai dich (goal)
}Node;
```

Với hình bên dưới, thì h sẽ có giá trị bằng 6, còn g sẽ có giá trị bằng 0 do nó là nút bắt đầu.

2	8	1
4	3	
7	6	5

Initial State

1	2	3
8		4
7	6	5

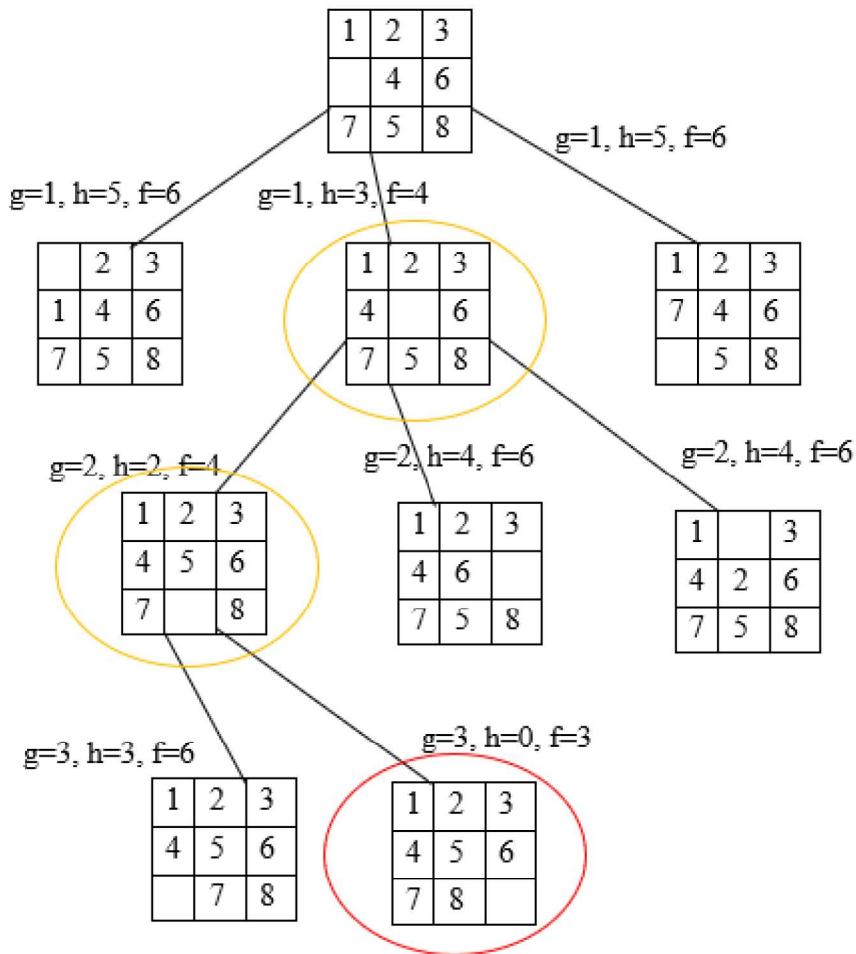
Goal State

Open_A_Star sẽ luôn được sắp xếp theo thứ tự tăng dần của hàm f sau mỗi lần thực hiện xong một nút. Do đó trạng thái tiếp theo được duyệt sẽ lấy ra vị trí đầu tiên trong Open_A_Star.

Close_A_Star thì không cần sắp xếp, sau khi nút nào duyệt xong sẽ bỏ vào vị trí cuối danh sách.

3. Thực hiện giải thuật

$$g=0, h=4, f = g+h = 4$$



Đầu tiên sẽ thực hiện di chuyển ô trống bằng các hành động sang trái, sang phải, lên, xuống từ trạng thái bắt đầu để sinh ra các trạng thái con tìm trạng thái mục tiêu. Sau khi thực hiện xong nó sẽ đưa vào Close_A_Star. Sau đó chọn phần tử đầu tiên trong Open_A_Star (đã sắp xếp tăng dần theo f). Quá trình này cứ tiếp tục cho đến khi tìm được trạng thái mục tiêu.

- Trạng thái bài toán: ma trận biểu diễn bằng mảng 2 chiều 3x3, vị trí ô trống (hàng máy, cột máy).
- Các thao tác/ hành động (operator) để tạo ra trạng thái mới: di chuyển ô trống sang trái, di chuyển ô trống sang phải, di chuyển ô trống lên trên, di chuyển ô trống xuống dưới.
- Trạng thái đầu: Giá trị của mỗi vị trí trong ma trận. Vị trí hàng và cột của ô trống
- Trạng thái cuối: Vị trí các ô trống vô trạng thái mục tiêu (GOAL: mục tiêu).
- Hàm tính heuristic
- Hàm A* thực hiện việc duyệt.

4. Cài đặt

Các hàm khác sinh viên tự viết như file thực hành trước

4.1 Cài đặt giải thuật A*

```
//Thuật toán A Star
//Ham f = g + h
Node* A_Star(State state, State goal){
    List Open_A_Star;
    List Close_A_Star;
    makeNull_List(&Open_A_Star);
    makeNull_List(&Close_A_Star);
    Node* root = (Node*)malloc(sizeof(Node));
    root->state = state;
    root->parent = NULL;
    root->no_function = 0;
    root->g = 0;
    root->h = heuristic1(root->state, goal);
    root->f = root->g + root->h;
    push_List(root, Open_A_Star.size+1, &Open_A_Star);
    while(!empty_List(Open_A_Star)){
        Node* node = element_at(1, Open_A_Star);
        delete_List(1, &Open_A_Star);
        push_List(node, Close_A_Star.size+1, &Close_A_Star);
        if(goalcheck(node->state, goal))
            return node;
        int opt;
        for(opt=1; opt<=MAX_OPERATOR; opt++){
            State newstate;
            if(callOperators(node->state, &newstate, opt)){
                Node* newNode = (Node*)malloc(sizeof(Node));
                newNode->state = newstate;
                newNode->parent = node;
                newNode->no_function = opt;
                newNode->g = node->g + 1;
                newNode->h = heuristic1(newstate, goal);
                newNode->f = newNode->g + newNode->h;
                //Kiem tra trang thai moi sinh ra co thuoc Open_A_Star/ Close_A_Star
                int pos_Open, pos_Close;
                Node* nodeFoundOpen = find_state(newstate, Open_A_Star, &pos_Open);
                Node* nodeFoundClose = find_state(newstate, Close_A_Star, &pos_Close);
                if(nodeFoundOpen == NULL && nodeFoundClose == NULL){
                    push_List(newNode, Open_A_Star.size+1, &Open_A_Star);
                }
                else if(nodeFoundOpen != NULL && nodeFoundOpen->g > newNode->g){
                    delete_List(pos_Open, &Open_A_Star);
                    push_List(newNode, pos_Open, &Open_A_Star);
                }
                else if(nodeFoundClose != NULL && nodeFoundClose->g > newNode->g){
                    delete_List(pos_Close, &Close_A_Star);
                    push_List(newNode, Open_A_Star.size+1, &Open_A_Star);
                }
            }
            sort_List(&Open_A_Star);
        }
        return NULL;
    }
}
```

4.2 Bài tập 1: Biểu diễn trạng thái và các phép toán (hành động) của bài toán, gọi các hành động sử dụng giải thuật A* và in ra màn hình.

Các bước thực hiện:

- Khai báo cấu trúc trạng thái: State
- Cài đặt hàm compareStates
- Cài đặt hàm find _State
- Cài đặt các hàm hành động chuyển trạng thái: di chuyển ô trống sang trái, di chuyển ô trống sang phải, di chuyển ô trống lên, di chuyển ô trống xuống.
- Cài đặt hàm callOperators để gọi các hành động.
- Cài đặt giải thuật A*
- Cài đặt hàm heuristic
- Cài đặt cấu trúc danh sách
- Cài đặt hàm in trạng thái ra màn hình.
- Cài đặt hàm main()

```
int main(){  
    State state;  
    state.emptyRow = 1;  
    state.emptyCol = 1;  
    state.eightPuzzel[0][0] = 1;  
    state.eightPuzzel[0][1] = 2;  
    state.eightPuzzel[0][2] = 3;  
    state.eightPuzzel[1][0] = 8;  
    state.eightPuzzel[1][1] = 0;  
    state.eightPuzzel[1][2] = 4;  
    state.eightPuzzel[2][0] = 7;  
    state.eightPuzzel[2][1] = 6;  
    state.eightPuzzel[2][2] = 5;  
    State goal;  
    goal.emptyRow = 1;  
    goal.emptyCol = 0;  
    goal.eightPuzzel[0][0] = 2;  
    goal.eightPuzzel[0][1] = 8;  
    goal.eightPuzzel[0][2] = 1;  
    goal.eightPuzzel[1][0] = 0;  
    goal.eightPuzzel[1][1] = 4;  
    goal.eightPuzzel[1][2] = 3;  
    goal.eightPuzzel[2][0] = 7;  
    goal.eightPuzzel[2][1] = 6;  
    goal.eightPuzzel[2][2] = 5;  
    Node* p = A_Star(state, goal);  
    print_WaysToGetGoal(p);  
    return 0;  
}
```

Kết quả chương trình:

Action 0: First State

1	2	3
8	0	4
7	6	5

Action 1: Move cell EMPTY to UP

1	0	3
8	2	4
7	6	5

Action 2: Move cell EMPTY to LEFT

0	1	3
8	2	4
7	6	5

Action 3: Move cell EMPTY to DOWN

8	1	3
0	2	4
7	6	5

Action 4: Move cell EMPTY to RIGHT

8	1	3
2	0	4
7	6	5

Action 5: Move cell EMPTY to RIGHT

8	1	3
2	4	0
7	6	5

Action 6: Move cell EMPTY to UP

8	1	0
2	4	3
7	6	5

Action 7: Move cell EMPTY to LEFT

8	0	1
2	4	3
7	6	5

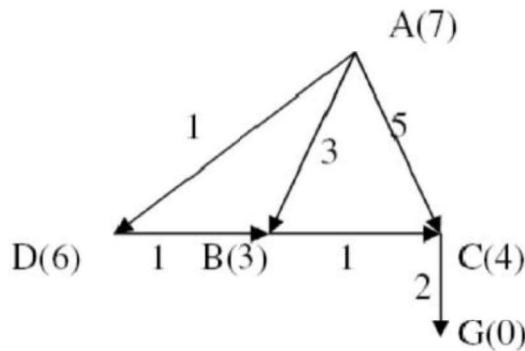
Action 8: Move cell EMPTY to LEFT

0	8	1
2	4	3
7	6	5

Action 9: Move cell EMPTY to DOWN

2	8	1
0	4	3
7	6	5

4.3 Bài tập 2: Sử dụng giải thuật A* để tìm đường đi từ A đến G và in ra màn hình



4.3.1 Xây dựng đỉnh của đồ thị

- Mỗi đỉnh sẽ lưu giá trị heuristic và lân cận của nó
- Giá trị của lân cận tương ứng với g

```
// Xay dung dinh cua do thi
typedef struct{
    int neighbor[MAX_VERTICES];
    int h;
}Vertices;
```

4.3.2 Xây dựng đồ thị

- Gồm có số lượng đỉnh và giá trị của mỗi đỉnh

```
// Xay dung do thi
typedef struct{
    int num_vertices;
    Vertices v[MAX_VERTICES];
}Graph;
```

4.3.3 Khởi tạo đồ thị

```
// Khoi tao do thi
void initGraph(int num_vertices, Graph *G) {
    G->num_vertices = num_vertices;
    int i, j;
    for (i = 0; i < num_vertices; i++)
        for (j = 0; j < num_vertices; j++)
            G->v[i].neighbor[j] = 0;
    G->v[i].h = 0;
}
```

4.3.4 Khai báo cấu trúc trạng thái

- Mỗi trạng thái lưu đỉnh hiện tại
- VD: với 0: đỉnh A, 1: đỉnh B,... Tuỳ vào mỗi bài toán ta có thứ tự đỉnh

```
const char name[] = {'A', 'B', 'C', 'D', 'G'};
```

```
//Khai bao cau truc trang thai
typedef struct{
    int vertex;
}State;
```

4.3.5 In trạng thái

```
//In trang thai
void printState(State state){
    printf("%c", name[state.vertex]);
}
```

4.3.6 So sánh trạng thái 1 với trạng thái 2

```
//So sanh trang thai statel co giong voi trang thai state2
int compareStates(State state1, State state2){
    return state1.vertex == state2.vertex;
}
```

4.3.7 Khai báo node

```
//Khai bao cau truc Node de dung cay tim kiem
typedef struct Node{
    State state;
    struct Node* parent; // Nut cha
    int f; //f=g+h
    int g; //Chi phi tu trang thai bat dau (start) den trang thai hien tai
    int h; //Chi phi tu trang thai hien tai den trang thai dich (goal)
}Node;
```

4.3.8 Khai báo cấu trúc list để lưu trữ KGTK

- Sinh viên tự viết cấu trúc danh sách và các hàm cần thiết của danh sách

4.3.9 Tìm kiếm trạng thái

```
//Tim trang thai state co thuoc Open hoac Close hay ko?
//Luu vi tri tim duoc vao bien *position
Node* find_State(State state, List list, int *position){
    int i;
    for(i=1; i<=list.size; i++){
        if(compareStates(element_at(i,list)->state, state)){
            *position = i;
            return element_at(i,list);
        }
    }
    return NULL;
}
```

4.3.10 Kiểm tra goal

```
//Ham kiem tra trang thai muc tieu
int goalcheck(State state, State goal){
    return compareStates(state, goal);
}
```

4.3.11 Xây dựng giải thuật A* cho bài toán tìm đường đi từ A đến G

```
//Thuat toan A Star
//Ham f = g + h
Node* A_Star(Graph G, State state, State goal){
    List Open_A_Star;
    List Close_A_Star;
    makeNull_List(&Open_A_Star);
    makeNull_List(&Close_A_Star);
    Node* root = (Node*)malloc(sizeof(Node));
    root->state = state;
    root->parent = NULL;
    root->g = 0; // Dinh dau tien g = 0;
    root->h = G.v[state.vertex].h; // Lay h cua dinh state.vertex
    root->f = root->g + root->h;
    push_List(root, Open_A_Star.size+1, &Open_A_Star);
    while(!empty_List(Open_A_Star)){
        Node* node = element_at(1,Open_A_Star);
        delete_List(1, &Open_A_Star);
        push_List(node, Close_A_Star.size+1, &Close_A_Star);
        if(goalcheck(node->state, goal))
            return node;
        int opt;
        for(opt=0; opt<G.num_vertices; opt++){ // Duyet qua tung dinh
            State newstate;
            if(G.v[node->state.vertex].neighbor[opt] > 0){ // Kiem tra > 0 la co duong di
                Node* newNode = (Node*)malloc(sizeof(Node));
                newNode->state.vertex = opt;
                newNode->parent = node;
                newNode->g = node->g + G.v[node->state.vertex].neighbor[opt]; // Lay g tu dinh state.vertex toi opt
                newNode->h = G.v[opt].h; // Lay gia tri h cua dinh state.vertex
                newNode->f = newNode->g + newNode->h;
                //Kiem tra trang thai moi sinh ra co thuoc Open_A_Star/ Close_A_Star
                int pos_Open, pos_Close;
                Node* nodeFoundOpen = find_State(newNode->state, Open_A_Star, &pos_Open);
                Node* nodeFoundClose = find_State(newNode->state, Close_A_Star, &pos_Close);
                if(nodeFoundOpen == NULL && nodeFoundClose == NULL){
                    push_List(newNode, Open_A_Star.size+1, &Open_A_Star);
                }
                else if(nodeFoundOpen != NULL && nodeFoundOpen->g > newNode->g){
                    delete_List(pos_Open, &Open_A_Star);
                    push_List(newNode, pos_Open, &Open_A_Star);
                }
                else if(nodeFoundClose != NULL && nodeFoundClose->g > newNode->g){
                    delete_List(pos_Close, &Close_A_Star);
                    push_List(newNode, Open_A_Star.size+1, &Open_A_Star);
                }
            }
        }
        sort_List(&Open_A_Star);
    }
    return NULL;
}
```

4.3.12 In kết quả đường đi

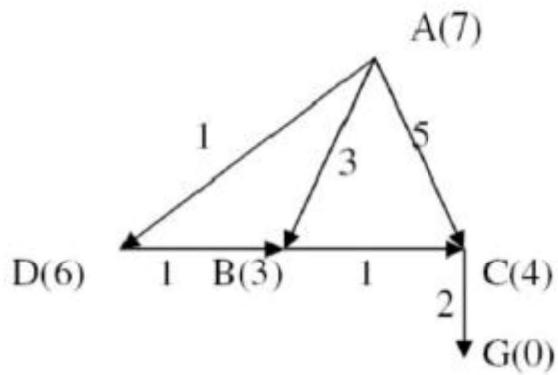
```
//Ham in ket qua cua thuat toan BFS
void print_WaysToGetGoal(Node* node) {
    List listPrint;
    makeNull_List(&listPrint);
    //Duyet nguoc ve nut parent
    while(node->parent != NULL){
        push_List(node, listPrint.size+1, &listPrint);
        node = node->parent;
    }
    push_List(node, listPrint.size+1, &listPrint);
    //In ra thu tu hanh dong di chuyen o trong
    int no_action = 0, i;
    for(i=listPrint.size; i>0; i--){
        printState(element_at(i, listPrint)->state);
        if(i!=1){
            printf("->");
        }
        no_action++;
    }
}
```

4.3.13 Hàm main

```
int main(){
    int i, j;
    Graph graph;
    freopen("test.txt","r",stdin);
    initGraph(MAX_VERTICES,&graph);
    for( i = 0 ; i < MAX_VERTICES; i++){
        int x;
        scanf("%d", &x);
        graph.v[i].h = x;
        for(j = 0; j < MAX_VERTICES; j++){
            scanf("%d", &x);
            graph.v[i].neighbor[j]= x;
        }
    }
    State A, G;
    A.vertex = 0;
    G.vertex = 4;
    Node* result = A_Star(graph, A, G);
    print_WaysToGetGoal(result);

    return 0;
}
```

4.3.14 File dữ liệu của tam giác



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

- Hàng 0 tương ứng với đỉnh A:
 - + Giá trị cột đầu tiên là h = 7
 - + Cột còn lại là giá trị g
 - + VD: cột 2 là giá trị từ A đến A = 0, cột 3 là giá trị từ A đến B = 3, A đến C = 5, A đến D = 1, A đến G = 0. Với giá 0 thì coi như không có đường đi.