CT332: TRÍ TUỆ NHÂN TẠO

BÀI THỰC HÀNH SỐ 2

TÌM KIẾM HEURISTIC

I. Mục tiêu

Trên cơ sở trò chơi 8 ô số (8 puzzle) và không gian trạng thái đã được học, cài đặt bài toán và sử dụng hàm đánh giá Heuristic để chọn triển khai nút có tiềm năng dẫn về mục tiêu nhằm làm giảm chi phí cho việc tìm ra giải pháp (giảm số nút duyệt và phát sinh).

II. Nội dung

1. Tìm kiếm Heuristic

Hàm đánh giá Heuristic tại trạng thái n là f(n):

$$f(n) = g(n) + h(n)$$

g(n) = khoảng cách thực sự từ trạng thái n đến trạng thái bắt đầu

h(n) = ước lượng heuristic cho khoảng cách từ trạng thái n đến trạng thái đích

Chú ý: Đối với bài toán 8 puzzle

- g(n) là độ sâu của trạng thái n
- h(n) có thể đánh giá dựa vào số vị trí sai khác của trạng thái n so với goal hoặc sử dụng khoảng cách Manhattan

2. Cài đặt

- Kiểu dữ liệu node sẽ có 5 thuộc tính quan trọng là: g, h, f, state và parent lưu trữ giá trị của các hàm g(n), h(n), f(n), trạng thái hiện tại và cha của nút đang xét.
- Hàm *A_star* minh họa giải thuật A*, thực hiện tìm kiếm bằng cách chọn triển khai các nút với giá trị f nhỏ nhất.

(Xem code đính kèm)

3. Câu hỏi

a. Cài đặt các phép toán còn lại (down, left, right) cho bài toán 8 ô số.

b. Viết hàm main để chạy code đính kèm, nhận xét kết quả đạt được

c. Kiểm tra lại code ở câu b với các trường hợp phần tử đã xuất hiện trong frontiers hoặc explored, sử dụng các hàm find_node và find_state đã cài đặt.

d. Cài đặt hàm f(n) với $h_1(n)$ là số vị trí sai khác của trạng thái n so với goal. Xác định các giá trị:

- tổng số nút đã duyệt
- tổng số nút phát sinh
- độ sâu của goal

e. Tương tự câu c, cài đặt $h_2(n)$ là khoảng cách Manhattan

```cpp
#include <iostream>
#include <set>
#include <stack>
#include <vector>
#include <cmath>

using namespace std;
const int BOARD_ROWS = 3;
const int BOARD_COLS = 3;
const int EMPTY_TILE = 0;
const int MAX_OP = 4;

struct State{
    int num[BOARD_ROWS][BOARD_COLS];
    int empty_row; //luu chi muc dong cua o trong
    int empty_col; //luu chi muc cot cua o trong
};
struct Node{
    State state;
    Node* parent;
    int g; //Luu gia tri duong di cua nut goc den nut hien tai
    int h; //Luu ket qua uoc luong cua ham heuristic cua nut hien tai den trang thai dich
    int f; // = g + h

};

//dinh nghia phep toan so sanh trong multiset
struct node_cmp{
    bool operator() (Node* a, Node* b){
        return a->f < b->f;
    }
};
//kiem tra 2 trang thai co giong nhau khong?
bool sameState (State s1, State s2){
    if (s1.empty_col != s2.empty_col || s1.empty_row != s2.empty_row) {
        return false;
    }
    for (int row =0;row<BOARD_ROWS;row++){
        for (int col=0;col<BOARD_COLS;col++){
            if (s1.num[row][col] !=s2.num[row][col]){
                return false;
            }
        }
    }
    return true;
}

```

```
55  //phep toan di chuyen o trong len tren
56  bool up (State s, State &out) {
57      int er = s.empty_row, ec = s.empty_col;
58      if (er>0){
59          out = s;
60          out.empty_col = ec;
61          out.empty_row = er-1;
62
63          out.num[er][ec] = s.num[er-1][ec];
64          out.num[er-1][ec] = EMPTY_TILE;
65
66          return true;
67
68      }
69      return false;
70  }
```

Sinh viên tự cài đặt 3 phép toán còn lại.

```
126 //ham sinh cac phep toan
127 bool call_operator (State s, State& out, int op_no){
128     switch (op_no) {
129     case 1:
130         return up(s,out);
131     case 2:
132         return down(s,out);
133     case 3:
134         return left(s,out);
135     case 4:
136         return right(s,out);
137     default:
138         return false;
139     }
140 }
```

```
143 //in trang thai s
144 void print_state(State s) {
145     for (int i=0;i<BOARD_ROWS;i++){
146         for (int j=0;j<BOARD_COLS;j++) {
147             cout << s.num[i][j]) << " ";
148         }
149         cout << "\n" ;
150
151     }
152 }
```

```
158 //kiem tra trang thai co phai la goal?
159 bool is_goal(State s, State goal) {
160     return sameState(s,goal);
161 }
```

```
/*
 * ham H1 dem so vi tri sai khac
 */
int h1(State s, State s2){
    int count = 0;
    for (int row=0;row<BOARD_ROWS;row++) {
        for (int col=0;col<BOARD_COLS;col++) {
            if (s.num[row][col] != s2.num[row][col]) {
                count++;

            }
        }
    }
    return count;

}
```

Kiểm tra phần tử đã có trong frontier chưa?

```
//search phan tu da co trong frontier?
Node* find_node(State s, multiset<Node *, node_cmp> list) {
    for (Node* n: list){
        if (sameState(s, n->state)){
            return n;
        }
    }
    return NULL;

}
```

Kiểm tra phần tử đã có trong explored chưa?

```
//search phan tu da co trong explore?
bool find_state(State s, vector <State> *explored){
    for (State c1: *explored) {
        if (sameState(s,c1))
            return true;
        }
    }
```

Khai báo trạng thái đầu và cuối cho bài toán, cài đặt hàm getState()

```
//nhap trang thai cho bai toan
State* getState(){
    State *s = new State();
    for (int row=0;row<BOARD_ROWS;row++) {
        for (int col=0;col<BOARD_COLS;col++){
            cin >> s->num[row][col];
            if (s->num[row][col]==0){
                s->empty_row  = row;
                s->empty_col = col;
            }
        }

    }
    return s;
}
```

Giải thuật A*:

```
274  //giai thuat A*
275  Node* A_star(State init_state, State goal_state, vector<State> *explored){
276      //initial
277      Node* root = new Node();
278      root->state = init_state;
279      root->parent = NULL;
280      root->g = 0;
281      root->h = h1(init_state,goal_state);
282      root->f = root->g + root->h;
283      //frontiers la tap Open duoc sap thu tu
284      multiset<Node*, node_cmp> frontiers ;
285      frontiers.insert(root);
286      while (!frontiers.empty()){
287          Node* node = *frontiers.begin();
288          frontiers.erase(frontiers.begin());
289          explored->push_back(node->state);
290          if (sameState(node->state, goal_state)){
291              return node;
292          }
293
294          //generate states
295          for (int op=1;op<=4;op++){
296              State new_state;
297              if (call_operator(node->state,new_state,op)){
298                  if (find_state(new_state,explored)){
299                      continue;
300                  }
301                  Node* n= find_node(new_state,frontiers);
302                  if (n==NULL){
303                      n = new Node();
304                      n->parent=node;
305                      n->state = new_state;
306                      n->h = h1(new_state,goal_state);
307                      print_state(new_state);
308                      n->g=node->g+1;
309                      n->f=n->g+n->h;
310                      cout<< "==== Gia tri g: "<<n->g << "==== Gia tri f: "<<n->f<<endl;
311                      frontiers.insert(n);
312                  }
313                  else{//neu nut con moi tim da thuoc duong bien
314                      //kiem tra gia tri f cuanut con co nho hon cacnut da nam trong frontier ko?
315                      //neu nho hon thi cap nhat lai, neu lonhonthi ko can lam gica
316                      n->g=node->g+1;
317                      n->f=n->g+n->h;
318
319                  }
320              }
321          }
322      }
323      return NULL;
324  }
```

```
341 void print_path(Node*r){
342     int i =0;
343     stack<State> q;
344     cout << "\nDuong di loi giai\n";
345     while (r->parent !=NULL) {
346         q.push(r->state);
347         r=r->parent;
348        // i++
349     }
350     q.push(r->state); //them nut goc vao stack
351     while (!q.empty()){
352         cout << "\Trang thai thu " << i++ << endl;
353         print_state(q.top());
354         cout << endl;
355         q.pop();
356     }
357 }
```

Lưu ý: Trường hợp lỗi biên dịch, vào Tools->Compiler Option->Programs, thêm `-std=c++11`