CS-UY 4613 Project 1

Charles pan, cp3723

Nick Li, ql2015

Instructions

Uncompressed the .zip file. The project has been compiled with Visual Studio 2022 and C++ 14. To run the program, run the 4613_Project1.exe, the output file will be in the output folder, named from output1.txt to output8.txt .

Source Code

main.cpp

```
#include<iostream>
#include<fstream>
#include<format>
#include<string>
#include<string_view>
#include<vector>
#include<direct.h>
#include "Puzzle.h"

void readInput() {
  int temp;
  if (_mkdir("output") == -1) {
    std::cerr << "Error\n";
  }
  else {
    std::cout << "Folder created\n";
  }
  for (int i = 1; i <= 8; i++) {
    std::cout << "\n[ Input" << std::to_string(i) << " ]\n";
    float weight = 1.0f;
    std::vector<int> init;
    std::vector<int> goal;
    std::string res;

    // read file with name input(i).txt
    std::ifstream file("input/input" + std::to_string(i) + ".txt");
    // store weight
    file >> weight;

    // store initial state data
    for (int i = 0; i < 16; i++) {
```

```
      file >> temp;
      init.push_back(temp);
    }

    // store goal state data
    for (int i = 0; i < 16; i++) {
      file >> temp;
      goal.push_back(temp);
    }

    file.close();

    // create the output file


    std::ofstream ofs;
    ofs.open("output/output" + std::to_string(i) + ".txt");

    // declare Puzzle object and solve it
    Puzzle p(weight, init, goal);

    p.solve();

    // print the output
    ofs << p;
    std::cout << p;
    std::cout << "************************************" << std::endl;

    ofs.close();
  }
}

int main() {
  readInput();
}
```

Puzzle.h

```
#include<vector>
#include<string>
#include<queue>

#ifndef PUZZLE_H
#define PUZZLE_H


struct Chess {
  int value;
  int x;
  int y;
  // Chess constructor
  Chess(int v, int x, int y) : value(v), x(x), y(y) {};
};

struct State {
  int heuristicValue;       // h(n)
  int stepTaken;          // g(n)
  float fValue;         // f(n) = g(n) + W * h(n)
```

```
  std::vector<Chess> board;   // vector of Chess with value and location
  std::string path;       // a path of string storing movement {L R U D}
  std::vector<float> fValuePath;  // a vector of float storing f(n) of each nodes in the path

  State() : heuristicValue(16), stepTaken(0), fValue(0.0f), path("") {};  // default constructor
  void calculateHeuristic(const State&);                  // calculate h(n)
  void calculateFValue(float);                      // calculate f(n)
  inline bool operator==(const State&);

  // operator< overload for prioritize state
  bool operator<(const State& rhs) const
  {
    return fValue > rhs.fValue;
  }
};


class Puzzle {
  friend std::ostream& operator<<(std::ostream& os, const Puzzle& dt);
public:
  Puzzle(float aWeight, std::vector<int>& aInit, std::vector<int>& aGoal);
  Puzzle(const Puzzle&);
  bool isVisited(State&);           // check if a state(node) is visited
  void findNeighbors(std::vector<State>&);  // find the neighbors of the current state
  void solve();               // method for solving the 15 puzzle

  // helper function
  void printState();
  void printQueue();

private:
  std::priority_queue<State> queue;     // priority queue for unvisited states
  std::vector<State> visited;         // visited states
  State init;                 // initial state
  State cur;                  // current state
  State goal;                 // goal state
  float weight;               // weight
  int totalNodes;               // total nodes generated

};


#endif // !PUZZLE_H
```

Puzzle.cpp

```
#include<iostream>

#include "Puzzle.h"

using namespace std;


// calculate h(n)
void State::calculateHeuristic(const State& g) {
  int res = 0;
  for (const Chess& i : board) {
    for (const Chess& j : g.board) {
```

```
      if (i.value == j.value && i.value != 0) {
        res += max(abs(i.x - j.x), abs(i.y - j.y));
      }
    }
  }
  heuristicValue = res;
}

// calculate f(n)
void State::calculateFValue(float weight) {
  fValue = (float)stepTaken + weight * (float)heuristicValue;
}

// check if two board are the same
bool State::operator==(const State& rhs) {
  for (int i = 0; i < 16; i++) {
    if (board[i].value != rhs.board[i].value) {
      return false;
    }
  }
  return true;
}

// output the result in format
ostream& operator<<(ostream& os, const Puzzle& p) {
  // init
  for (int y = 0; y < 4; y++) {
    for (int x = 0; x < 4; x++) {
      os << p.init.board[x + 4 * y].value << " ";
    }
    os << endl;
  }
  os << endl;
  // goal
  for (int y = 0; y < 4; y++) {
    for (int x = 0; x < 4; x++) {
      os << p.goal.board[x + 4 * y].value << " ";
    }
    os << endl;
  }
  os << endl;
  // weight
  os << p.weight << endl;
  // shallowest depth
  os << p.cur.path.length() / 2 << endl;
  // total nodes
  os << p.totalNodes << endl;
  // solution
  os << p.cur.path << endl;
  // f(n) value path
  for (float i : p.cur.fValuePath) os << i << " ";
  os << endl;
  return os;
}

// constructor store weight, init, and goal state from input
Puzzle::Puzzle(float aWeight, vector<int>& aInit, vector<int>& aGoal):
  weight(aWeight), totalNodes(0)
{
  cur = State();
  goal = State();
  // store init, current, and goal state
```

```
  for (int y = 0; y < 4; y++) {
    for (int x = 0; x < 4; x++) {
      init.board.push_back(Chess(aInit[x + y * 4], x, y));
      cur.board.push_back(Chess(aInit[x + y * 4], x, y));
      goal.board.push_back(Chess(aGoal[x + y * 4], x, y));
    }
  }
  // calculate h(n), f(n) for current(initial) state
  cur.calculateHeuristic(goal);
  cur.calculateFValue(weight);
  cur.fValuePath.push_back(cur.fValue);
  goal.calculateHeuristic(goal);

  // push initial state into priority queue
  queue.push(cur);
  totalNodes++;
}

// copy constructor
Puzzle::Puzzle(const Puzzle& rhs) {
  queue = rhs.queue;
  visited = rhs.visited;
  cur = rhs.cur;
  goal = rhs.goal;
  weight = rhs.weight;
}

// check if a state is visited
bool Puzzle::isVisited(State& p1) {
  for (State& p2 : visited) {
    if (p1 == p2) {
      return true;
    }
  }
  return false;
}

// find neighbor states
void Puzzle::findNeighbors(vector<State>& nbs) {
  // flag represents if the current state can move to ( ) state
  bool canL = true, canR = true, canU = true, canD = true;
  // postion of empty block
  int x0 = 0, y0 = 0;

  // calculate g(n) for neighbors
  int g = cur.stepTaken + 1;

  // find empty
  for (Chess& c : cur.board) {
    if (c.value == 0) {
      x0 = c.x;
      y0 = c.y;
      if (x0 == 0) canL = false;
      if (x0 == 3) canR = false;
      if (y0 == 0) canU = false;
      if (y0 == 3) canD = false;
      break;
    }
  }
  // left
  if (canL) {
    State LState(cur);
```

```
    // swap empty block with the block in the next position
    LState.board[x0 + y0 * 4].x = x0 - 1;
    LState.board[x0 + y0 * 4 - 1].x = x0;
    swap(LState.board[x0 + y0 * 4], LState.board[x0 + y0 * 4 - 1]);
    // calculate g(n), h(n), f(n) for the next state
    // append action path and vector of f(n)
    LState.stepTaken = g;
    LState.calculateHeuristic(goal);
    LState.calculateFValue(weight);
    LState.path += "L ";
    LState.fValuePath.push_back(LState.fValue);
    // append new state into the vector
    nbs.push_back(LState);
  }
  // right
  if (canR) {
    State RState(cur);
    // swap empty block with the block in the next position
    RState.board[x0 + y0 * 4].x = x0 + 1;
    RState.board[x0 + y0 * 4 + 1].x = x0;
    swap(RState.board[x0 + y0 * 4], RState.board[x0 + y0 * 4 + 1]);
    // calculate g(n), h(n), f(n) for the next state
    // append action path and vector of f(n)
    RState.stepTaken = g;
    RState.calculateHeuristic(goal);
    RState.calculateFValue(weight);
    RState.path += "R ";
    RState.fValuePath.push_back(RState.fValue);
    // append new state into the vector
    nbs.push_back(RState);
  }
  // up
  if (canU) {
    State UState(cur);
    // swap empty block with the block in the next position
    UState.board[x0 + y0 * 4].y = y0 - 1;
    UState.board[x0 + y0 * 4 - 4].y = y0;
    swap(UState.board[x0 + y0 * 4], UState.board[x0 + y0 * 4 - 4]);
    // calculate g(n), h(n), f(n) for the next state
    // append action path and vector of f(n)
    UState.stepTaken = g;
    UState.calculateHeuristic(goal);
    UState.calculateFValue(weight);
    UState.path += "U ";
    UState.fValuePath.push_back(UState.fValue);
    // append new state into the vector
    nbs.push_back(UState);
  }
  // down
  if (canD) {
    State DState(cur);
    // swap empty block with the block in the next position
    DState.board[x0 + y0 * 4].y = y0 + 1;
    DState.board[x0 + y0 * 4 + 4].y = y0;
    swap(DState.board[x0 + y0 * 4], DState.board[x0 + y0 * 4 + 4]);
    // calculate g(n), h(n), f(n) for the next state
    // append action path and vector of f(n)
    DState.stepTaken = g;
    DState.calculateHeuristic(goal);
    DState.calculateFValue(weight);
    DState.path += "D ";
    DState.fValuePath.push_back(DState.fValue);
```

```
    // append new state into the vector
    nbs.push_back(DState);
  }
}


// solve puzzle and return the string a path
void Puzzle::solve() {
  //is-goal
  while (!queue.empty()) {
    // count nodes number
    // extract min
    cur = queue.top();
    visited.push_back(cur);
    queue.pop();
    // if goal is met, return
    if (cur.heuristicValue == 0) return;
    // add states into neighbors
    std::vector<State> neighbors;
    findNeighbors(neighbors);

    // add unvisited neighbors into priority queue
    for (State& n : neighbors) {
      if (!isVisited(n)) {
        totalNodes++;
        queue.push(n);
      }
    }
  }
}


// helper function: print chessboard, print priority queue(fvalue)
void Puzzle::printState() {
  cout << endl;
  for (int y = 0; y < 4; y++) {
    for (int x = 0; x < 4; x++) {

      cout << cur.board[x + 4 * y].value << " ";
    }
    cout << endl;
  }
  cout << endl;
}

void Puzzle::printQueue() {
  priority_queue<State> g = queue;
  while (!g.empty()) {
    cout << " " << g.top().fValue;
    g.pop();
  }
  cout << '\n';
}
```

Output:

[Input1]
1 5 3 13
8 0 14 4
15 10 7 2
11 6 9 12

1 5 3 13
8 10 14 4
0 15 9 2
11 7 6 12

1
6
22
D R D L U L
5 5 6 6 6 6 6

[Input2]
2 13 7 4
12 3 0 1
9 15 5 14
6 10 11 8

13 3 7 4
2 1 0 14
12 9 5 8
6 15 10 11

1
12
27
R D D L L U L U U R D R
12 12 12 12 12 12 12 12 12 12 12 12 12

[Input3]
13 12 9 11
10 1 8 2
0 3 15 6
14 4 7 5

10 13 12 11
8 1 9 2
3 4 15 5
14 0 6 7

1
16
218
R U R U L L D R D R R D L U L D
11 11 13 13 13 13 13 13 13 13 15 16 16 16 16 16 16

[Input4]
13 12 9 11
10 1 8 2
0 3 15 6
14 4 7 5

10 13 12 11
8 1 9 2
3 4 15 5
14 0 6 7

1.2
16
124
R U R U L L D R D R R D L U L D
13.2 13 15.2 15 14.8 14.6 14.4 14.2 14 13.8 16 17 16.8 16.6 16.4 16.2 16

[Input5]
13 12 9 11
10 1 8 2
0 3 15 6
14 4 7 5

10 13 12 11
8 1 9 2
3 4 15 5
14 0 6 7

1.4
16
77
R U R U L L D R D R R D L U L D
15.4 15 17.4 17 16.6 16.2 15.8 15.4 15 14.6 17 18 17.6 17.2 16.8 16.4 16

[Input6]
7 1 4 12
5 3 9 10
15 14 8 6
13 11 0 2

4 9 10 12
1 7 0 6
15 5 3 2
13 11 14 8

1
20
971
U L U L U R R D R D D L U L U L U R R D
12 13 14 15 16 17 18 18 19 20 20 20 20 20 20 20 20 20 20 20

[Input7]
7 1 4 12
5 3 9 10
15 14 8 6
13 11 0 2

4 9 10 12
1 7 0 6
15 5 3 2
13 11 14 8

1.2
20
637
U L U L U R R D R D D L U L U L U R R D
14.4 15.4 16.4 17.4 18.4 19.4 20.4 20.2 21.2 22.2 22 21.8 21.6 21.4 21.2 21 20.8 20.6 20.4 20.2 20

[Input8]
7 1 4 12
5 3 9 10
15 14 8 6
13 11 0 2

4 9 10 12
1 7 0 6
15 5 3 2
13 11 14 8

1.4
20
565
U L U L U R R D R D D L U L U L U R R D

16.8 17.8 18.8 19.8 20.8 21.8 22.8 22.4 23.4 24.4 24 23.6 23.2 22.8 22.4 22 21.6 21.2 20.8 20.4 20