Source Code

```
//  CS 4613
//  Project 1
//
//  Micheal Zhang

#include <iostream>
#include <fstream>
#include <algorithm>
#include <queue>

using namespace std;

const int x = 0, y = 1, z = 2;

int man_distance(int x1, int y1, int z1, int x2, int y2, int z2) {
    return abs(x1 - x2) + abs(y1 - y2) + abs(z1 - z2);
}

class Node {
private:
    Node* prev; /* pointer to parent node */
    string path;
    int g /* steps taken */, f /* estimate total cost */;
    int blank[3] /* blank position */, state[3][3][3];
public:
    Node(int input[3][3][3]) { /* constructor for creating new nodes */
        g = 0;
        f = -1;  /* not calculated yet */
        prev = nullptr;
        for (int i = 0; i < 3; i++) for (int j = 0; j < 3; j++) for (int k = 0; k < 3; k++) {
            state[i][j][k] = input[i][j][k];
            if (input[i][j][k] == 0) {
                blank[x] = k;
                blank[y] = j;
                blank[z] = i;
            }
        }
    }
    Node(const Node& other) { /* copy constructor */
        g = other.g;
        f = other.f;
        prev = other.prev;
        path = other.path;
```

```
        for (int i = 0; i < 3; i++)
            blank[i] = other.blank[i];
        for (int i = 0; i < 3; i++) for (int j = 0; j < 3; j++) for (int k = 0; k < 3; k++)
            state[i][j][k] = other.state[i][j][k];
    }
    bool operator==(const Node& other) { /* is a repeated state? */
        for (int i = 0; i < 3; i++) for (int j = 0; j < 3; j++) for (int k = 0; k < 3; k++)
            if (state[i][j][k] != other.state[i][j][k])
                return false;
        return true;
    }
    bool operator!=(const Node& other) {
        return !(*this == other);
    }
    bool operator>(const Node& other) { /* comparison on f value */
        return (f > other.f);
    }
    bool operator<(const Node& other) {
        return (f < other.f);
    }
    friend ostream& operator<<(ostream& os, const Node& node) { /* output state */
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                for (int k = 0; k < 3; k++) {
                    os << node.state[i][j][k] << " ";
                }
                os << endl;
            }
            os << endl;
        }
        return os;
    }
    int heurstic(const Node& goal) { /* calculate heurstic and update f */
        int sum = 0;
        for (int i1 = 0; i1 < 3; i1++) for (int j1 = 0; j1 < 3; j1++) for (int k1 = 0; k1 < 3;
k1++)
            if (state[i1][j1][k1] != 0 && state[i1][j1][k1] != goal.state[i1][j1][k1])
                for (int i2 = 0; i2 < 3; i2++) for (int j2 = 0; j2 < 3; j2++) for (int k2 = 0; k2
< 3; k2++)
                    if(state[i1][j1][k1] == goal.state[i2][j2][k2]) {
                        sum += man_distance(k1, j1, i1, k2, j2, i2);
                        break;
                    }
        f = g + sum;
        return sum;
    }
```

```
    Node* east(const Node& goal) { /* create child node after action: east */
        if (blank[x] == 2) /* action not applicable... abort */
            return nullptr;
        Node* new_node = new Node(*this);
        new_node->prev = this;
        new_node->path.push_back('E');
        swap(new_node->state[blank[z]][blank[y]][blank[x]],
             new_node->state[blank[z]][blank[y]][++new_node->blank[x]]); /* update state */
        new_node->f = ++new_node->g + new_node->heurstic(goal); /* update g, f */
        return new_node;
    }
    Node* west(const Node& goal) { /* create child node after action: west; similar to above... */
        if (blank[x] == 0)
            return nullptr;
        Node* new_node = new Node(*this);
        new_node->prev = this;
        new_node->path.push_back('W');
        swap(new_node->state[blank[z]][blank[y]][blank[x]],
             new_node->state[blank[z]][blank[y]][--new_node->blank[x]]);
        new_node->f = ++new_node->g + new_node->heurstic(goal);
        return new_node;
    }
    Node* north(const Node& goal) {
        if (blank[y] == 0)
            return nullptr;
        Node* new_node = new Node(*this);
        new_node->prev = this;
        new_node->path.push_back('N');
        swap(new_node->state[blank[z]][blank[y]][blank[x]],
             new_node->state[blank[z]][--new_node->blank[y]][blank[x]]);
        new_node->f = ++new_node->g + new_node->heurstic(goal);
        return new_node;
    }
    Node* south(const Node& goal) {
        if (blank[y] == 2)
            return nullptr;
        Node* new_node = new Node(*this);
        new_node->prev = this;
        new_node->path.push_back('S');
        swap(new_node->state[blank[z]][blank[y]][blank[x]],
             new_node->state[blank[z]][++new_node->blank[y]][blank[x]]);
        new_node->f = ++new_node->g + new_node->heurstic(goal);
        return new_node;
    }
    Node* up(const Node& goal) {
        if (blank[z] == 0)
```

```
            return nullptr;
        Node* new_node = new Node(*this);
        new_node->prev = this;
        new_node->path.push_back('U');
        swap(new_node->state[blank[z]][blank[y]][blank[x]],
             new_node->state[--new_node->blank[z]][blank[y]][blank[x]]);
        new_node->f = ++new_node->g + new_node->heurstic(goal);
        return new_node;
    }
    Node* down(const Node& goal) {
        if (blank[z] == 2)
            return nullptr;
        Node* new_node = new Node(*this);
        new_node->prev = this;
        new_node->path.push_back('D');
        swap(new_node->state[blank[z]][blank[y]][blank[x]],
             new_node->state[++new_node->blank[z]][blank[y]][blank[x]]);
        new_node->f = ++new_node->g + new_node->heurstic(goal);
        return new_node;
    }
    int f_val() {
        return f;
    }
    Node* prev_node() {
        return prev;
    }
    const string& solution() {
        return path;
    }
};

class Less { /* priority queue comparator */
public:
    bool operator()(Node* a, Node* b) {
        return *a > *b;
    }
};

bool validate_child(Node* child, priority_queue<Node*, vector<Node*>, Less>& frontier,
vector<Node*>& reached) {
    if (child == nullptr) /* returned by inapplicable action... abort */
        return false;
    for (int i = 0; i < reached.size(); i++) {
        if (*child == *reached[i]) { /* repeated state... */
            if (*child < *reached[i]) { /* with less cost */
                reached.push_back(reached[i]);
```

```
                reached[i] = child;             // send the one with large cost to the end of
`reached`
                frontier.push(child);           // ... to prevent memory leakage
                return true;
            }
            else { /* with equal or greater cost... abort */
                delete child;
                return false;
            }
        }
    }
    reached.push_back(child);
    frontier.push(child);
    return true;
}

bool best_first_search(const Node& init, Node& goal, int& N, vector<int>& f_vals) {
    priority_queue<Node*, vector<Node*>, Less> frontier;
    frontier.push(new Node(init));
    vector<Node*> reached;
    reached.push_back(frontier.top());
    bool found = false;
    while(frontier.size()) {
        Node* node = frontier.top();
        frontier.pop();
        if (*node == goal) {
            goal = *node; /* save goal node */
            found = true;
            while (node != nullptr) {
                f_vals.push_back(node->f_val());
                node = node->prev_node();
            }
            break;
        }
        validate_child(node->east(goal), frontier, reached);
        validate_child(node->west(goal), frontier, reached);
        validate_child(node->north(goal), frontier, reached);
        validate_child(node->south(goal), frontier, reached);
        validate_child(node->up(goal), frontier, reached);
        validate_child(node->down(goal), frontier, reached);
    }
    N = (int)reached.size();
    while(reached.size()) { /* clean up */
        delete reached.back();
        reached.pop_back();
    }
```

```
    return found;
}

int main(int argc, const char * argv[]) {
    ifstream fin;
    fin.open("input.txt");
    if (!fin.good()) cerr << "Failed to open input.txt" << endl;
    int init_state[3][3][3], goal_state[3][3][3];
    for (int i = 0; i < 3; i++) for (int j = 0; j < 3; j++) for (int k = 0; k < 3; k++)
        fin >> init_state[i][j][k];
    for (int i = 0; i < 3; i++) for (int j = 0; j < 3; j++) for (int k = 0; k < 3; k++)
        fin >> goal_state[i][j][k];
    fin.close();
    Node init(init_state), goal(goal_state);
    init.heurstic(goal);
    goal.heurstic(goal);
    ofstream fout;
    fout.open("output.txt");
    if (!fout.good()) cerr << "Failed to open output.txt" << endl;
    int nodes_generated;
    vector<int> f_values;
    if (!best_first_search(init, goal, nodes_generated, f_values)) {
        cerr << "No solution found." << endl;
        return -1;
    }
    fout << init << goal;
    fout << goal.solution().size() << endl;     // d
    fout << nodes_generated << endl;            // N
    fout << goal.solution() << endl;            // A * (d)
    while (f_values.size()) {
        fout << f_values.back() << " ";         // f * (d + 1)
        f_values.pop_back();
    }
    fout.close();
    return 0;
}
```

Output 1

```
1 2 3
4 0 5
6 7 8

9 10 11
12 13 14
15 16 17

18 19 20
21 22 23
24 25 26

1 2 3
4 13 5
6 7 8

9 10 11
15 12 14
24 16 17

18 19 20
21 0 23
25 22 26

6
23
DWSDEN
6 6 6 6 6 6 6
```

Output 2

```
1 2 3
4 0 5
6 7 8

9 10 11
12 13 14
15 16 17

18 19 20
21 22 23
```

```
24 25 26

1 10 2
4 5 3
6 7 8

9 13 11
21 12 14
15 16 17

18 0 20
24 19 22
25 26 23

13
44
ENWDSWDSEENWN
13 13 13 13 13 13 13 13 13 13 13 13 13 13
```

Output 3

```
1 2 3
4 0 5
6 7 8

9 10 11
12 13 14
15 16 17

18 19 20
21 22 23
24 25 26

0 2 3
1 7 14
6 8 5

12 9 10
4 13 11
21 16 17

18 19 20
22 25 23
```

```
15 24 26

16
59
SENDNWWSDESWUNUN
16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16
```