CS3642-01 Programming Assignment #2 (Fall 2023)

Due: October 13, 2023 (11:30PM)

To implement Breadth-First Search (BFS), Uniform-Cost Search (UCS), Best-First Search (BFS), and A* Algorithm to solve the following 8-puzzle problem (i.e. find the goal):

8-puzzle Problem:

The 8-puzzle consists of eight numbered, movable tiles set in a 3x3 frame. One cell of the frame is always empty thus making it possible to move an adjacent numbered tile into the empty cell. Start with a random state (cannot be fixed). The goal state is listed below.

1	2	3
8		4
7	6	5

The program is to change the initial configuration into the goal configuration. A solution to the problem is an appropriate sequence of moves. You must write your own codes for the algorithms. Make sure your submission meets all of the requirements and free of plagiarism.

Your program should be able to address any initial configuration and provide a table of statistics below in your PDF file.

Algorithm	**Average number of nodes visited** (you need repeat each algorithm several times with different initial configuration)	**Average run time in your program**	**Your comment on these algorithms**
Breadth First Search			
Uniform Cost Search			
Best First Search			
A*using Nilsson's sequence score			

You may write your code in a contemporary language of your choice; typical languages would include C/C++, Python, Java, Ada, Pascal, Smalltalk, Lisp, and Prolog. A GUI interface is preferred.

1. Submit a PDF file of your well-commented source program, your design and your printed outputs (screen shots). Please include your codes in your PDF file. It is plagiarism to take any codes from the website or others. Try to understand the algorithm and implement the algorithm by your own. You must have all the information required in your PDF file.
2. Provide a video presentation of your programming assignment in MP3, YouTube, or any media.
3. Please upload items 1) and 2) above separately to D2L.

4. Restriction: No zipped files.

Adding the following two sections (I and II) at the beginning of your PDF including your code and outputs.

I. Your Information:

// Course: Artificial Intelligence
// Student name: Raehyeong Lee
// Student ID: 000996758
// Assignment #: Assignmnet 2
// Due Date: October 13, 2023
// Signature: Raehyeong Lee (Your signature assures that everything is your own work. Required.)
// Score: _____(Note: Score will be posted on D2L)

II. The statistics table:

Algorithm	**Average number of nodes visited** (you need repeat each algorithm several times with different initial configuration)	**Average run time in your experiments**	**Your comment on these algorithms**
Breadth First Search	**20+23+3+25+13+14=16.3**	**6**	**This must be the largest but the results was not working in right way.**
Uniform Cost Search	**20+23+3+25+13+14=16.3**	**6**	**It seems UCS and A* algorithms are not operating properly. The result of the algorithms is always same with BFS above.**
Best First Search	**60+63+3+75+95+16=52**	**6**	**This algorithm must be shortest average but, in my program, it has the biggest average number of nodes moving.**
A*using Nilsson's sequence score	**20+23+3+25+13+14=16.3**	**6**	**It seems UCS and A* algorithms are not operating properly. The result of the algorithms is always same with BFS above.**

1st I/O
Initial State:
0 6 3
1 5 4
2 8 7

Breadth-First Search Result:
Number of moves: 20
Uniform-Cost Search Result:
Number of moves: 20
Best-First Search Result:
Number of moves: 60
A* Algorithm Result:
Number of moves: 20

2nd I/O
Initial State:
1 6 7
0 5 3
2 8 4

Breadth-First Search Result:
Number of moves: 23
Uniform-Cost Search Result:
Number of moves: 23
Best-First Search Result:
Number of moves: 63
A* Algorithm Result:
Number of moves: 23

3rd I/O
Initial State:
8 1 7
3 5 0
2 4 6

Breadth-First Search Result:
Number of moves: 25
Uniform-Cost Search Result:
Number of moves: 25
Best-First Search Result:
Number of moves: 75
A* Algorithm Result:
Number of moves: 25

4th I/O
Initial State:
1 2 3
8 4 5
7 0 6

Breadth-First Search Result:
Number of moves: 3
Uniform-Cost Search Result:
Number of moves: 3
Best-First Search Result:
Number of moves: 3
A* Algorithm Result:
Number of moves: 3

5th I/O
Initial State:
1 3 6
8 4 5
7 0 2

Breadth-First Search Result:
Number of moves: 13
Uniform-Cost Search Result:
Number of moves: 13
Best-First Search Result:
Number of moves: 95
A* Algorithm Result:
Number of moves: 13

6th I/O
Initial State:
2 5 3
1 6 4
0 8 7

Breadth-First Search Result:
Number of moves: 14
Uniform-Cost Search Result:
Number of moves: 14
Best-First Search Result:
Number of moves: 16
A* Algorithm Result:
Number of moves: 14

Program Codes

```
import heapq
import copy

#goal state
goal_state = [[1, 2, 3], [8, 0, 4], [7, 6, 5]]

#up, down, left, right
movements = [(-1, 0), (1, 0), (0, -1), (0, 1)]

#find the empty tile
def find_empty_tile(state):
    for i in range(3):
        for j in range(3):
            if state[i][j] == 0:
                return i, j

#check if a state is valid
def is_valid(state):
    return all(0 <= state[i][j] < 9 for i in range(3) for j in range(3))

#print the puzzle state
def print_state(state):
    for row in state:
        print(" ".join(map(str, row)))
    print()

def bfs(initial_state):
    # Create a queue for BFS
    queue = [(initial_state, [])]
    visited = set()

    while queue:
        state, path = queue.pop(0)
        if state == goal_state:
            return path

        empty_i, empty_j = find_empty_tile(state)

        for move_i, move_j in movements:
            new_i, new_j = empty_i + move_i, empty_j + move_j
            if 0 <= new_i < 3 and 0 <= new_j < 3:
                new_state = copy.deepcopy(state)
                new_state[empty_i][empty_j], new_state[new_i][new_j] = new_state[new_i][new_j], new_state[empty_i][empty_j]
                if tuple(map(tuple, new_state)) not in visited and is_valid(new_state):
                    queue.append((new_state, path + [(new_i, new_j)]))
                    visited.add(tuple(map(tuple, new_state)))

def ucs(initial_state):
    priority_queue = [(0, initial_state, [])]
    visited = set()

    while priority_queue:
        cost, state, path = heapq.heappop(priority_queue)
        if state == goal_state:
            return path

        empty_i, empty_j = find_empty_tile(state)

        for move_i, move_j in movements:
            new_i, new_j = empty_i + move_i, empty_j + move_j
            if 0 <= new_i < 3 and 0 <= new_j < 3:
                new_state = copy.deepcopy(state)
                new_state[empty_i][empty_j], new_state[new_i][new_j] = new_state[new_i][new_j], new_state[empty_i][empty_j]
                if tuple(map(tuple, new_state)) not in visited and is_valid(new_state):
                    new_cost = cost + 1
                    heapq.heappush(priority_queue, (new_cost, new_state, path + [(new_i, new_j)]))
                    visited.add(tuple(map(tuple, new_state)))

def heuristic(state):
    total_distance = 0
    for i in range(3):
        for j in range(3):
            if state[i][j] != 0:
```

```
            target_i, target_j = divmod(state[i][j] - 1, 3)
            total_distance += abs(target_i - i) + abs(target_j - j)
    return total_distance

def best_first_search(initial_state, heuristic):
    priority_queue = [(heuristic(initial_state), initial_state, [])]
    visited = set()

    while priority_queue:
        _, state, path = heapq.heappop(priority_queue)
        if state == goal_state:
            return path

        empty_i, empty_j = find_empty_tile(state)

        for move_i, move_j in movements:
            new_i, new_j = empty_i + move_i, empty_j + move_j
            if 0 <= new_i < 3 and 0 <= new_j < 3:
                new_state = copy.deepcopy(state)
                new_state[empty_i][empty_j], new_state[new_i][new_j] = new_state[new_i][new_j], new_state[empty_i][empty_j]
                if tuple(map(tuple, new_state)) not in visited and is_valid(new_state):
                    heapq.heappush(priority_queue, (heuristic(new_state), new_state, path + [(new_i, new_j)]))
                    visited.add(tuple(map(tuple, new_state)))

#Nilsson's sequence score
def nilsson_sequence_score(state):
    score = 0
    for i in range(3):
        for j in range(3):
            if state[i][j] != 0:
                target_i, target_j = divmod(state[i][j] - 1, 3)
                if (i, j) != (target_i, target_j):
                    score += 2
                    if (i, j) != (0, 0) and state[i][j] == goal_state[i][j]:
                        score -= 2
                    elif (i, j) != (0, 1) and state[i][j] == goal_state[i][j]:
                        score -= 2
    return score

def a_star_nilsson(initial_state):
    priority_queue = [(nilsson_sequence_score(initial_state) + 0, initial_state, [])]
    visited = set()

    while priority_queue:
        _, state, path = heapq.heappop(priority_queue)
        if state == goal_state:
            return path
        empty_i, empty_j = find_empty_tile(state)
        for move_i, move_j in movements:
            new_i, new_j = empty_i + move_i, empty_j + move_j
            if 0 <= new_i < 3 and 0 <= new_j < 3:
                new_state = copy.deepcopy(state)
                new_state[empty_i][empty_j], new_state[new_i][new_j] = new_state[new_i][new_j], new_state[empty_i][empty_j]
                if tuple(map(tuple, new_state)) not in visited and is_valid(new_state):
                    new_cost = len(path) + 1
                    heapq.heappush(priority_queue, (new_cost + nilsson_sequence_score(new_state), new_state, path + [(new_i, new_j)]))
                    visited.add(tuple(map(tuple, new_state)))

#initial state
initial_state = [[2, 5, 3], [1, 6, 4], [0, 8, 7]]

print("Initial State:")
print_state(initial_state)

print("Breadth-First Search Result:")
bfs_result = bfs(initial_state)
print("Number of moves:", len(bfs_result))
for move in bfs_result:
    print(move)
print()

print("Uniform-Cost Search Result:")
ucs_result = ucs(initial_state)
print("Number of moves:", len(ucs_result))
for move in ucs_result:
```

```
    print(move)
print()

print("Best-First Search Result:")
best_first_search_result = best_first_search(initial_state, heuristic)
print("Number of moves:", len(best_first_search_result))
for move in best_first_search_result:
    print(move)
print()

print("A* Algorithm Result:")
a_star_result = a_star_nilsson(initial_state)
print("Number of moves:", len(a_star_result))
for move in a_star_result:
    print(move)
```

Screenshots of Output examples

```
Initial State:
0 6 3
1 5 4
2 8 7

Breadth-First Search (BFS) Result:
Number of moves: 20
(1, 0)
(2, 0)
(2, 1)
(1, 1)
(0, 1)
(0, 2)
(1, 2)
(1, 1)
(2, 1)
(2, 2)
(1, 2)
(1, 1)
(1, 0)
(2, 0)
(2, 1)
(2, 2)
(1, 2)
(0, 2)
(0, 1)
(1, 1)

Uniform-Cost Search (UCS) Result:
Number of moves: 20
(1, 0)
(2, 0)
(2, 1)
(1, 1)
(0, 1)
(0, 2)
(1, 2)
(1, 1)
(2, 1)
(2, 2)
(1, 2)
(1, 1)
(1, 0)
(2, 0)
(2, 1)
(2, 2)
(1, 2)
(0, 2)
(0, 1)
(1, 1)

Best-First Search Result:
Number of moves: 60
(1, 0)
```

```
Initial State:
2 8 3
1 6 4
7 0 5

Breadth-First Search (BFS) Result:
Number of moves: 5
(1, 1)
(0, 1)
(0, 0)
(1, 0)
(1, 1)

Uniform-Cost Search (UCS) Result:
Number of moves: 5
(1, 1)
(0, 1)
(0, 0)
(1, 0)
(1, 1)

Best-First Search Result:
Number of moves: 15
(2, 2)
(1, 2)
(1, 1)
(0, 1)
(0, 0)
(1, 0)
(1, 1)
(2, 1)
(2, 2)
(1, 2)
(1, 1)
(2, 1)
(2, 2)
(1, 2)
(1, 1)

A* Algorithm Result:
Number of moves: 5
(1, 1)
(0, 1)
(0, 0)
(1, 0)
(1, 1)
PS C:\Users\lrh14>
```

```
Initial State:
2 5 3
1 6 4
0 8 7

Breadth-First Search (BFS) Result:
Number of moves: 14
(2, 1)
(2, 2)
(1, 2)
(1, 1)
(0, 1)
(0, 0)
(1, 0)
(2, 0)
(2, 1)
(1, 1)
(1, 2)
(2, 2)
(2, 1)
(1, 1)

Uniform-Cost Search (UCS) Result:
Number of moves: 14
(2, 1)
(2, 2)
(1, 2)
(1, 1)
(0, 1)
(0, 0)
(1, 0)
(2, 0)
(2, 1)
(1, 1)
(1, 2)
(2, 2)
(2, 1)
(1, 1)

Best-First Search Result:
Number of moves: 16
(2, 1)
(2, 2)
(1, 2)
(1, 1)
(0, 1)
(0, 0)
(1, 0)
(2, 0)
(2, 1)
(2, 2)
(1, 2)
(1, 1)
(2, 1)
(2, 2)
(1, 2)
(1, 1)

A* Algorithm Result:
Number of moves: 14
(2, 1)
(2, 2)
(1, 2)
(1, 1)
(0, 1)
(0, 0)
(1, 0)
(2, 0)
(2, 1)
(1, 1)
(1, 2)
(2, 2)
(2, 1)
(1, 1)
```

```
A* Algorithm Result:
Number of moves: 20
(0, 1)
(0, 2)
(1, 2)
(1, 1)
(1, 0)
(2, 0)
(2, 1)
(2, 2)
(1, 2)
(0, 2)
(0, 1)
(1, 1)
(1, 0)
(0, 0)
(0, 1)
(1, 1)
(1, 0)
(2, 0)
(2, 1)
(1, 1)
```