Alexandria University
Faculty of Engineering
Computer and Systems Engineering Dept.
CS482:Artificial Intelligence

Using Informed and Uninformed Search Algorithms to Solve 8-Puzzle

Assignment(1)

Report

Name : Fatma Mohamed Abd El-Aty.

Id : 43.

- **Overall algorithm :**
  - Firstly I take an initial state and perform the three algorithms on it by order (BFS, DFS, A*(1→ with Manhattan Distance Heuristics),(2→ with Euclidean Distance Heuristics)).
  - All of the search algorithms written based on the scudo code that provided in the assignment pdf file :
    - BFS :

```
function BREADTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    frontier = Queue.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.dequeue()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.enqueue(neighbor)

    return FAILURE
```

    - DFS :

```
function DEPTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    frontier = Stack.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.pop()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.push(neighbor)

    return FAILURE
```

- A*(1→ with Manhattan Distance Heuristics),(2→ with Euclidean Distance Heuristics) :

```
function A-STAR-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :  /* Cost f(n) = g(n) + h(n) */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

  - And within each method I calculate :
    - path to goal
    - cost of path
    - number of nodes that expanded
    - search depth
    - running time

- **Main functions :**
  - BFS : used to perform the bfs algorithm.
  - DFS : used to perform the bfs algorithm.
  - A : used to perform the a* algorithm.
  - allocate_neighbors : used to find the neighbors of the state and push it in the state's vector that called "neighbors".
  - is_here_with_stack : that checks if the state exists in the explored set or the frontier (frontier → stack)(with DFS).
  - is_here_with_queue : that checks if the state exists in the explored set or the frontier (frontier → queue)(with BFS).
  - is_here_with_priorityQueue : that checks if the state exists in the explored set or the frontier (frontier → priorityQueue)(with A*).
  - Total cost : calculate the f(n) in the A* search.

- **Data structures used :**
  - I used a class called “State” which has the following attributes :
    - vector of int called “tiles”.
    - int cost.
    - int total_cost.
    - vector of states called “neighbors”.
    - vector of states called “parents”.

- **Sample runs :**
  - initial state : 1 4 2 3 0 5 6 7 8
    - BFS :

```
BFS : (Expanded nodes = 6)(Time = 3992microsecond)
Success
Total_Cost : 2
Step 0 :  (with Cost = 0),(In depth = 0)
1 4 2
3 0 5
6 7 8

Step 1 :  (with Cost = 1),(In depth = 1)
1 0 2
3 4 5
6 7 8

Step 2 :  (with Cost = 2),(In depth = 2)
0 1 2
3 4 5
6 7 8
```

    - DFS :

```
DFS : (Expanded nodes = 3)(Time = 6746microsecond)
Success
Total_Cost : 2
Step 0 :  (with Cost = 0),(In depth = 0)
1 4 2
3 0 5
6 7 8

Step 1 :  (with Cost = 1),(In depth = 1)
1 0 2
3 4 5
6 7 8

Step 2 :  (with Cost = 2),(In depth = 2)
0 1 2
3 4 5
6 7 8
```

- A*(1):

```
A 1 : (Expanded nodes = 3)(Time = 23577microsecond)
Success
Total_Cost : 2
Step 0 :  (with Cost(g)= 0), (with h = 4),(with Total_cost(f) = 4),(In depth = 0)
1 4 2
3 0 5
6 7 8

Step 1 :  (with Cost(g)= 1), (with h = 2),(with Total_cost(f) = 3),(In depth = 1)
1 0 2
3 4 5
6 7 8

Step 2 :  (with Cost(g)= 2), (with h = 0),(with Total_cost(f) = 2),(In depth = 2)
0 1 2
3 4 5
6 7 8
```

- A*(2):

```
A 2 : (Expanded nodes = 3)(Time = 78858microsecond)
Success
Total_Cost : 2
Step 0 : (with Cost(g)= 0), (with h = 3),(with Total_cost(f) = 3),(In depth = 0)
1 4 2
3 0 5
6 7 8

Step 1 : (with Cost(g)= 1), (with h = 2),(with Total_cost(f) = 3),(In depth = 1)
1 0 2
3 4 5
6 7 8

Step 2 :  (with Cost(g)= 2), (with h = 0),(with Total_cost(f) = 2),(In depth = 2)
0 1 2
3 4 5
6 7 8
```

- initial state : 0 1 2 3 4 5 6 7 8

```
BFS : (Expanded nodes = 1)(Time = 93microsecond)
Success
Total_Cost : 0
Step 0 :  (with Cost = 0),(In depth = 0)
0 1 2
3 4 5
6 7 8

DFS : (Expanded nodes = 1)(Time = 88microsecond)
Success
Total_Cost : 0
Step 0 :  (with Cost = 0),(In depth = 0)
0 1 2
3 4 5
6 7 8

A 1 : (Expanded nodes = 1)(Time = 324microsecond)
Success
Total_Cost : 0
Step 0 :  (with Cost(g)= 0), (with h = 0),(with Total_cost(f) = 0),(In depth = 0)
0 1 2
3 4 5
6 7 8

A 2 : (Expanded nodes = 1)(Time = 689microsecond)
Success
Total_Cost : 0
Step 0 :  (with Cost(g)= 0), (with h = 0),(with Total_cost(f) = 0),(In depth = 0)
0 1 2
3 4 5
6 7 8
```

- initial state : 3 1 2 0 4 5 6 7 8
  - BFS :

```
BFS : (Expanded nodes = 2)(Time = 434microsecond)
Success
Total_Cost : 1
Step 0 :  (with Cost = 0),(In depth = 0)
3 1 2
0 4 5
6 7 8

Step 1 :  (with Cost = 1),(In depth = 1)
0 1 2
3 4 5
6 7 8
```

- DFS :

```
DFS : (Expanded nodes = 2)(Time = 1465microsecond)
Success
Total_Cost : 1
Step 0 :  (with Cost = 0),(In depth = 0)
3 1 2
0 4 5
6 7 8

Step 1 :  (with Cost = 1),(In depth = 1)
0 1 2
3 4 5
6 7 8
```

- A*(1):

```
A 1 : (Expanded nodes = 2)(Time = 5023microsecond)
Success
Total_Cost : 1
Step 0 :  (with Cost(g)= 0), (with h = 2),(with Total_cost(f) = 2),(In depth = 0)
3 1 2
0 4 5
6 7 8

Step 1 :  (with Cost(g)= 1), (with h = 0),(with Total_cost(f) = 1),(In depth = 1)
0 1 2
3 4 5
6 7 8
```

- A*(2):

```
A 2 : (Expanded nodes = 2)(Time = 14307microsecond)
Success
Total_Cost : 1
Step 0 : (with Cost(g)= 0), (with h = 2),(with Total_cost(f) = 2),(In depth = 0)
3 1 2
0 4 5
6 7 8

Step 1 :  (with Cost(g)= 1), (with h = 0),(with Total_cost(f) = 1),(In depth = 1)
0 1 2
3 4 5
6 7 8
```