Contents

1 Learning Goals

By the end of the exercise, you should be able to

- Formulate a real-world problem as a search problem by defining the states, the initial state, the goal state, the action, the successor function, and the cost function.
- Trace the execution of and implement uninformed search algorithms including Breadth-First Search and Depth-First Search.
- Given a scenario, explain why it is or it is not appropriate to use an uninformed algorithm.

2 References

2.1 The 8-puzzle

An instance of the 8-puzzle consists of a 3×3 board with 8 numbered tiles and a blank space. Each tile has a number from 1 to 8. A tile adjacent to the blank space can slide into the space. The goal is to reach a specified goal state, such as the one shown below.

Initial State

5	3	
8	7	6
2	4	1

Goal State

1	2	3
4	5	6
7	8	

2.2 A generic search algorithm

Algorithm 1 Search

```
1: let the frontier to be an empty list
2: add initial state to the frontier
3: while the frontier is not empty do
4:     remove curr_state from the frontier
5:     if curr_state is a goal state then
6:         return curr_state
7:     end if
8:     get all the successors of curr_state
9:     add all the successors to the frontier
10: end while
11: return no solution
```

3 Formulating a Search Problem

1. Formulate the 8-puzzle as a search problem. Be sure to define the states, the initial state, the goal state, the action, the successor function, and the cost function.

   Solution: State: Each state is given by $x_{00}x_{10}x_{20}, x_{01}x_{11}x_{21}, x_{02}x_{12}x_{22}$ where x_{ij} is the value in the space at row i and column j. If the space is blank, the value is zero. $i, j \in \{0, 1, 2\}$. $x_{ij} \in \{0, \ldots, 8\}$. The values of x_ij must be unique.

   Initial state: $530, 876, 241$

   Goal state: $123, 456, 780$

   Action: Move the blank space up, down, left or right, wherever possible.

   Successor function: The resulting state after taking one action.

   Cost function: Each move has a cost of 1.

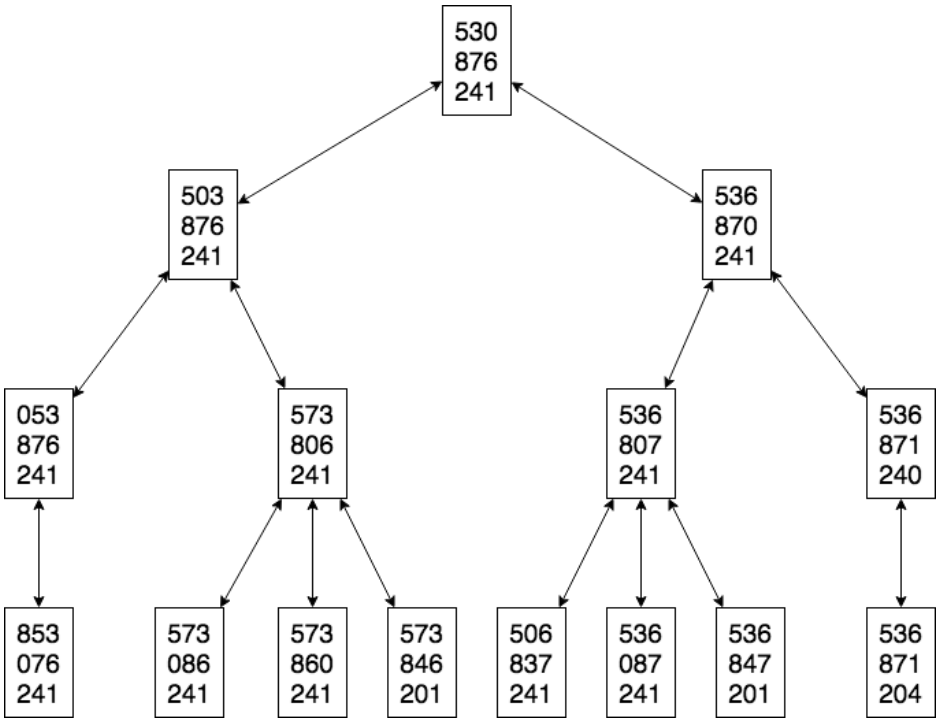
2. List all the successors of the initial state.

   Solution: Move the blank space left: $503, 876, 241$

   Move the blank space down: $536, 870, 241$

3. Starting with the initial state, draw the search graph until there are at least 6 nodes in the graph.

   Solution:



   Note that:

- Every arrow is bi-directional because every action can be reversed.
- Your graph may be different depending on which node you choose to generate successors.

4 Breadth-First Search

1. Consider the generic search algorithm provided on the reference sheet. If we add the two requirements below, the algorithm becomes the Breadth-first search algorithm.

   (a) We will add nodes to the frontier in lexicographical order. For example, we will add the state $503,876,241$ before state $536,870,241$.

   (b) The frontier is a queue (FIFO).

   Execute the Breadth-first search algorithm on the 8-puzzle problem until you have expanded 5 nodes. Describe the execution step by step. For each step, be sure to do the following:

   - Give the node removed from the frontier.
   - List all the nodes that are added to the frontier.
   - Finally, list all the nodes in the frontier.

   We have given you the first two steps below.

   In addition, as you execute the algorithm, draw the search tree. In the search tree, label the expanded nodes in order of expansion.

   (1) Add $530,876,241$ to the frontier.
   frontier = { 530,876,241 }

   (2) Remove 530,876,241 from the frontier.
   Add 503,876,241 and 536,870,241 to the frontier.
   frontier = { 503,876,241, 536,870,241 }

   (3)

   (4)

   (5)

   (6)

Draw the search tree below.

Solution: The steps of execution of the breadth-first search algorithm are below.

(1) Add 530,876,241 to the frontier.
frontier = { 530,876,241 }

(2) Remove 530,876,241 from the frontier.
Add 503,876,241 and 536,870,241 to the frontier.
frontier = { 503,876,241, 536,870,241 }

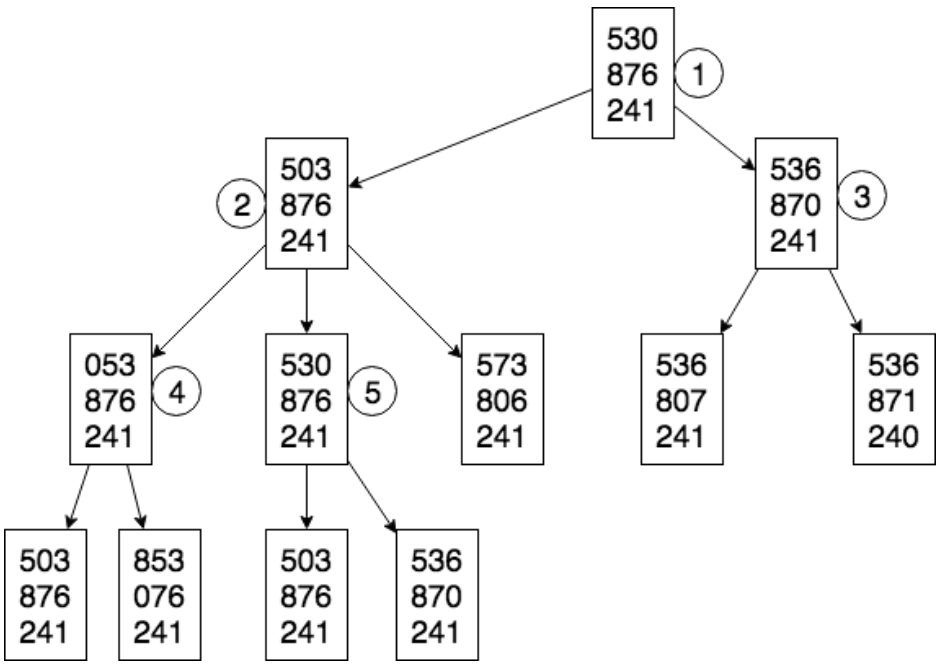
(3) Remove 503,876,241 from the frontier.
Add 053,876,241, 530,876,241, and 573,806,241 to the frontier.
frontier = { 536,870,241, 053,876,241, 530,876,241, 573,806,241 }

(4) Remove 536,870,241 from the frontier.
Add 530,876,241, 536,807,241 and 536,871,240 to the frontier.
frontier = { 053,876,241, 530,876,241, 573,806,241, 536,807,241, 536,871,240}

(5) Remove 053,876,241 from the frontier.
Add 503,876,241 and 853,076,241 to the frontier.
frontier = { 530,876,241, 573,806,241, 536,807,241, 536,871,240, 853,076,241 }

(6) Remove 530,876,241 from the frontier.
Add 503,876,241 and 536,870,241 to the frontier.
frontier = { 573,806,241, 536,807,241, 536,871,240, 853,076,241, 503,876,241 and 536,870,241 }

See the search tree below.



2. Could you describe the behaviour of Breadth-first search at a high level?

   Solution: The algorithm visits the states level by level. It visits all the states that are 1 edge away. Then it visits all the states that are 2 edges away.

3. If we add states to the frontier in a different order, does the search tree change?

   Solution: Yes. We would explore the states on each level in a different order.

   If we add states to the frontier in a different order, does the high-level property of the algorithm (as described in the previous question) change?

   Solution: No. We still explore the states level by level.

5 Depth-First Search

1. Consider the generic search algorithm provided on the reference sheet. If we add the two requirements below, the algorithm becomes the Depth-first search algorithm.

   (a) We will add nodes to the frontier in lexicographical order. For example, we will add the state $503,876,241$ before state $536,870,241$.

   (b) The frontier is a stack (LIFO).

   Execute the Depth-first search algorithm on the 8-puzzle problem until you have expanded 5 nodes. Describe the execution step by step. For each step, be sure to do the following:

   - Give the node removed from the frontier.
   - List all the nodes that are added to the frontier.
   - Finally, list all the nodes in the frontier.

   We have given you the first two steps below.

   In addition, as you execute the algorithm, draw the search tree. In the search tree, label the expanded nodes in order of expansion.

   (1) Add 530,876,241 to the frontier.
   frontier = { 530,876,241 }

   (2) Remove 530,876,241 from the frontier.
   Add 503,876,241 and 536,870,241 to the frontier.
   frontier = { 503,876,241, 536,870,241 }

   (3)

   (4)

   (5)

   (6)

Draw the search tree below.

Solution:

(a) Add 530,876,241 to the frontier.
frontier = { 530,876,241 }

(b) Remove 530,876,241 from the frontier.
Add 503,876,241 and 536,870,241 to the frontier.
frontier = { 503,876,241, 536,870,241 }

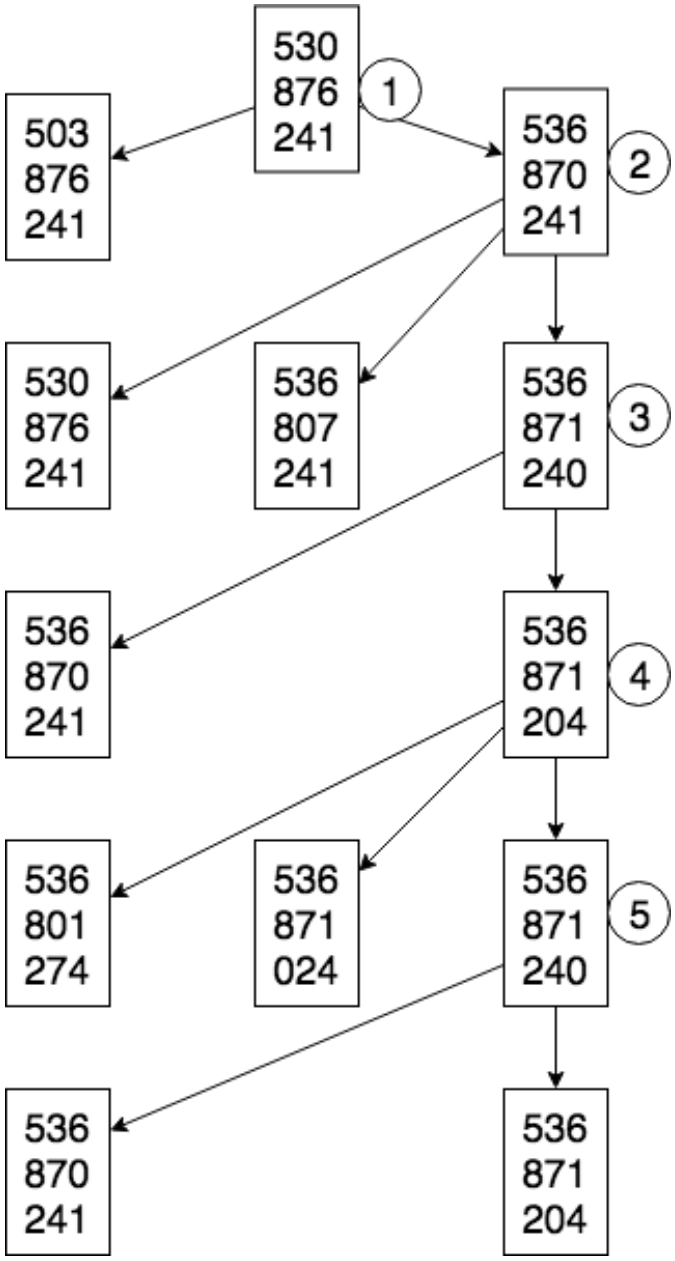
(c) Remove 536,870,241 from the frontier.
Add 530,876,241, 536,807,241, and 536,871,240 to the frontier.
frontier = { 503,876,241, 530,876,241, 536,807,241, 536,871,240}

(d) Remove 536,871,240 from the frontier.
Add 536,870,241 and 536,871,204 to the frontier.
frontier = { 503,876,241, 530,876,241, 536,807,241, 536,870,241, 536,871,204}

(e) Remove 536,871,204 from the frontier.
Add 536,801,274, 536,871,024, and 536,871,240 to the frontier.
frontier = { 503,876,241, 530,876,241, 536,807,241, 536,870,241, 536,801,274, 536,871,024, 536,871,240}

(f) Remove 536,871,240 from the frontier.
Add 536,870,241 and 536,871,204 to the frontier.
frontier = { 503,876,241, 530,876,241, 536,807,241, 536,870,241, 536,801,274, 536,871,024, 536,870,241, 536,871,204}

See the search tree below.



2. Could you describe the behaviour of Depth-first search at a high level?

   Solution: Depth-first search proceeds immediately to the deepest level of the search tree. Once it reaches a state without successors, it backtracks to a state that has unexplored successors and continues.

6 BFS or DFS?

Consider the scenarios below. Which of BFS and DFS would you choose?

Assume that we are using the naive versions of these algorithms which do not keep track of explored states.

1. Memory is limited.

   Solution: DFS is better — BFS uses much more memory than DFS.

2. All solutions are deep in the tree.

   Solution: Both are fine, but DFS is better — DFS might find a solution faster, although the solution might not be optimal.

3. There are infinite paths in the tree. (The search graph contains cycles.)

   Solution: BFS, not DFS — DFS will not terminate and will go down an infinite path forever.

4. The branching factor is large.

   Solution: Both are fine but DFS is better — BFS will spend a lot of time generating successors.

5. We must find the shallowest goal node.

   Solution: We should use BFS and not DFS. BFS is guaranteed to find the shallowest goal node.

6. Some solutions are very shallow.

   Solution: Both are fine but BFS is better. BFS will find the shallow solution faster.