Machine, Data and Learning (CS7.301)

Spring 2022, IIIT Hyderabad
Assignment 2

Abhinav S Menon

Question 1

The five parameters of the blocks world problem are:

- States: A state would consist of a description of which blocks are on the table, and which blocks are on top of the others. For example, "A and B are on the table and C is on top of A".
- Actions: An action consists of moving a block from one location to another (each location is either the table or the top of a block). The admissible actions are those in which the block being moved has no block on top of it.
- The initial state is specified by a certain instance of the problem. For example, all blocks might be on the table.
- The goal state is also specified by the instance of the problem.
- The cost of a path can be defined as the number of moves in it.

Question 2

Breadth-First Search

In BFS, the expansions of nodes are added to the end of the open list (the open list is a queue).

Initial state:

```
[ C   ]
  A B
```

Iteration 1:

```
        B
[   C ; C ;       ]
  A B   A   A B C
```

Iteration 2 (expanding node 1 above, adding to end of list):

```
  B                 A
[ C ;       ; C   ; C ;      ]
  A   A B C   A B   B   A B C
```

Iteration 3 (expanding node 1 above, adding to end of list):

```
               A
[      ; C   ; C ;       ; C   ]
 A B C   A B   B   A B C   A B
```

Depth-First Search

In DFS, the expansions of nodes are added to the beginning of the open list (the open list is a stack).

Initial state:

```
[ C   ]
  A B
```

Iteration 1:

```
        B
[   C ; C ;       ]
  A B   A   A B C
```

Iteration 2 (expanding node 1 above, adding to beginning of list):

```
        A           B
[ C   ; C ;       ; C ;       ]
  A B   B   A B C   A   A B C
```

Iteration 3 (expanding node 1 above, adding to the beginning of the list):

```
        B           A           B
[   C ; C ;       ; C ;       ; C ;       ]
  A B   A   A B C   B   A B C   A   A B C
```

Uniform-Cost Search

In UCS, the least-cost unexpanded node is expanded. In this case, however, we consider the path cost as the number of nodes; thus this becomes equivalent to BFS.

Initial state:

```
[ C   ]
  A B
```

Iteration 1:

```
        B
[   C ; C ;       ] # All have path costs 1
```

```
  A B   A   A B C
```

Iteration 2 (expanding node 1 above, adding to end of list):

```
  B                 A
[ C ;       ; C   ; C ;      ] # First two have path costs 1; next three have 2
  A   A B C   A B   B   A B C
```

Iteration 3 (expanding node 1 above, adding to end of list):

```
                A
[      ; C   ; C ;       ; C   ] # First has path cost 1; next four have 2
 A B C   A B   B   A B C   A B
```

Question 3

Two examples of admissible heuristics that can be used for this are:

- Sum of absolute differences of heights of each block from their heights in the goal state (height of a block = number of blocks under it). For example, if we have A, C, B on a single stack (with B on the table), the distance is $|2-2|+|1-0|+|0-1|=2$.
  This heuristic is admissible as the absolute difference is a lower bound on the number of moves needed to achieve the state.
- Absolute difference between number of stacks in current state and in goal state. For example, if we have A and B on the table, and C on top of B, then the distance is $|2-1|=1$.
  This heuristic is also admissible as one move can change it by at most 1; thus it can only underestimate the cost.

Question 4

We will use the first heuristic above to carry out A* search. Thus, $h(n)$ is defined above, $g(n)$ is the depth of a node, and $f(n)=h(n)+g(n)$ is what we will minimise.

Initial State:

```
[ C   (4)]
  A B
```

Iteration 1:

```
                          B
[       (1+3);   C (1+4); C (1+4)]
  A B C        A B        A
```

Iteration 2:

```
[ A   (2+2);   A (2+2); B   (2+2);   B (2+2); # These are from node 1
  B C        B C        A C        A C

             B
      (1+3); C (1+4);                        # These carry over
A B C        A


C   (2+4);   C (2+4)]                        # These are also from node 1
A B        A B
```

Iteration 3:

```
                                             B
[   A (2+2); B   (2+2);   B (2+2);       (1+3); C (1+4) # These carry over
  B C        A C        A C        A B C        A


  A (3+2);                                             # This is from node 1
B C


C   (2+4);   C (2+4);                                  # These carry over
A B        A B

             C
      (3+3); A (3+4)]                                  # These are from node 1
A B C        B
```