CS360 – Homework #4

STRIPS

1) Formulate in STRIPS the planning problem of transforming the given start configuration of the eight-puzzle to the given goal configuration. (In the eight-puzzle, the empty tile can be swapped with its North, South, East or West neighbor.)

1	2	3
7		4
8	6	5

(a) Start configuration.

1	2	3
4	5	6
7	8	

(b) Goal configuration.

We use T1, T2, ..., T8 to denote the tiles and L1, L2, ..., L9 to denote the possible locations for those tiles. We use the following predicates:

At(X,Y): Tile X is at location Y.

Blank(X): There is no tile at location X.

Adjacent(X,Y): Location X is adjacent to location Y.

Our only action is Move(X,Y,Z), which moves tile X from its current location Y to an adjacent location Z. It is defined as follows:

```
Action Move(X,Y,Z):
Preconditions = {At(X,Y), Blank(Z), Adjacent(Y,Z)};
Effects = {-At(X,Y), -Blank(Z), At(X,Z), Blank(Y)};
```

Start configuration:

```
At(T1, L1), At(T2, L2), At(T3, L3),
At(T7, L4), Blank(L5), At(T4, L6),
At(T8, L7), At(T6, L8), At(T5, L9),
Adjacent(L1, L2), Adjacent(L2, L3), Adjacent(L4, L5),
Adjacent(L5, L6), Adjacent(L7, L8), Adjacent(L8, L9),
Adjacent(L1, L4), Adjacent(L4, L7), Adjacent(L2, L5),
Adjacent(L5, L8), Adjacent(L3, L6), Adjacent(L6, L9),
... and the reverse of each adjacency, that is, Adjacent (L2,L1) ...
```

Goal configuration:

```
At(T1, L1), At(T2, L2), At(T3, L3),
At(T4, L4), At(T5, L5), At(T6, L6),
At(T7, L7), At(T8, L8).
```

It is also okay to add Blank(L9) to the goal configuration, even though it is not necessary. This is because our Move action always swaps the blank tile with an adjacent tile and, if the initial configuration is valid (each tile is at a different location and one location is blank), any configuration we reach from the initial configuration is also valid.

2) In the Tower of Hanoi game (`http://en.wikipedia.org/wiki/Tower_of_Hanoi`) there are three rods (pegs) and a number of disks of different sizes which can slide onto any rod. In the initial state, the disks are stacked on the first rod in ascending order of size, with the smallest on top. In the goal state, the disks are stacked on the third rod in the same order. At each turn, the player can take the topmost disk on a stack and place it on top of another stack. However, a disk may not be placed on top of a smaller disk. Formulate in STRIPS the Tower of Hanoi game played with three disks.

We use D1, D2, D3 to denote the disks (D3 being the largest), and P1, P2, P3 to denote the pegs. We use the following predicates:

On(X,Y): Disk X is on disk/peg Y. For instance, On(D3, P1) means that the largest disk is the lowermost disk on the first peg.

Clear(X): There is no disk on top of disk/peg X.

Smaller(X,Y): Disk X is smaller than disk/peg Y (disk X can be placed on top of disk/peg Y). Notice that Y can be a peg, since any disk can be placed on any empty peg.

Our only action is Move(X,Y,Z), which moves disk X from its current position (on top of Y) to the top of Z. It is defined as follows:

```
Action Move(X,Y,Z):
Preconditions = {Clear(X), On(X,Y), Clear(Z), Smaller(X,Z)};
Effects = {-On(X,Y), Clear(Y), On(X,Z), -Clear(Z)};
```

Start configuration:

```
On(D1, D2), On(D2, D3), On(D3, P1),
clear(D1), clear(P2), clear(P3),
Smaller(D1, D2), Smaller(D1, D3), Smaller(D2, D3),
Smaller(D1, P1), Smaller(D1, P2), Smaller(D1, P3),
Smaller(D2, P1), Smaller(D2, P2), Smaller(D2, P3),
Smaller(D3, P1), Smaller(D3, P2), Smaller(D3, P3).
```

Note that we have also specified that any disk is smaller than any peg.

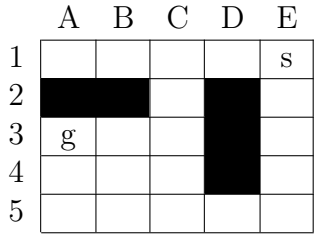
Goal configuration:

```
On(D1, D2), On(D2, D3), On(D3, P3).
```

It is also okay to add Clear(D1), Clear(P1), Clear(P2) to the goal configuration, even though it is not necessary (similar to Problem 1).

Breadth-First and Depth-First Search

3) A 4-neighbor gridworld is given below. In which order do depth-first search and breadth-first search (both with a sensible node pruning strategy) expand the cells when searching from s to g? Ties are broken in lexicographic order. That is, A1 is preferred over A2 and B1, and A2 is preferred over B1.



The node pruning strategy of breadth-first search is to prune nodes whenever some node in the search tree is already labeled with the same state. The node pruning strategy of depth-first search is to prune nodes whenever some node on the same branch of the search tree is already labeled with the same state (cycle detection).

Breadth-first search: (E1), (D1, E2), (C1, E3), (B1, C2, E4), (A1, C3, E5), (B3, C4, D5), (A3). The parentheses group states based on their depth in the search tree. Depending on the tie-breaking strategy, the order of expansions can change, but only within a group.

Depth-first search: E1, D1, C1, B1, A1, (backtrack to B1 and then C1), C2, C3, B3, A3.

4) Compare the advantages and disadvantages of breadth-first and depth-first search and discuss to which degree pruning of tree nodes is important for them.

In a finite state space, breadth-first search (BFS) and depth-first search (DFS) compare as follows:

- Both breadth-first search and depth-first search can use node pruning to decrease the number of expanded nodes and thus increase their efficiency. Breadth-first search needs to keep more information in memory than depth-first search and can then use more powerful node pruning strategies.

- BFS is complete (even without node pruning). DFS is not complete without cycle detection and thus not complete without any node pruning.
- BFS is optimal (assuming unit-cost edges), DFS offers no such guarantee.

In a finite tree with branching factor b, goal depth d, tree depth m, BFS and DFS compare as follows:

- In the worst case, BFS expands $O(b^d)$ states to find a solution, whereas DFS expands $O(b^m)$ states. Therefore, since the goal depth cannot be larger than the tree depth (that is, $d \leq m$), BFS has a better worst case performance.
- In the worst case, BFS requires memory to store $O(b^d)$ states, whereas DFS requires memory to store only $O(bm)$ states with an appropriate memory-deallocation strategy.

5) Does depth-first search always terminate if there is a path of finite length from the start to the goal? Why?

No. If it does not use proper cycle detection, it might get stuck in a loop. Even with cycle detection, if the state space is infinite (but there is a finite length path from the start to the goal), DFS might get stuck exploring an infinite subspace of the state space.

6) In Manhattan, you want to reach a given destination from your current location with as few left turns as possible. Can this be formulated as finding a minimum cost path in a graph? If so, how? If not, why not?

Yes. Each state is a pair of the agent's current location and the compass direction it is facing (North, East, South or West). There are four goal states, namely pairs of the given destination and the four compass directions. The actions are to move forward (cost 0, changes location depending on which way the agent is facing, the direction the agent is facing does not change), turn right and move forward (cost 0) and turn left and move forward (cost 1). If 0 cost actions are a problem for the search algorithm we want to use, we can instead use a cost of ϵ, where ϵ is smaller than 1/the number of states in our state space. Assuming Manhattan is finite, our state space representation is also finite.