TCSS 435 Programming Assignment 1

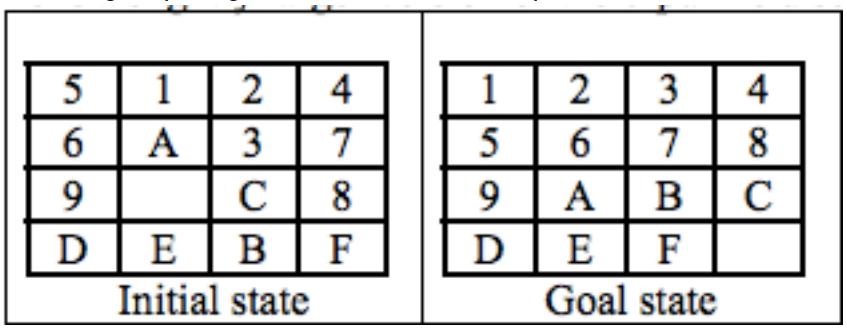
<u>NOTE:</u> Be sure to adhere to the University's **Policy on Academic Integrity** as discussed in class. Programming assignments are to be written individually and submitted programs must be the result of your own efforts. Any suspicion of academic integrity violation will be dealt with accordingly

Assignment Details:

For this programming assignment, you will create a set of search algorithms that find solutions to the 15-puzzle.

Problem Description:

The 15-puzzle is a slightly larger version of the 8-puzzle discussed in Class:



5	1	2	4
6	A	3	7
9		C	8
D	E	B	F

Initial state

1	2	3	4
5	6	7	8
9	A	B	C
D	E	F	

Goal state

The 15-puzzle has a higher average branching factor than the 8-puzzle, but the maximum branching factor is the same (4).

Assignment summary:

The search algorithms you are expected to implement are:

- Breadth-first search (BFS)
- Depth-first search (DFS) – depth-first search needs to check for cycles, or a solution will most likely never be found.
- Greedy best-first search (GBFS), for h(1) and h(2) mentioned in Russell & Norvig
- A* (AStar), for h(1) and h(2) mentioned in Russell & Norvig
- Either:
    - Depth-limited search (DLS), or
    - Iterative deepening (ID)

INPUT: Your program will accept instructions from the command line. The program should accept the following inputs in the following format:

- `“[initialstate]” [searchmethod] [options]`
- `[initialstate]` must contain sixteen characters, namely the digits 1-9, letters AF, and a space, in any order.
- `[searchmethod]` can be: BFS, DFS, DLS, ID, GBFS, AStar.
- `[options]` are only relevant for ***DLS*** (depth-limited search), where the option specifies the maximum depth to explore, ***GBFS*** and ***AStar***, where the option specifies which heuristic to use.
- Examples:
  - `“123456789AB DEFC” BFS`
  - `“123456789AB DEFC” DFS`
  - `“123456789AB DEFC” DLS 2`
  - `“123456789AB DEFC” GBFS h1`
  - `“123456789AB DEFC” AStar h2`

OUTPUT: The output should be a comma-separated list of integers listing the following format. These format should represent the state of your search tree at the moment the solution was discovered:

- `[depth], [numCreated], [numExpanded], [maxFringe]`
- `[depth]` represents the depth in the search tree where the solution is found. The integer will be zero if the solution is at the root and it will be “-1” if a solution was not found.
- `[numCreated]` is the counter that is incremented every time a node of the search tree is created (output 0 if depth == -1).
- `[numExpanded]` is the counter that will be incremented every time the search algorithm acquires the successor states to the current state; i.e. every time a node is pulled off the fringe and found not to be the solution (output 0 if depth == -1).
- `[maxFringe]` is the maximum size of the fringe at any point during the search (output 0 if depth == -1).
- Example:

  -
    ```
    java FifteenProblem "123456789ABC DFE" BFS

    3, 54, 17, 37
    ```

Important:

- Technically, the order in which you expand the nodes for exploration is arbitrary, but to standardize the output of your programs, please adhere to the following convention. Expand all nodes in the following order:
  - Right – (space moves right)
  - Down – (space moves down)
  - Left – (space moves left)
  - Up – (space moves up)
- Consider the following goal states: All puzzles are solvable to either of the two states (`"123456789ABCDEF " or "123456789ABCDFE "`)

Hints:

- The successors of a state in this problem depend greatly on the position of the blank spot. Rather than think about which tiles can move into the blank spot, try considering where the blank spot can move. Certain numerical qualities about this position will determine whether or not the blank can move left, right, up, or down. If a blank spot moves up, then its location is being swapped with the location above it.
- The heuristics can be generated by comparing the state being evaluated to the goal state. The number of misplaced tiles is easily calculated in time linear to the number of tiles in the puzzle, but the simple solution to the Manhattan distance requires time quadratic to the number of tiles in the puzzle.
- If you plan to start from scratch, spend a lot of time thinking about what data structures you will use for your fringe. Much of the variation between the algorithms comes in how to determine which elements to pull off the fringe next.
- Many of these algorithms are more similar than different. Think about how you can use polymorphism to make your life easier.

<u>Submission Guidelines:</u>

Submit your files on Canvas using the Programming Assignment 1 submission Link.
You will submit a zip file containing:

- Source code with necessary source documentation for your program.
- <u>Readme.txt:</u> A text file containing the following information:
    - Output for each of the search algorithm (BFS, DFS, GBFS for h1 and h2, AStar for h1 and h2, ID or DLS) in the format explained in the output section. Atleast show output for 2 initial state configuration.
    - Time complexity for each of search above mentioned search technique.