FIT3080 Assignment 1
Problem Solving as Search (15%)
Due September 2, 2016

Background

Consider a sliding block puzzle with an initial configuration such as the following:

B	B	B	W	W	W	E

There are three black tiles (B), three white tiles (W), and an empty cell (E). The puzzle has the following moves:

(a) A tile may move to an adjacent empty cell with unit cost.

(b) A tile may hop over at most two other tiles into an empty cell with a cost equal to the number of tiles hopped over.

In addition,

(c) If two moves have the same cost, prefer move right over move left, and prefer hopping over moving to an adjacent tile.

(d) We suggest operators that move the empty cell, rather than the tiles. The operators are: 1L, 2L, 3L, 1R, 2R and 3R.

The goal of the puzzle is to have all of the white tiles to the left of all of the black tiles (without regard to the position of the empty cell).

The Task

Your task is to write a program called `solvepuzzle` that solves this puzzle. Your program calls two main modules: one that implements the Backtrack algorithm and one that implements the Graphsearch algorithm, as specified by a command-line input (Section *The Input*):

- For Backtrack1 (from the lectures), you may devise a scheme for ordering the operators depending on the circumstances and the above preferences.
- For Graphsearch, you should implement **one procedure** that uses an ordering function to distinguish between Depth Limited Search (DLS) and A (or A*). You may implement Graphsearch or Treesearch.
- For algorithm A/A*:
  - Propose and implement a heuristic function for solving this problem.
  - Determine whether this function is admissible and/or monotonic. Is the resulting algorithm A or A*?

The Input

Your program, called `solvepuzzle`, should have the following parameters:

puzzle-string procedure-name outputfile-name Flag

For example,

```
solvepuzzle BBWEWBW A result1 0
```

where

- *puzzle-string* comprises a starting sequence of the symbols B, W and E (only one E is allowed),
- *procedure-name* is one of {BK, DLS, A}.
- *outputfile-name* is the name of the output file.
- *Flag* has value 0 when only the sequence of steps, resultant state and cost is to be printed (Section *The Output*), and value N when the diagnostic output of N steps (described in Section *Programming Requirements*) is to be printed.

The preferred programming language is Python. Your program should be activated as follows (**your program must compile on Python 3.5, which is installed in the labs**):

python *FullPath/*`solvepuzzle.py` *puzzle-string procedure-name FullPath/outputfile-name Flag*

where *FullPath* is the absolute path of the directory that contains your program and all relevant files.

IMPORTANT:

- If you don't know Python, please contact the teaching staff to discuss alternatives.
- Please make sure that you are familiar with how to handle command-line arguments.
- We plan to run your programs with input in the above format, hence programs which deviate from this format **will not be accepted**.
  - To help you ensure that your program is compliant, we have provided a syntax checker on moodle.
  - Failure to submit a compliant program will attract a deduction of 10 marks for every time we need to contact you because your program doesn't run.

The Output

Your program should produce a (hopefully optimal) sequence of moves to be performed to get from the starting configuration to the goal configuration, and the **accumulated cost** after each step. For example,

```
start BBBWWWE 0
3L    BBBEWWW 2
2L    BEBBWWW 3
3R    BWBBEWW 5
1R    BWBBWEW 6
3L    BWEBWBW 8
...
2L   WWEWBBB total_cost
```

Programming Requirements

- Your program should be written in Python.
- Backtrack requires a *BOUND*. Use a reasonable number based on your experience in solving the puzzle. If you make the bound too low, you will fail to find a solution.
- As mentioned above, your program receives an input parameter called *procedure-name*, which has possible values BK, DLS or A. If the value is BK, the Backtrack procedure is called; otherwise a Tree/Graphsearch procedure is called.
  IMPORTANT: You should implement only **one** Tree/Graphsearch procedure, where the *procedure-name* parameter (DLS or A) determines how the nodes in *OPEN* are ordered (DLS requires a bound L — this is a program variable that is not relevant to A). **If you implement more than one Graphsearch procedure, you will lose marks.**
- Regardless of whether you are running Backtrack or Graphsearch, your program should create a unique identifier for every generated node. The identifier of the start node should be **N0**, and other nodes should be identified as **N1, N2,** ... according to the order in which they are **generated**.
- Your program should work on different puzzles, including the above given one.
- The Graphsearch procedure should build a *search graph* or *search tree*. Each node in the search tree should have the following components: (1) an identifier, (2) the cost g of reaching that node, (3) a heuristic value h (not relevant for DLS), (4) a value f, where $f = g + h$, (5) a pointer to its children, and (6) a pointer to its parent (if you implement Treesearch, and your path is included in a node, then this pointer is not necessary).
- In diagnostic mode (for $Flag \geq 1$):
  - For Backtrack: each time an operator is applied, your program should output (1) the operator, (2) the identifier of the generated node, (3) the available operators, and (4) the list of moves executed so far in the path (not the rejected moves). In your output, indicate when Backtracking is performed and why (*DEADEND, BOUND REACHED, ANCESTOR, NO MORE OPS*) – if you implement only legal moves, you will not have DEADENDs.
  - For Tree/Graphsearch,

    * Each time a node is **expanded**, your program should output (1) the identifier of the node, (2) its order of expansion, (3) the value of g, h and f for that node (DLS does not require a value for h), and (4) the lists *OPEN* and *CLOSED*. The value of *Flag* will be small, so don't worry about *OPEN* and *CLOSED* being too long. **Failure to print *OPEN* and *CLOSED* will result in loss of marks.**
    * Each time a node is **generated**, you should output (1) the operator that was applied, (2) the identifier of the node, (3) the parent of the node, (4) the cost g of reaching that node, (5) a heuristic value h, and (6) a value $f = g + h$.
      IMPORTANT: If you implement Graphsearch (as opposed to Treesearch), you need to identify when a repeated instance of a node is reached, and use only the first identifier created for this node.

Submission Requirements

- You are allowed to work with **one** other student if you wish. Beyond that, make sure you haven't shared your work with other students.
  IMPORTANT: We use a program that reliably detects cheating. Make sure that your work is your own, and that you have not shared your work with other people. If cheating is detected, both the giver and the receiver are deemed to be at fault.
- Your code should have adequate in-line documentation.
- Demonstrate that you have tested your program on the given puzzle, and on at least four additional puzzles that you have devised. **Inadequate program testing will result in loss of marks.**
- Prepare a brief report (1-2 pages) that includes
  - a justification for your heuristic function for algorithm A, and for the bound and any heuristics you have used for the Backtrack algorithm, and a discussion of any implementation issues that are not straightforward. Indicate whether your h function is admissible and/or monotonic. Is the resulting Graphsearch algorithm A or A*?
  - a brief analysis of the results obtained by the various algorithms on your sample puzzles, e.g., run time, quality of the results (did they find the optimal path?).
- If you have done the work in pairs, prepare a very brief peer assessment report (1 paragraph) that includes who is responsible for what parts of the assignment (e.g., "Fred programmed Graphsearch and I programmed Backtrack"), and whether you are satisfied with your partner's contribution.
  IMPORTANT: Be prepared to be interviewed about the assignment, particularly about the parts that you did **not** program. Failure to exhibit adequate knowledge during the interview will result in loss of marks.
- Prepare your submission as follows:
  - Submit on moodle your source code, test puzzles and report in a zip file named FIT3080-*StudentID*-A1.zip, where *StudentID* is your Student ID number. **Assignments that don't follow this naming convention will be rejected.**

  - Make sure to include all the source files along with the executable file of your assignment. All the code, puzzles and executables should be **in the top directory** (no sub-directories).
  - If you have done the work in pairs, **use only the ID of one student** in your submission, but **put both names and ID numbers** in the report. In addition, **both students must complete the electronic Assignment Cover Sheet and the Peer Assessment**.
  - Hand in a hard copy of your report into the assignment box in the foyer of building 63, 25 Exhibition Walk (if you have done the work in pairs, please hand in **one** report with both names).

Assessment

- Backtrack: 4 marks.
- Graphsearch: 8 marks (heuristic 0.5 marks).
- Report, including appropriate demonstration of performance with evidence of testing (i.e., at least five puzzles): 3 marks.
  Note: We are not marking the quality of your code, but marks will be deducted for inadequate code and inadequate comments.
- **Bonus Marks:** if you implement the pointer redirection step of Graphsearch you will receive 2 bonus marks. **Indicate clearly in your submission whether you have done this.**
- **Late submission policy:**
  10 marks will be deducted for every work day a submission is late, and every time we need to contact you because we can't run your submission. **To avoid any penalties, make sure your submission runs in the labs before the due date.**