Introduction to AI NCTU Spr2018 Programming Assignment #1 Due 4/2/2018

The objective of this assignment is for you to practice with the classical search algorithms, both uninformed and informed ones.

In this assignment, the goal is to move from point (0,0) to a target point (*X*,*Y*), where *X* and *Y* are integers. A sequence of integers in the range of 0~9 are provided as part of the problem. In each step, the program takes the next integer in the sequence and select an operation to move to a new point. There are five operations allowed (i.e., the branching factor *b* for the search tree is 5):

(*x*+) add to *x*,
(*y*+) add to *y*,
(*x*–) subtract from *x*,
(*y*–) subtract from *y*,
(S) skip this integer and do not move.

Here is an example:

(*X*,*Y*) = (22, 9)
The given sequence of integers is 5 3 9 7 4 2 1 3 5 6 2 8 1 3 7 7 4 4 2 9 8 6 1 7 2 5 5 4

A possible solution (not necessarily optimal), with the position after each operation, is given below:

initial	(0,0)	
(*x*+) 5	(5,0)	
(*y*+) 3	(5,3)	
(*x*+) 9	(14,3)	
(*y*+) 7	(14,10)	
(S) 4	(14,10)	
(*x*+) 2	(16,10)	
(*y*–) 1	(16,9)	
(*x*+) 3	(19,9)	
(*x*+) 5	(24,9)	
(S) 6	(24,9)	
(*x*–) 2	(22,9)	Goal

You should implement and experiment with three searching strategies: BFS, IDS, and A*. For A*, a simple heuristic function is provided below for you to try. For IDS, you need to implement depth-limited search (DLS) first, and then use a loop to call DLS with increasing depth limits.

The problem of this assignment has the benefit that there is no difference between tree-search and graph-search, so you do not need to worry about the explored set or the need to check repeated states.

The test data will be provided as a text file. Each line represents a test set, in the format below:

```
BFS    22 9    5 3 9 7 4 2 1 3 5 6 2 8 1 3 7 7 4 4 2 9 8 6 1 7 2 5 5 4
```

Each line starts with a string, which is the strategy to test. The first two numbers are *X* and *Y*, and the following numbers are the sequence of integers. For output, your program should print out the list of operations as well as the position after each operation, in the same format as the example above.

You submission is a report file in Word or PDF format. The report (maximum 4 pages single-spaced) should describe your experiments and results, especially the comparison between the different algorithms. In your report, also include a section describing your observations, interpretations, things you have learned, remaining questions, and ideas of future investigation. Include your program code as an appendix (not counting toward the 4-page limit), starting from a separate page.

Heuristic function: A simple heuristic function to use is

$h(n)$ = floor($|d_x|$/9)+floor($|d_y|$/9),

where (d_x,d_y) is the vector from the location at node n to the goal, and floor(v) is the largest integer equal to or less than v.

We have talked in the class about how to design heuristic functions by relaxing some constraints on the problem. This topic is also mentioned in the textbook. In you report, try to answer the question: What constraint is relaxed to obtain the heuristic function given above?

Extra credit: You can try to devise your own heuristic function that is better than this one, and compare the performance.

The submission is to be through e3. Late submission is accepted for up to a week, with a 5% deduction per day.

You can use C/C++, Java, Python, or MATLAB to write your program. In general, the TAs will not actually compile or run your programs. The code listing is used to understand your thoughts during your implementation, and to find problems if your results look strange. Therefore, the code listing should be well-organized and contain comments that help the readers understand your code; this will also affect your grade.

Some notes about implementations if you are using C++ with STL:

- For BFS, `std::deque` template is the convenient choice for the frontier.
- For DLS, you can use either recursion or a stack. Either way, a node needs to remember its depth.
- For priority queue, you have some convenient choices:
  - `std::priority_queue` template: You need to supply a function for ordering two nodes.
  - `std::multimap` template: You can use your "priority" as the key.