CS 411 AI
Project 1 (Basic search algorithms)
Due: 12:00 p.m. of Sep 29th, 2010 (two weeks)

You are required to implement Breadth-first search (BFS) and Deapth-first search (DFS) to solve the 15-puzzle problems.

Implementation Requirements:

(1) Language can be either JAVA or C++ (JAVA preferred). You are required to upload both the source code and the executable file. For the executable file, it should be able to run from the command line with the following format (take BFS as an example):

```
java bfs input.txt output.txt
```

"bfs" is the name of the executable code;
"input.txt" is the input file;
"output.txt" is the output file.
In other words, the code has two parameters: one is the location of the input file and the other is the location of the output file.

- Input format:

It contains only 4 lines and each line corresponds to one line in the puzzle. And the blank is represented as "X". For example, the puzzle in Fig 1 is represented as:

1 2 5 X
6 3 8 7
10 11 12 13
15 4 14 9

1	2	5	
6	3	8	7
10	11	12	13
15	4	14	9

Fig 1 puzzle example

- Output format:

It contains multiple lines and one action per line. For example, it can be:
right
left
left
right
up
down

Examples of the input file and output file are also attached in the email.

(2) You are required to implement BFS based on the pseudo code in Figure 3.11 of page 82; and implement DFS based on the pseudo code in Figure 3.17 of page 88 (a depth-limited version, set the depth-limit to be 15).
(3) You are required to implement the data structure of each node as in Figure 3.10 of page 79. In other words, each node class should contain the state, the action, the parth-cost and the parent node.

Submission Requirements:

(1) Submitted through blackboard.
(2) Pack your files into one file with the file name as: yourname.tar. For example, "Xiaoxiao Shi.tar"
(3) The following files should be included in the package:
    (a) Source code, packed in one file with the name "source.tar"
    (b) Executable code, with the name "dfs" and "bfs"
    (c) A report file which compares the running time and memory used in different situations (You are required to design the test examples by yourself). For example:

Steps	DFS (time sec)	BFS (time sec)	DFS(memory)	BFS(memory)
2	0	0	455	755
4	**16**	**0**	**455**	**4552**
6	62	0	455	4554
8	79	0	455	4550
10	953	0	455	4550
12	11312	0	455	45511
13	64422	15	455	45532
14	133844	15	455	45563
15	290750	16	455	4555554
20	333333	Out of memory	455	Out of memory

This table reads, (take the second line as an example), for a puzzle which takes 4 steps to reach the solution, dfs takes 16 seconds and BFS takes less than 1 second to get the result, and the memory used (e.g., number of heaps) of DFS is 455 while BFS is 4552.

Evaluation:

Scoring will be based on three parts:

(1) Can give the correct solutions; that is, pass some test examples (will not release before deadline, don't ask the TA for the testing examples) 50%
(2) Can implement the pseudo code and the data structure correctly 20%
(3) Report 30%

Note: any executable file that cannot read the input file correctly or generate the output file correctly will get 0 score in the first part. Duplicate and near duplicate codes (checked by software) will all assign 0 score in the whole project!

Tips: How to output running time and memory used in JAVA (you have to search how to output running time and memory used in C++ by yourself if you want):

1. **Running time:**

```
long a = System.currentTimeMillis();
BFS bfs = new BFS();
bfs.search();
long b =System.currentTimeMillis();
System.out.println("Time in millisecond:");
System.out.println(b-a);
```

2. **Memory used:**

In your class, e.g., BFS, you have to define a function called usedMemory() and invoke the function after finishing your algorithm:

```
public Class BFS{
    private static final Runtime s_runtime = Runtime.getRuntime ();
    private static long usedMemory ()  {
        return s_runtime.totalMemory () - s_runtime.freeMemory ();
     }

    public static void main(String [] args){
        BFS bfs = new BFS();
        ……
        System.out.ptintln(usedMemory());
    }
}
```