CMPT 355 – Introduction to Artificial Intelligence
Project 1-Single Agent Search

Due:

- **Final report, code and team participation - February 25th, by midnight**
- **Mini report – February 4th by midnight**
- **Demo of functional application – February 15th during the lecture/lab**

Weight: 20% of the final mark
Type: Electronic
Work: Team project (each team has 2 or 3 students)

Objectives:

- Implement a solver for the AB puzzle, in C, C++, Java or Python.
- Write a mini report to explain your approach to solving the problem – no longer than a page
- Demo a functional application – correctly produces, and displays a required number of moves
- Write a short report – up to four pages to describe your implementation.

Submission information:

- The mini report must be typed using you favourite word processor, converted to PDF, and submitted electronically using BlackBoard.
- Submit a ZIP file and a report file (.PDF) electronically using BlackBoard.
  - The ZIP file must contain your implementation. The implementation has to be well documented. In the ZIP file use a **proper directory arrangement** and **have a make file** and **readme.txt** file about the directory structure and how to compile/execute your code. Your code must compile and execute on the `students` server.
- The ZIP file must be submitted electronically using BlackBoard.
- An e-mail detailing the individual contribution of each member. Every student must send an e-mail.

NOTE: ***Properly*** acknowledge (add a note and/or hyperlink and/or comment) any help or resource you used.

Marking scheme:

Component	**Mark**
Implementation	45
Performance on a selected set of instances	20
Mini report	5
Demo of functional application	10
Written Report	15
Compliant submission – functional ZIP, make, readme file	5
TOTAL	**100**

Description of AB (Courtesy of Air Berlin):

You are given:

- a set of n^2+1 large disks stamped with 1, 2 , 3 , or 4 .
- a set of n^2 smaller disks, of which n are stamped with 1, n are stamped with 2, n are stamped with 3, and finally n are stamped with n.

Arrange the large disks in a circle. Place the smaller disks on any of the large ones.

One large disk is still uncovered. The number stamped on it tells you which one of the small disks to move first. If the number stamped on it is, say 3, then you can move one of:

- the third smaller disk form the right
- the third smaller disk from the left
- the first smaller disk from the left
- the first smaller disk from the right

and place it on top of the uncovered large disk.

A new large disk with a new number on it has now become uncovered. This number tells you which smaller disk you can move next.

You have won the game when all of the same small disks stamped with 1 are next to each other, **followed** by all small disks stamp with 2, **and so on**, and finally followed by an uncovered large disk. Remember they are placed in a circle.

Input: Input to the program consists of two lines:

1. The first one represents the numbers stamped on the large disks.
2. The second one represents the numbers stamped on the smaller disks. A 0 means that the corresponding large disk is uncovered.

The input comes from the keyboard (standard input).

There is a one to one correspondence between the large disks and smaller disks, and the correspondence is given by the order of the corresponding disks in the lists.

Your program must accept a command line parameter which specifies the number of large disks.

You can assume that the input is correct; therefore you do not have to check the input.

Eg: The input can be

```
1 2 3 4 1 2 3 1 2 3
1 1 1 3 2 2 3 3 0 2
```

You would start the program as AB 10

Output: The output begins with "Solution is" and is followed by the sequence of states that led to the solution; one state per line. The first one is the initial state – **which must be identical to the initial state given as input. That is to say that no matter what internal representation you use for the states, the external representation MUST be the same as the given one. The order of the smaller disks must be in perfect correspondence with the initial order of the large disks. The output is sent to the screen (standard output).** If there is no solution, the output must be "No solution".

Eg: For the sample input above, the output can be something like this:

```
Solution is
1 1 1 3 2 2 3 3 0 2
1 1 1 3 2 2 0 3 3 2
1 1 1 0 2 2 3 3 3 2
1 1 1 2 2 2 3 3 3 0
```

Implementation: Using any kind of search algorithm, your assignment is to solve a given instance of the AB puzzle. This should be done with the smallest search tree possible, while still trying to get the optimal result (minimum number of moves). Enhance your search algorithm to eliminate as much of the search tree as possible using some of the standard techniques in the literature and, possibly, some of your own application-dependent enhancements. Of course, most of the magic happens in the evaluation function. Can you come up with a good evaluation? The evaluation function used MUST be described in your project.

A good way to test your program is to use positions that are a few moves away from a solution, and verify that your program finds the solution in the right number of moves. It is a good idea to try puzzles with a few disks first (5 or 10).

Specifications:

- Your code must compile/execute using the compilers/interpreters installed on the **`students`** server
- Your **makefile** MUST include a line:
  - for generating the executable file (.exe)
  - Or start Java with your code
  - Or start an interpreter with your code

Mini report: Present the search algorithm you're planning to use, ideas for the evaluation function, representation of the state space, programming language chosen. This document must be no longer than one page and must be proofread for spelling and grammar.

Report: Add a short description of the search algorithm. I want to know any search enhancements that you tried. Describe your heuristic function. Tell me what you did that was interesting. What problems did you encounter? How can your solution be improved? This document must be no longer than four pages and must be proofread for spelling and grammar.

Demo: You must demo a working application which can generate a required number of moves – they don't have to be the moves that produce a solution. For every move, your demo must also show all the possible choices.

Individual marks: Your marks will have two components: a group component – worth 50% of your project mark and an individual component – worth 50% of your project mark. The group component is based on the submitted project mark. The individual component is based on the work split evaluation - which every student must submit.
NOTE: No mark can be more than 100.