Computer Science 5400
Artificial Intelligence

Spring 2022

Puzzle Assignment #4

Version 03.13

DUE DATE : Monday, March 21th, 11:59:59pm

Assignment:

Create a program that implements **Greedy Best-First Search** (**GBFS**) over the Pengu game to find a desired sequence of game actions. You can read more about GBFS here: [Link]. If you own the Norvig textbook, you can also read Section 3.5 and 3.6 for more details on GBFS. Your program's **objective** is to find a sequence of actions (for Pengu) that achieves a score of at least **20**.

Specifications:

Your program will follow the same input/output structure from the previous *Puzzle Assignments*. Two **command line arguments** indicate the **input** and **output** files. The **input** file contains a Pengu game board specification in the same format used in previous assignments. The **output** file will also follow the same format used in previous assignments although this time the first line shall contain a sequence of valid moves that if executed by Pengu in the input file initial game board achieves a score of at least **20**.

As before, your output files are what is relevant to your grade, whatever is displayed to the screen during program execution will be ignored.

Initial Game Board Preconditions:

- You can safely assume that there will be **at least one** way for Pengu to achieve the desired score from the initial input game board.

Heuristic Function:

As you are implementing **GBFS**, candidate plans for Pengu should be considered in an order determined by a **heuristic function**. This function **h(s)** should ideally take a game state **s** and return an estimate of how close **s** is to the goal. Plans that lead to game states closer to the goal should be considered most promising and be explored first. Your code for the heuristic function **h(s)** should be **clearly identified** for the graders to find and evaluate. It is **not a requirement** for this assignment that your heuristic function lead to optimal solutions, but we would take it into consideration if you show us it does.

Input File:

The Pengu specification input file will have the same format as the one from *Puzzle Assignment #1*.

Input File Example:

```
8 17
#################
#P S#* #*U#U  *S#
#   # 0#0 **0*  #
#U *S**#**S#* S*#
#  U0 ***S#* S#*#
#*#** ***# *  #*#
#U#S* #***#UU#  *#
#################
```

Output File:

The output file follows the same format specification as the one from the previous *Puzzle Assignments*. The only change is the desired target score.

Output File Example:

```
39268624932
20
###########################
#   S#* #*U#U    S#
#    # 0#0    0    #
#U   S   # *S#   S #
#   U0   * S#   S# #
#*#*       #     # #
#U#S   #**#UU#   P#
###########################

```

In the above example output file the first line is the sequence of moves executed by Pengu, the next line shows that a score of 20 is attained, and the next 8 lines display the final configurations of the game board after the execution of the shown sequence of moves

Submission:

Same submission system as before. Place all code/scripts in **this assignment**'s git repository in the course's GitLab server, [link] .

When your assignment is complete and ready, **modify** the content of the '`ReadyForGrading.txt`' file to '`Yes`'. This will indicate to the graders that your assignment is ready to be graded. Additionally, add to "`ReadyForGrading.txt`" the language of which you are coding in: **one** of the strings `c` , `c++` , `python` , `javascript` , `java` , `typescript` , `rust` , `c#`

You can implement your assignment using one of the supported programming languages, but it must be in a way compatible with the Computer Science Department's Linux machines. In order to accommodate such different languages, your submission should include a bash script named '`run.sh`' that **compiles** and **runs** your program with all the necessary options and commands.

For example, the following is a possible '`run.sh`' script for C++ 11.

```
#!/bin/bash
g++ -std=c++11 *.cpp -o hw1.ex
./hw1.ex $1 $2
```

An example '`run.sh`' script for Python3 if the program is called '`hw4.py3`' :

```
#!/bin/bash
python3 hw4.py3 $1 $2
```

An example '`run.sh`' script for Java if the program is called '`hw4.java`' :and the main class is called '`hw4`'.

```
#!/bin/bash
javac hw4.java
java hw4 $1 $2
```

Your program will be evaluated and graded using the command:

```
./run.sh gradinginput.txt gradingoutput.txt
```

IMPORTANT: Remember to make your '`run.sh`' file executable as a script with the LINUX command:

```
chmod +x run.sh
```

Grading:

Your program will be evaluated and graded on the Computer Science department's Linux machines so your program needs to be compatible with the current system, compilers and environment.

Description	Weight
Correct Actions and Final State Generation.	40%
Specified Search Algorithm.	40%
Good Programming Practices (adherence to coding standards, modular design, documentation etc.)	20%

A Primer for Puzzle Assignment #4:

Intuition:

We continue exploring the Pengu game tree, but now we explore plans by imposing into them a priority based on a heuristic function. We use a priority queue to store candidate plans. The final implementation is similar to Breadth-First Search, but instead of a Queue we use a Priority-Queue.

Implementation:

We reuse from previous *Puzzle Assignments* the **GameBoard** structure and the **TransitionFunction()**.

As a reminder:

> **GameBoard** : A "snapshot" of an Pengu game, including location of the walls, snow cells, hazards and fish, the score, the location of Pengu.
>
> **TransitionFunction** (s, p) → s'
> INPUT: s : a GameBoard
> p : A sequence of Pengu actions
> OUTPUT: s' : the GameBoard that results from executing p in s.

Refining GBFS for Pengu:

The main change from before is that the frontier is now a **priority-queue**, where sequences of actions / plans are ordered by how close they get to the final goal, as determined by a heuristic function. **Plans that we believe are closer to the goal should be explored first.**

```
PROCEDURE Pengu_Greedy_BFS
INPUT s0 : the initial GameBoard
      goal(s) : boolean function that tests if the goal is true in
                GameBoard s.
      h(s) : A heuristic function that returns an estimate of the "cost"
             from s to the goal.

OUTPUT : a sequence of actions that takes the game from s0 to a state that
         satisfies the goal.

VAR frontier : Priority-Queue of sequences of actions.
             // Smallest priority value is highest priority

BEGIN
    enqueue [] (empty sequence) into frontier
    WHILE frontier is not empty
        dequeue sequence of actions p = [a0, a1, a2, ..., ak] from frontier
```

```
        // p is the most preferred path in the frontier.
        sk ← TransitionFunction( s0, p )
        // sk is the GameBoard that results from executing p in s0

        IF goal(sk)
            RETURN p

        FOR every valid action a at sk
            px ← [a0, a1, a2, ..., ak, a]
            sx ← TransitionFunction( s0, px )
            // sx is the GameBoard that results from executing px in s0
            enqueue px into frontier with priority h(sx)
            // h() is the heuristic function
    ENDW
END.
```

This is, of course, just a first draft.

Further refinement:

You can experiment with different heuristic functions to find the one that works the best with your program. You are not required to implement pruning on this assignment, but it will help it speed up execution. Many of the components carry from previous *Puzzle Assignments*, but you need to be careful as this is a different algorithm.

Good Luck!