CS4341
Assignment #1
Due date: 9/9/16 @ 11:59 p.m.

For this assignment you will learn about basic search techniques. You will use iterative deepening and greedy (best-first) search on a set of arithmetic operators. Your goal is to start with an initial number, and use the smallest number of arithmetic operations to reach the target numeric value. In the event the value is unreachable, your program should get as close as possible to the target value in the provided time. For this assignment, “close” is measured as the absolute difference between your program’s answer and the target value.

Reading in the search problem

Your program should accept a command-line argument specifying the name of the file defining the search problem to work on. The file contains the following information, one entry per line:

- Whether to use iterative deepening (iterative) or greedy best-first search (greedy)
- A starting numeric value
- A target numeric value
- The number of seconds your program has to find the best answer possible. Your program **must** terminate and output an answer in the time allocated.
- A series of mathematical operators, followed by a numeric value
    - +, -, /, *, ^

As an example, consider a file with the following contents:

```
iterative
4
11
2.5
+ 3
- 1
/ 2
* 5
^ 2
```

For this problem, your program should use iterative deepening, start with the value of 4, and its goal is to reach a value of 11 within 2.5 seconds. Operators it is permitted to apply are adding 3 to the current value, subtracting 1, dividing by 2, multiplying by 5, or squaring the number. **All arithmetic operations have the same (unit) cost.**

What your program should output

Your program should output the operators needed to reach the answer (or get as close as possible) in the following format:

```
4 ^ 2 = 16
16 / 2 = 8
8 + 3 = 11

Error:  0
Number of steps required:  3
Search required:  0.3 seconds
Nodes expanded:  45
Maximum search depth:  3
```

Output should be displayed on the console, and follow the format shown in the example. It is in your interest to make life easy on the grader. (the example is meant to be illustrative; the exact numbers provided are probably not correct so don't worry if your program has different ones)

The search techniques you will use

You should use iterative deepening and greedy (best-first) search for this assignment.

For iterative deepening, implement it as specified in the text. Given that all arithmetic operations have the same cost, this search technique will find a solution with the minimal cost.

Greedy (best-first) search requires a heuristic function to guide the search. Your first task is to figure out what that heuristic function should be. Once you have that, use it to guide action selection with greedy search.

Analysis

You should develop a suite of test problems and run both iterative deepening and greedy (best-first) search on them. You should be sure to have at least one test problem where greedy search returns a suboptimal solution. Similarly, develop at least one test problem where iterative deepening is not able to search deep enough to get a solution but greedy search can.

In general, how did the number of steps required by greedy search compare to iterative deepening? How does the number of nodes expanded vary for the two techniques? A graph may be helpful to compare them.

Compute the effective branching factor of iterative deepening. Compare it to the effective branching factor of greedy search.

What you should hand in

- Your code
- A set of at least 5 test problems you used to compare your algorithms
- A writeup of your comparison of the relative performance of the two algorithms

Tips

- You may assume that your program will have sufficient time to search to depth 3. You could, for example, use these early runs to project the time taken for later iterations and know how deeply to search.
- You might not have time to complete the next level of the search. In this case, you should output the best answer you have found so far. Do not go over the specified time.
- As always, start simple. Have a straightforward search problem where you can hand-compute the best answer and expected statistics to be sure there aren't any glaring errors.