CSCE625: Artificial Intelligence
Programming Assignment 3 : Simplifying Mathematical Expressions via Search

Name: Satya Sreenadh Meesala
UIN: 232003265.

SEARCH ALGORITHM:

Used A* - Search Algorithms with g(n) = 0 and h(n) as mentioned below.

1) simplify
    Simplifies a given operation.
    For example : (Before : x + 3 = 3 + 4, After : x + 3 = 7)

2) solveIdentities
    Solves (sin(x)^2 + cos(x)^2 and cos(x)^2 + sin(x)^2)
    For example : (Before : sin(x)^2 + cos(x)^2 + y = 2 , After : 1 + y = 2)

3) squarIt
    Solves (sqrt(x) = equation)
    For example : (Before : sqrt(x) = 2 + 3 , After : x = (2 + 3) ^ 2)

4) unLogIt
    Solves (log(x) = equation)
    For example : (Before : log(x) = 2 + 3 , After : x = 10 ^ (2 + 3))

5) unLnIt
    Solves (ln(x) = equation)
    For example : (Before : ln(x) = 2 + 3 , After : x = e ^ (2 + 3))

6) inverseIdentity
    It takes the inverse of an operation and takes the operand to the other side of the equation. For example : (Before : x + 3 = 4 , After : x = 4 - 3)

7) commutative
    Gives commutative of two operands
    For example : (Before : x + 3 = 4 , After : 3 + x = 4)

HEURISTIC:

h = (2*findOperations(x) + findDepthOfX(x,self.variable) + ifXInLeft(x,self.variable) + ifXAtLeft(x,self.variable) + ifIdentityLeft(x))

where,
findOperations(x) : Function to find out number of operations left in equation
findDepthOfX(x, v) : Function to find depth of variable v
ifXInLeft(x, v) : Function to check if v in left subtree
ifXAtLeft(x, v) : Function to check if v is left child of root
ifIdentityLeft(x) : Function to check if identities left in the tree.

Rationale behind this heuristic:
1) It is good to have less number of operations in the equation to reach a solution faster hence "findOperations(x)" is being used. I have multiplied it by 2 to give it more priority.

2) We need variable 'v' at less depth in the tree to get it solved, so "findDepthOfX(x, v)" is being used.

3) We need variable 'v' in the left subtree of '=' in the equation. Hence "ifXInLeft(x, v)" is being used.

4) We need the variable 'v' on the left side of '=' in the equation. Hence "ifXAtLeft(x, v)" is being used.

5) We don't want the identity sin^2 + cos^2 in the equation. Hence "ifIdentityLeft(x)" is being used.

So the node in the frontier with the minimum value of the combination of above mentioned things will be selected next.

OUTPUTS:

1) eq>x=(5+6)*(2^2)
    var>x
    Produces Output: x = 44

2) eq>x=(7*11)/(1-1)
    var>x
    Produces output: x = undefined

3) eq>6*x*1*y*6*z*7*6=200
    var>z
    Produces Output: z = 33.3333333333 / 6 * x * 1 * y * 6 / 7

4) eq>x=sin(y)^2+cos(y)^2+z
    var>x
    Produces Output: x = (z + 1)

5) eq>sin(x)^2 + cos(x)^2 + x + 2+19 = 54
    var>x
    Produces Output: x = 32

6) eq>sqrt(x)=(1+3-2)*(2^4)
    var>x
    Produces Output: x = 1024

7) eq>sqrt(x)=(1+3-2)*(2^6) + sqrt(y)
    var>y
    Produces Output: y = sqrt(x) - 128^2

8) eq>log(x)+11*2-19=sin(x)^2+cos(x)^2+62
    var>x
    Produces Output: x =
10000000000000000000000000000000000000000000000000000000000000000

9) eq>ln(x)*2=2*3*4/3
    var>x
    Produces Output: x = e^4

10) eq>sin(x)^2 + cos(x)^2 + ln(y) + 18 = 21 + log(x)
    var>y
    Produces Output: y =e^21 + log(x) - 18 - 1

RUNNING THE PROGRAM:

1) Install ply. (pip uninstall ply; pip uninstall pyhcl; pip install ply; pip install pyhcl)
2) Run main.py
3) Input the equation after the prompt “eq>”
4) Input the variable for which the equation needs to be solved for after prompt “var>”

BRIEF NOTES AND LIMITATIONS

1) Used python 2.7.5 due to issues with insort functionality of Python 3.
2) Multiple instances of the variable for which equation needs to be solved is not handled (eg. : “2x + 3 = 4 + x“ this is not handled.
3) Calculus is not handled

USED RESOURCES:

1) https://code.google.com/archive/p/aima-python/
2) http://robotics.cs.tamu.edu/dshell/cs625/asgn3/equationparser-0.1.tar.gz
3) https://docs.python.org for finding out usage of inbuilt libraries like operators.

APPENDIX:
Entire code can be found at https://github.com/mssreenadh/eq-simplifier-main
main.py

```
import search
import eqparser
import helperfns

s=input("eq>")
v=input("var>")
p = eqparser.parse(s)
pbm = search.equationSolver(initial=p,variable=v)

print(pbm.astar_search(pbm))
```

search.py

```
def best_first_graph_search(self, problem, f):
        """Search the nodes with the lowest f scores first.
        You specify the function f(node) that you want to minimize; for example,
        if f is a heuristic estimate to the goal, then we have greedy best
        first search; if f is node.depth then we have breadth-first search.
        There is a subtlety: the line "f = memoize(f, 'f')" means that the f
        values will be cached on the nodes as they are computed. So after doing
        a best first search you can examine the f values of the path returned."""
        global currentstate
        f = memoize(f, 'f')
        node = Node(problem.initial)
        if problem.goal_test(node.state):
            return node
```

```
    frontier = PriorityQueue(min, f)
    frontier.append(node)
    explored = set()
    while frontier:
        node = frontier.pop()
        currentstate = node.state
        #print node
        if problem.goal_test(node.state):
            return node
        explored.add(node.state)
        for child in node.expand(problem):
            if child.state not in explored and child not in frontier:
                #print child
                frontier.append(child)
            elif child in frontier:
                incumbent = frontier[child]
                if f(child) < f(incumbent):
                    del frontier[incumbent]
                    frontier.append(child)
    return None
```

utils.py

```
class PriorityQueue(Queue):
  """A queue in which the minimum (or maximum) element (as determined by f and
  order) is returned first. If order is min, the item with minimum f(x) is
  returned first; if order is max, then it is the item with maximum f(x).
  Also supports dict-like lookup."""
  def __init__(self, order=min, f=lambda x: x):
      update(self, A=[], order=order, f=f)
  def append(self, item):
      bisect.insort(self.A,(self.f(item),item))
      print(self.A)
      print(self.f(item))
      print(item)
  def pop(self):
      if self.order == min:
          return self.A.pop(0)[1]
      else:
          return self.A.pop()[1]
```

```
  def __contains__(self, item):
      return some(lambda __x: __x[1] == item, self.A)
  def __getitem__(self, key):
      for _, item in self.A:
          if item == key:
              return item
  def __delitem__(self, key):
      for i, (value, item) in enumerate(self.A):
          if item == key:
              self.A.pop(i)
              return
```