perform breadth first search from every element, not expanding any nodes with value higher than its parent.

At the end of each path, mark it as P, A, or F.

When unwinding from all the paths from breadth first search, mark each parent with the same value as its child, unless the parent has already been marked P or A, in which case, if the child is marked with the opposite value, mark as T. If the child was marked T, mark its parent as T.

perform breadth-first search from the center element.

when retracing steps from the ends of the search, mark elements as P or A if returning from P or A respectively if the parent node has a higher value than the previous one

consider each element of the matrix in order

If it has already been marked "true" or "false", skip it.

if it touches an element of less or equal value that is marked true, mark it "true" and add its indices to the list

if all of its neighbors either are of higher value or are marked "false," mark it as false

if it touches an element of less or equal value that goes to P or A or itself touches P or A.

If the element has not been marked true or false, perform depth first search. If an element marked P, A, T, or F is encountered, begin backtracking, and:

when returning from a node that is marked P or A, if a node that is backtracked to:

has the opposite designation and has a higher value, mark it as true

has a higher value, mark it as P or A accordingly

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.children = []

class naryTree:

def \_\_init\_\_(self, node):

self.root = node

def compare(str1, str2):

if str1 == str2:

return True

else:

return False

root = Node("<html>", None)

tree1 = naryTree(root)

tree2 = naryTree(root)

def getStr(tree):

ans = ""

stack = []

stack.append(root)

while stack:

curr = stack.pop()

if not curr.children and curr.data[0] != '<':

ans += curr.data

else:

for child in reversed(children):

stack.append(child)

str1 = getStr(tree1)

str2 = getStr(tree2)

print compare(str1, str2)

12345678901234567890

how are you how are

you how are you how

are you how are you 5 per pattern

how are you how are

you how are you how

8

3 lines per pattern

15/3 = 5

5 \* 5 = 25

Given a text with N words a1..aN, and a very large textual screen of width C chars and height L lines (C, L are much larger than N). How many times the text would fully fit on the screen if words cannot be broken?

def numFit(text, C, L, N):

times\_per\_pattern = 1

space\_taken = 0

pattern\_length = 0

for j in xrange(L):

space\_left = C - space\_taken

if space\_taken != 0:

times\_per\_pattern += 1

fit\_times += space\_left / N

rem = space\_left % N

if rem == 0:

pattern\_length = j

break

i = N - rem

while i >= 0 and text[i] != ' ':

i -= 1

space\_taken = C - i

return times\_per\_pattern \* pattern\_length