CSC1001 Tutorial 11

Recursion

Frederick Khasanto (122040014)

16 November 2023

Definition

A technique where a function makes one or more calls of itself during execution

Structure

1. Base case
2. Original problem
3. Subproblem

Factorial

$$n! = \begin{Bmatrix} 1 & \text{if } n = 0 \\ n(n-1)\ldots 2.1 & \text{if } n \geq 1 \end{Bmatrix}$$

--- Recursive Definition

$$n! = \begin{Bmatrix} 1 & \text{if } n = 0 \\ n(n-1)! & \text{if } n \geq 1 \end{Bmatrix}$$

```python
# Factorial
def fact(n):
  if n == 0:
    return 1
  else:
    return n*fact(n-1)

print(fact(5))
```

```
120
```

Recursive Call Tree

Consists of boxes and directed edges.

Each box represents a function call and is labeled with the name of the function and the actual argument passed to the function when it was called.

Directed edges between the boxes indicate the flow of execution

- Solid edges (-----) indicate the function from which a call originated.
- Dashed edges (- - -) indicate function returns and are labeled with the **return value** if a value is returned to the caller.

The edges are listed **left to right** in the order the calls are made.

Practice Questions

- For recursive call trees, please check the PDF titled "RecursiveCallTree-Tutorial 11.pdf" in the Onedrive folder.

Q1: Number system conversion

i) Write a program that define a recursive function `def conv(n, base)` that can print out the binary (base=2) or octal (base=8) version of the decimal number n.

ii) Modify the program a little bit to provide a way for solving problem Q2 in Assignment 1. What about the reverse order?

```
# i) NumberConversion
def conv(n,base):
    if n//base==0:
        print(n,end='')
    else:
        conv(n//base,base)
        print(n%base,end='')

print('In binary system , 12=')
conv(12,2)    #should be 1100
print()
print('In octal system , 20=')
conv(20,8)    #should be 24
```

```
In binary system , 12=
1100
In octal system , 20=
24
```

```
# ii) PrintDigit
def PrintDigit(n):
    if n//10==0:
        print(n)
    else:
        PrintDigit(n//10)
        print(n%10)

PrintDigit(3125)
```

```
3
1
```

```
2
5

# ii) PrintDigit reverse
def PrintDigit(n):
    if n//10==0:
        print(n)
    else:
        print(n%10)
        PrintDigit(n//10)

PrintDigit(3125)

5
2
1
3
```

Q2: Permutations of a string

Write a recursive program to print out all the possible pemutations given a string with distinct characters.

```
#uses a loop to move a character from s2 to s1 and
#recursively invokes it with a new s1 and s2.
#The base case is that s2 is empty and prints s1 to the console.
def displayPermutationHelper(s1,s2):
    if s2 == "":
        print(s1)
    else:
        for i in s2:
            index=s2.index(i)
            displayPermutationHelper(s1+i,s2[:index]+s2[index+1:])

def displayPermutation(s):
    displayPermutationHelper("",s)

def main():
    s=input("Enter a string:")
    displayPermutation(s)

main()

Enter a string:abc
abc
acb
bac
bca
cab
cba
```

Q3: Judge Palindrome

Write a recursive program to define a function to judge whether it is a palindrome or not. If a string is equal to its reverse, then it is a palindrome. For example, "*level*" and "*madam*" are palindromes.

```
def check(string):
    if string=='':
        return True
    elif string[0]==string[len(string)-1] and 
check(string[1:len(string)-1]):
        return True
    else:
        return False

def main():
    string=input('Enter a string:')
    print(check(string))

main()

Enter a string:level
True
```

Q4: Rearrange a list

Write a recursive algorithm to rearrange a list of integer values so that all the even values appear before all the odd values.

```
def rearrange(lst):
    if len(lst)==0:
        return []
    #If the first element is odd,rearrange the sublist after this 
element,
    #and then insert it to position at the end of all even numbers,
    #to avoid changing the original order of odd numbers.
    elif lst[0]%2==1:
        L0=lst[0]
        lst=rearrange(lst[1:len(lst)])
        index=0
        while index<len(lst):
            if lst[index]%2==1:
                lst.insert(index,L0)
                break
            index+=1
        if index==len(lst):
            lst.insert(index,L0)
        return lst
    else:
        lst=[lst[0]]+rearrange(lst[1:len(lst)])
        return lst

def main():
    lst=[1,4,3,2]
    print(rearrange(lst))

main()
```

```
[4, 2, 1, 3]
```

Q5: Find all possible subsets

Write a recursive algorithm to find all the possible subsets of a list (excluding itself).

```
def findSubset(L,T):
    if len(L)==0:
        return []
    for i in range(len(L)):
        subset=list()
        for element in L:
            if element !=L[i]:
                #Find all the possible true subsets with only one
element excluded
                subset.append(element)
        if subset not in T:
            findSubset(subset,T)
            T.append(subset)
    return T

L=[1,2,3]
Output=findSubset(L,[])
print(Output)
```

```
[[], [3], [2], [2, 3], [1], [1, 3], [1, 2]]
```

Q6: Tic-tac-toe

This is the simplest example of how Artificial Intelligence may work. You may explore more if you are interested in this field. Please refer to the codes in the Onedrive folder: `Board.py`, `Player.py`, `6-Game.py`.