Dynamic programming

July 10, 2022

https://www.youtube.com/watch?v=oBt53YbR9Kk

memoization

write a function that takes the position n and returns the nth fib no basic wrong/inneficient implementation

the below program has a time complexity of O(2^n)

the space complexity is O(n)

because he function calls happen only one at a time in LNR

[ ]:
```
# basic wrong/inneficient implementation


def fib(n):
    if n <= 2:
        return 1
    return fib(n - 1) + fib(n - 2)


print(fib(6))
print(fib(7))
print(fib(8))
print(fib(9))
# fib(50)  takes forever to compute dont even try
# this takes 2^50 steps
```

```
8
13
21
34
```

better implementaion using memoization

time complexity O(2n) -> O(n)

space complexity -> O(n)

[ ]:
```python
def fib(n, memo={}):
    if n in memo:
        return memo[n]
    if n <= 2:
        return 1
    memo[n] = fib(n - 1, memo) + fib(n - 2, memo)
    return memo[n]


print(fib(2009))

"""
dict = {}
print(fib(50,dict))

print(dict)
"""
```

```
3211324998268158284523847203209708620740607826436921382820726973887352837216465813601607624368278410255618945484808988900205426234968094236646697230669809316249087757247394023948814830096114330544153482671776403312226728356316295649078585843078505084761515748916701170477392061781243206175561976990874233674141811052493007153851287942959207659531735086437334610990020876787656487609216126565715584580255100846170091289 09
```

[ ]:
```
'\ndict = {}\nprint(fib(50,dict))\n\nprint(dict)\n'
```

find the number of ways you can travel corner to corner in m*n grid provided you can only move right and down

finction takes in m and n rows and columns time complexity = O(2^n+m) space complexity = O(n+m)

[ ]:
```python
def gridtravel(m, n, memo={}):

    if m == 1 and n == 1:
        return 1
    elif m == 0 or n == 0:
        return 0

    return gridtravel(m - 1, n) + gridtravel(m, n - 1)


gridtravel(11, 11)
```

[ ]:
```
184756
```

time complexity m*n space complexity n+m

[ ]:
```
dict = {}
# joels better code
def gridtravel(m, n, memo={}):
    l = tuple(sorted((m, n)))
    if l in memo:
        return memo[l]
    elif 1 in l:
        return 1
    else:
        memo[l] = gridtravel(m - 1, n, memo) + gridtravel(m, n - 1, memo)
        return memo[l]


gridtravel(1001, 1001)
```

[ ]:
```
204815162698948971433516250298082504439642488798139703382038263767174818620208375582893299418261020620146476631999802369241548179800452479201804754976926157856301289663432064714851152395251651227768588611539546256147907378668464154444533617613770073855673814589630071306510455959514479888746206368718514551828551173166276253663773084682932255389049743859481431755030783796444370810085163724827462791417016619883764840843541430817785947037746565188475514680749694674923803033101818723298009668567458560252549910118113525353465888794196665367490451130611009631190627034250229315591110897673396399114912 0
```

memoization recipe

1. make it work (with recursion look for correctness)(this is harder than step 2) visualize as trees implement the tree using recursion test it
2. make it efficient add a memo object (hash object/dict) add a base case to return memo values store the return values into the memo

0.0.1 cansum()

cansum(targetsum, numbers)

eg, cansum(7,[5,3,4,7]) -> true

```
3 + 4
```

```
7
```

cansum(7,[2,4]) -> false

[ ]:
```
# joels noobie implementation


def cansum(m, n, d={}):
    j = False
```

```
    if m in d:
        return d[m]
    if m == 0:
        return True
    if m < 0:
        return False
    for i in n:
        j = j or cansum(m - i, n, d)
        d[m] = j
    return d[m]


cansum(7, [5, 3, 4, 7])
```

[ ]: True

time = n^m space = m

where n is lenghth of array and m is target sum

with memo time = n*m

[ ]:
```
def cansum(m, n, d={}):
    if m in d:
        return d[m]
    if m == 0:
        return True
    if m < 0:
        return False
    for i in n:
        if cansum(m - i, n, d) == True:
            d[m] = True
            return True
    d[m] = False
    return False


cansum(1071, [33, 4])
```

[ ]: True

howsum() return array that add up to target sum with elements from the given array

without memo time = n^m * m space = m

[ ]:
```
def howsum(m, n, d={}):
    if m == 0:
        return []
    if m < 0:
```

```
        return None

    for i in n:
        if howsum(m - i, n) != None:
            return howsum(m - i, n) + [i]
    return None


print(howsum(300, [2, 4]))
```

with a memo time = n*m^2 space = m

[ ]:
```
def howsum(m, n, d={}):
    if m in d:
        return d[m]
    if m == 0:
        return []
    if m < 0:
        return None

    for i in n:
        if howsum(m - i, n, d) != None:
            d[m] = howsum(m - i, n, d) + [i]
            return d[m]
    d[m] = None
    return None


print(howsum(97, [2, 3, 30, 80, 99]))
```

```
[2, 3, 3, 3, 3, 3, 80]
```

bestsum() return array with shortest combination of numbers that adds up to target sum

without memo

timw = n^m * m space = m * m = m^2

with memo

time = m* m *n space = m^2

[ ]:
```
def bestsum(m, n, d={}):
    if m in d:
        return d[m]
    if m == 0:
        return []
    if m < 0:
        return None
```

```python
    shortest = None
    for i in n:
        if bestsum(m - i, n, d) != None:
            c = bestsum(m - i, n, d) + [i]
            if (shortest == None) or (len(c) < len(shortest)):
                shortest = c

    d[m] = shortest
    return shortest


print(bestsum(100, [2, 33, 70, 1]))
```

```
[1, 33, 33, 33]
```

canconstruct(target,wordbank) accepts a target string and array of strings

should return a boolean indicating whether or not the target can be constructed by concatenating the elements of the word bank

you may reuse elements from the word bank

without memoization time complexity = n^m *m (slicinf gives an extra* m) space complexity = m^2

where m = target length n = wordbakn length

[ ]:
```python
def canconstruct(m, n):
    if m == "":
        return True

    for i in n:
        if i in m:
            if m.index(i) == 0:
                if canconstruct(m[len(i) :], n) == True:
                    return True

    return False


print(canconstruct("abcdef", ["ab", "def", "cd", "cdef"]))

print(
    canconstruct(
        "eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee\
  eeeeeeeeeeeef",
        ["e", "ee", "eee", "eeee", "eeeee", "eeeeee", "eeeeeee"],
    )
)
```

with memo
m = target length
n = wordbakn length

```
[ ]: def canconstruct(m, n, d={}):
         if m in d:
             return d[m]
         if m == "":
             return True

         for i in n:
             if i in m:
                 if m.index(i) == 0:
                     if canconstruct(m[len(i) :], n, d) == True:
                         d[m] = True
         d[m] = False
         return False


     print(canconstruct("abcdef", ["ab", "def", "cd", "cdef"]))

     print(
         canconstruct(
             "eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee\
       eeeeeeeeeeeef",
             ["e", "ee", "eee", "eeee", "eeeee", "eeeeee", "eeeeeee"],
         )
     )
```

```
False
False
```

0.0.2 countconstruct()

find the total nuber of ways the target string can be constructed

```
[ ]: def countconstruct(m, n, d={}):
         if m in d:
             return d[m]
         if m == "":
             return 1

         c = 0
         for i in n:
             if i in m:
                 if m.index(i) == 0:
                     c += countconstruct(m[len(i) :], n, d)
```

```
    d[m] = c
    return c


print(countconstruct("abcdef", ["ab", "def", "cd", "cdef"]))

print(
    countconstruct(
        "eeeefeeee\
eeef",
        ["e", "ee", "eee", "eeee", "eeeee", "eeeeee", "eeeeeee", "f"],
    )
)
```

```
1
256
```

0.0.3 allconstruct()

return a 2d array containing all the possible ways the string cant be constructed

blah bla random test code snippets below

[ ]:
```
y = 3
x = 5
print(x + y)
```

```
8
```

[ ]:
```
s1 = "AJYFAJYF"
s2 = "JY"
import re

if s1.startswith(s2):
    s3 = re.sub("^" + s2, "", s1)
s3
```

[ ]:
```
'YFAJYF'
```

[ ]:
```
s1 = "AJYFAJYF"
s2 = "JY"
s1.index(s2)

s2 in s1
```

[ ]:
```
True
```

[ ]:
```
# for i in range(9):
#   print(i)

item2 = "hello"

print(type(item2))


class item:
    h = 0

    def gun(self):
        pass


print("kooooooool")
item1 = item()

item1.name = "joel"


print(type(item1))
print(type(item1.gun))
```

```
<class 'str'>
kooooooool
<class '__main__.item'>
<class 'method'>
```

[ ]:
```
h = "hello"


v = "varghese"
print(h, "joel " + v)
```

```
hello joel varghese
```

[ ]:
```
h = "hello"


v = "varghese"
print(h, "joel " + v)
```

```
hello joel varghese
```