Khrystian Clark

CS 325, Winter 2022

Assignment: Backtracking & Greedy Algorithms

1. **Implement a backtracking algorithm**
   Given a collection of amount values (A) and a target sum (S), find all unique combinations in A where the amount values sum up to S.
   Each amount value may be used only once in the combination.
   The solution set should not contain duplicate combinations.
   Amounts will be positive numbers.

   Example: A = [11,1,3,2,6,1,5]; Target Sum = 8
   Result = [3, 5]
   [2, 6]
   [1, 2, 5]
   [1, 1, 6]

   a. Describe a backtracking algorithm to solve this problem.
      - Sort array A in ascending order
      - Create holders for counter and the result list
      - Create a storage database to store the already printed values.
      - Find pairs or sets that have a sum of the target, within the array, A.
        - Return the values in an array
        - Search for other combinations until the A list is exhausted
          - Create sum_holder value that holds the updated sum for comparison with the target
          - Create a value holder for the length of in order to iterate.
          - Create an empty array to store values for printing
            - Resets after the result is printed to create another result.

   b. Implement the solution in a function **amount(A, S)**. Name your file **Amount.py**

```
def amount(A, S):
    result = []
    count = len(A)+1
    s_holder = 0
    A.sort()
    return(amount_helper(A, S, result, count, s_holder))

def amount_helper(A, S, result, count, s_holder):
    stored = []
    # of the target sum is reached
    if (S == s_holder) and (result.sort() not in stored):
        print(result)
        stored.append(result.sort())
        return
    for i in range(0, len(A), 1):
        # if the tracking sum is greater than the target, reset.
        if (s_holder + A[i] > S):
            continue
        if (i > count) and (A[i] == A[i-1]):
            continue

        result.append(A[i]) # add the value to the result
        amount_helper(A, S, result, i-1, s_holder+A[i]) #iterate
        result.remove(result[len(result) - 1]) # pop the end

A = [11,1,3,2,6,1,5]
S = 8
amount(A, S)
```

a. What is the time complexity of your implementation, you may find time complexity in detailed or state whether it is linear/polynomial/exponential. etc.?
   - I believe it is exponential because of the rate of growth is based on the amount of values in the array.
   - O(2^n)

30. Implement a Greedy algorithm

You are a pet store owner and you own few dogs. Each dog has a specific hunger level given by array hunger_level [1..n] (ith dog has hunger level of hunger_level [i]). You have couple of dog biscuits of size given by biscuit_size [1…m]. Your goal to satisfy maximum number of hungry dogs. You need to find the number of dogs we can satisfy.
If a dog has hunger hunger_level[i], it can be satisfied only by taking a biscuit of size biscuit_size [j] >= hunger_level [i] (i.e biscuit size should be greater than or equal to hunger level to satisfy a dog.)
Conditions:
You cannot give same biscuit to two dogs.
Each dog can get only one biscuit.
Example 1:

Input: hunger_level[1,2,3], biscuit_size[1,1]
Output: 1
Explanation: Only one dog with hunger level of 1 can be satisfied with one cookie of size 1.

Example 2:

Input: hunger_level[2, 1], biscuit_size[1,3,2]
Output: 2
Explanation: Two dogs can be satisfied. The biscuit sizes are big enough to satisfy the hunger level of both the dogs.

a. Describe a greedy algorithm to solve this problem
   - First try to find dogs hunger that match the biscuit satisfaction level, and eliminate those values, if possible.
   - Then, feed the least hungry dog first with the least of amount of biscuits, in order to satisfy
   - Sort the arrays
     - If we have matches in the arrays, we increment the counter and remove those values from the arrays.
     - We go dog by dog to feed the least hungry first by iterating through the sorted hunger array and matching up and removing the values one by one until the biscuits are gone.
     - Return the counter.

b. Write an algorithm implementing the approach. Your function signature should be **feedDog(hunger_level, biscuit_size)**; hunger_level, biscuit_size both are one dimention arrays . Name your file **FeedDog.py**

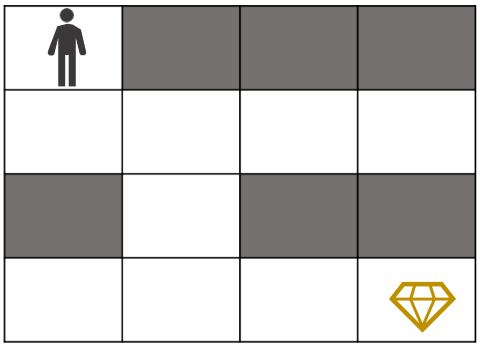
```
def feedDog(hunger_level, biscuit_size):
    count = 0 # satisfied puppies
    hunger_level.sort()
    biscuit_size.sort()
    for i in biscuit_size:
        if i in hunger_level:
            count += 1
            biscuit_size.remove(i)
            hunger_level.remove(i)
        for j in hunger_level:
            if (i >= j):
                count += 1
                i = i-j
                hunger_level.remove(j)
            if len(hunger_level) == 0:
                print (count, “satisfied pups”)
    print (count, “satisfied pups”)

feedDog([1,2,3], [1,1])
feedDog([2,1], [1,3,2])
```

a. Analyze the time complexity of the approach.
   - O(nlogn)
   - I am not fully sure of this because there is a loop within a loop. But I am going with it.

----------------------(Ungraded question: you can try this question if time permits)---------------------

You are given a puzzle in the form of a nxn matrix. Your location is in the start of a matrix at top left corner (location [0][0]) location and there is treasure location at the destination of the matrix at the bottom right corner of the matrix (location [n-1][n-1]). You can only move left/right or up/down in the puzzle. Your goals is to find out if you can reach the treasure or not.
Matrix is marked with 1 where there is a path and with 0 where is a wall.
For example, this matrix represents below shown puzzle.
Matrix: [[1, 0, 0,0],[1, 1, 1, 1], [0, 1, 0, 0], [1, 1, 1,1]]



a. Describe a backtracking algorithm to solve the puzzle, you goal is to return True if you can reach the destination or return False otherwise.
b. Write the pseudocode for the algorithm that you described in a. Your function signature should be **reachTreasure (puzzle)** and it should return True/False
c. Implement the pseudocode to solve the problem. Name your file **Puzzle.py**