Computational Thinking and Problem Solving (COMP1002) and Problem Solving Methodology in Information Technology (COMP1001)

Assignment 3 Sample Solutions

1. [20 marks] We consider a coin-changing problem. For a currency with coins c_1, c_2, ..., c_N, the problem is to find the minimum number of coins needed to make x dollars of change, as well as how many coins are needed for each of c_i where $1 \leq i \leq N$, assuming there are always enough coins. For example, in Hong Kong, we have 0.1-dollar, 0.2-dollar, 0.5-dollar, 1-dollar, 2-dollar, 5-dollar and 10-dollar coins. For x = 63.7, the minimum number of coins is 10: 6 x 10-dollar coins, 1 x 2-dollar coin, 1 x 1-dollar coin, 1 x 0.5-dollar coin and 1 x 0.2 dollar coin. You can assume x must be a sum of the face value of coins. Write the pseudocode to solve this problem.

[The aim of this question is to let you think <u>thoroughly</u> the problem. To solve this problem, dynamic programming is required, which is not covered in this course.
1. If you use the greedy algorithm to solve the above example and point out <u>clearly</u> that for some scenarios it cannot be solved by greedy algorithm, but dynamic programming. And there is no pseudocode for dynamic programming (20 marks).
2. If the student uses dynamic programming to solve the above example. (20 marks)
3. If the student uses dynamic programming and elaborate <u>clearly</u> the reason why greedy algorithm cannot handle all cases, a bonus of 5 marks are awarded. (20 + 5 marks)]

```
# The greedy algorithm approach
Set a dictionary, D, to represent how many coins are needed for the input amount, x
Set coins = [10, 5, 2, 1, 0.5, 0.2, 0.1] #assume coins is a list and the values are arranged in descending order
Set minCoins = 0
Prompt the user for x
While x != 0
        For each coin in coins
                If x >= coin
                        x = x – coin
                        Add 1 to D[coin]
                        Add 1 to minCoins
                        Exit the For-loop # or, break
Return minCoins and D
```

2. [30 marks] Suppose there is a number game between two persons, Alice and Bob. At the beginning of the game, each player starts with a number 0. Each player takes turns and calls a number which is either 1 or 2 bigger than his/her current value. At any moment, the difference between the two numbers of Alice and Bob should not be greater than 2. The winner is the one who calls 5 first.

   For example, at the beginning, Alice holds 0 and Bob holds 0. Alice calls 1. Bob calls 1. Alice calls 3. Bob calls 3. Alice calls 5. Alice wins the game.

   Another example: Alice calls 2. Bob calls 1. Alice cannot call 4 because her number would be 3 greater than Bob. So, her only choice is to call 3. Bob calls 3. Alice calls 5. Alice wins the game.

   Assume Alice will always take the first call.

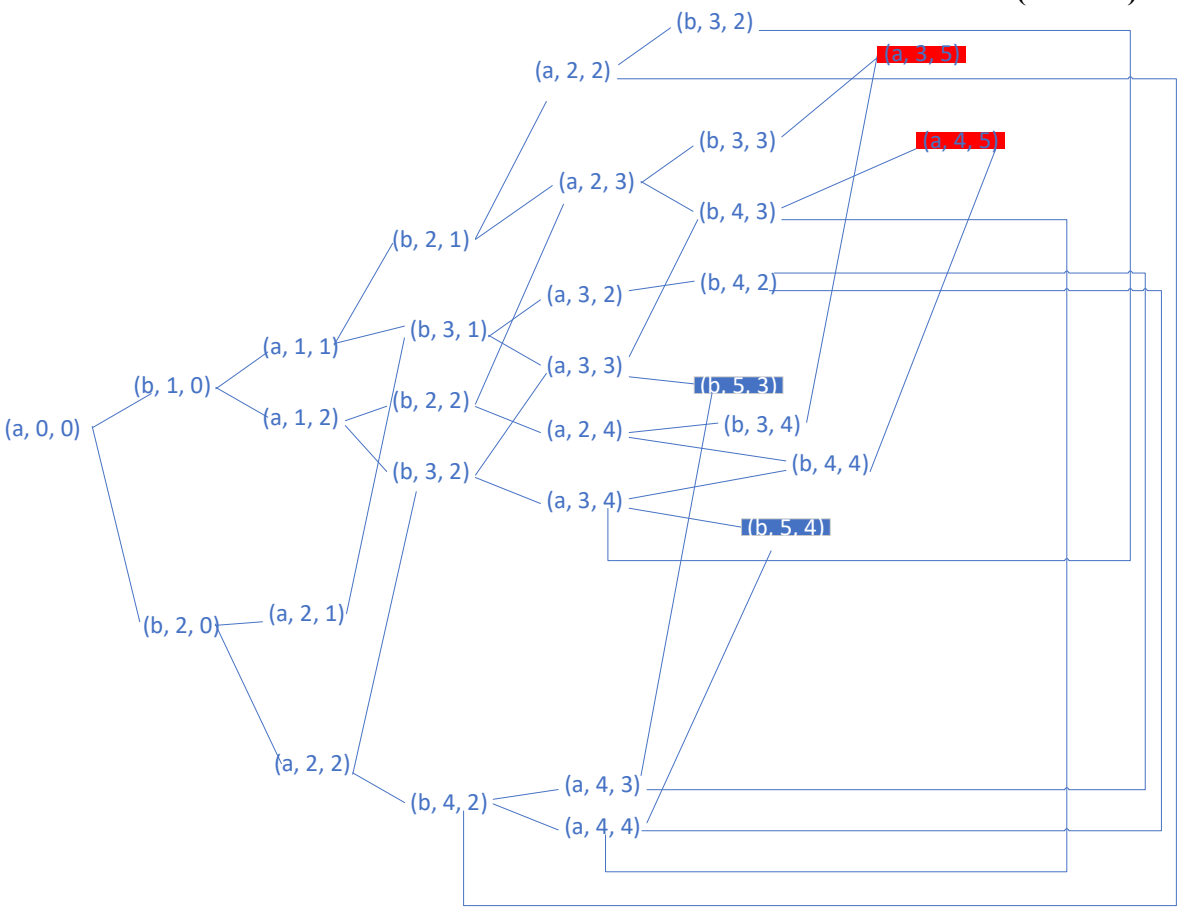
   Answer the following questions:

   a) Define the "state" of this game. Draw a graph to represent all states and their relationship.

   b) Can Bob win the game? Why? State your assumption(s).

a) (x, y, z)
x – The next person who calls a number. 'a' represents Alice and 'b' represent Bob.
y – Current number of Alice
z – Current number of Bob

(5 marks)



(15 marks)

b) Yes. One possible transition is (a, 0, 0) -> (b, 1, 0) -> (a, 1, 1) -> (b, 2, 1) -> (a, 2, 3) -> (b, 3, 3) -> (a, 3, 5). Assumption: The calls of Alice and Bob are random.
No. If Alice always picks the largest possible call, she will always win (because she calls first).
(a, 0, 0) -> (b, 2, 0) -> (a, 2, 1) -> (b, 3, 1) -> (a, 3, 3) -> (b, 5, 3)
(a, 0, 0) -> (b, 2, 0) -> (a, 2, 2) -> (b, 4, 2) -> (a, 4, 3) -> (b, 5, 3)
(a, 0, 0) -> (b, 2, 0) -> (a, 2, 2) -> (b, 4, 2) -> (a, 4, 4) -> (b, 5, 4)

(Either Yes or No with proper assumption can be awarded 10 marks. If you are able to answer BOTH Yes and No with proper assumptions, a bonus of 5 marks can be awarded.)

(10 marks)

3. [15 marks] Create your own *split* function in Python, using the name *mySplit*, which returns a list of words based on two parameters, an input string, and a list of separators such as `[',', '?', '.']` flexibly defined by you. The punctuations and symbols are not considered as a part of a word.

   Write a function called *a3q3()* to test *mySplit*. When *a3q3()* is called, the input/output of your program will look like below:

```
Please input a paragraph: a for apple! b for boy; and c for cat, d for dog. but e for egg? No grammar mistakes checking!!
A list of words from your input:
 ['a', 'for', 'apple', 'b', 'for', 'boy;', 'and', 'c', 'for', 'cat,', 'd', 'for', 'dog', 'but', 'e', 'for', 'egg', 'No', 'gramm
ar', 'mistakes', 'checking']
```

   Zero mark will be awarded if any built-in/external functions, other than the function, *append()*, of list, are used in *mySplit*.

```
def mySplit(strInput, separators = [' ' , ', ' , '; ' , '.', '!', '?']):
    wordsList = []
    word =""
    for i in strInput:
        if i not in separators:
            word += i
        else:
            if word !="": wordsList.append(word)
            word =""
    return wordsList

# a for apple! b for boy; and c for cat, d for dog. but e for egg? No grammar mistakes checking!!
def a3q3():
    strInput = input("Please input a paragraph: ")
    wordsList = mySplit(strInput)
    print("A list of words from your input: \n", wordsList)

a3q3()
```

mySplit (12 marks): separators can be defined in various ways
main (3 marks)

4. [35 marks] Create a Python function, named *wordFeq*(), which returns a sorted list of words or numbers based on frequencies from input paragraph(s). If more than one words are having the same frequency, the sorted results are determined by number, alphabetic case and then alphabetic order (i.e., the order in ASCII code). Your program is required to call *mySplit* developed in Q3. The words are case-sensitive.

   Write a function called *a3q4()* to test your *wordFeq*. When *a3q4()* is called, it should read in a text file which contains the input paragraph(s). The input/output of your program will look like below:

```
Please enter the text file name: A3Q4.txt
Content in the text file:
Computational ? thinking ! involves ideas ' like abstraction, !! data representation, and logically organizing data, which / ar
e also prevalent in other kinds of \ thinking, [such] (as) scientific thinking, engineering thinking, &systems& thinking, desig
n ;
thinking, model-based thinking, and the like.(Source: wikipedia, accessed in 2021)
Words and their Frequencies are shown as below
       Words        |  Frequencies
-------------------------------------
      thinking      |       7
        and         |       2
        data        |       2
         in         |       2
        like        |       2
        2021        |       1
   Computational    |       1
       Source       |       1
    abstraction     |       1
      accessed      |       1
        also        |       1
        are         |       1
         as         |       1
       design       |       1
    engineering     |       1
       ideas        |       1
      involves      |       1
       kinds        |       1
     logically      |       1
    model-based     |       1
         of         |       1
     organizing     |       1
       other        |       1
     prevalent      |       1
   representation   |       1
     scientific     |       1
        such        |       1
      systems       |       1
        the         |       1
       which        |       1
     wikipedia      |       1
```

```
def wordFeq(str):
    # create a word list
    words = mySplit(str,separators = [' ' , ':', ',' , '/' , '\\' ,
                                ';' , '.', '!', '?', '&','(',
                                ')','[',']','\n','\'','\"'])
    wordCount = dict()
    wordCountS = dict()
    topList = {}

    for a in words:
        if a not in wordCount.keys():
            wordCount[a] = 1
        else:
            wordCount[a] = wordCount[a] + 1

    # Sort by number, alphabetic case and then alphabetic order
    for s in sorted(wordCount):
        wordCountS[s] = wordCount[s]

    # Sort by values
    sNum = sorted(set(wordCountS.values()),reverse=True)
    for s in sNum:
        for key,val in wordCountS.items():
            if val == s:
              topList[key] = val

    return topList


def a3q4():
    fileName = input("Please enter the text file name: ")
    infile = open(fileName, "r")
    content = infile.read()
    print("Content in the text file:")
    print(content)
```

```
    topList =  wordFeq(content)
    print ("Words and their Frequencies are shown as below")
    print("{0:^20}|{1:^15}".format("Words","Frequencies"))
    print("-"*(20+15+1))
    for k,v in topList.items():
        print("{0:^20}|{1:^15}".format(k,v))
```

a3q4()

A3Q4.txt

Computational ? thinking ! involves ideas ' like abstraction, !! data representation, and logically organizing data, which / are also prevalent in other kinds of \ thinking, [such] (as) scientific thinking, engineering thinking, &systems& thinking, design ;
thinking, model-based thinking, and the like. (Source: wikipedia, accessed in 2021)

wordFeq: (25 marks)
a3q4: (10 marks)