Assignment-4 Part A

(Total Points: 35)
(Extra Credit: 20)

Instructions

Submission

You can either write your answer using Word/LaTex or take a picture/scan of your hand-written (neat and tidy) solution, then put your solution in one PDF file. Submit your PDF online using Blackboard. All submission must be posted before the deadline.

Problem 1 (Max Points: 10)

Consider the problem of making change for n cents using the fewest number of coins. Assume that each coin's value is an integer.

a) Describe a greedy algorithm to make change consisting of quarters, dimes, nickels and pennies.

b) Give a set of coin denominations for which the greedy algorithm does not yield an optimal solution. Your set should include a penny so that there is a solution for every value of n.

Problem 2 (Max Points: 5)

Given a rod of length n inches and an array of prices (p_i for i = 1..n) of all pieces of size smaller than n, the Rod-cutting problem is to determine the maximum value obtainable by cutting up the rod and selling the pieces.

You do not need to solve the Rod cutting problem, but instead show, by means of a counter example, that the following greedy strategy does not always determine an optimal way to cut rods. Define the density of a rod of length i to be p_i / i, that is its value per inch. The greedy strategy for a rod of length n cuts off a first piece of length i, where 1=< i <= n, having maximum density. It then continues by applying the greedy strategy to the remaining piece of length n–i.

Problem 3 (Max Points: 10)

The Fibonacci numbers are defined by recurrence $F_0 = 0$, $F_1 = 1$ and $F_i = F_{i-1} + F_{i-2}$. Give an O(n) time dynamic-programming algorithm to compute the nth Fibonacci number. Draw the sub-problem graph. How many vertices and edges are in the graph.

Problem 4 (Max Points: 10)

Give a dynamic-programming solution to the 0-1 knapsack problem that runs in O(nW) time where n is the number of items and W is the maximum weight of items that can be put in the knapsack.

Run your algorithm for 5 items (w_i, v_i) as (2,3), (3,4), (4,5), (5,8), (9,10) where w_i is the weight of i-th item and v_i is the value of i-th item and also the maximum weight that can be put in the knapsack(W) is 20. Give the maximum value of items that can be put in the knapsack.

Problem 5 (Max Points: 10) [Extra Credit]

You are climbing a stair case. It takes *n* (> 0) steps to reach to the top.
Each time you can either climb 1, 2 or 3 steps. In how many distinct ways can you climb to the top?

Example:
Number of stairs: 3
Number of ways: 4
Distinct ways to climb to the top: {(1,1,1), (1, 2), (2, 1), (3)}

Note: your algorithm only needs to return the number of distinct ways and not the actual ways to climb to the top.

Problem 6 (Max Points: 10) [Extra Credit]

Describe an efficient algorithm that, given a set {x_1, x_2, …., x_n} of points on the real line, determines the size of the smallest set of unit-length closed intervals that contains all of the given points. Also give the time complexity of your algorithm.

For example: Given points {0.8, 4.3, 1.7, 5.4}, the size of the smallest set of unit-length closed intervals to cover them is 3.

Run your algorithm on the following set of points:
{0.8, 2.3, 3.1, 1.7, 3.6, 4.0, 4.2, 5.5, 5.2, 1.0, 3.9, 4.7} and determine the size of the smallest set of unit-length closed intervals that contains all of the given points.

Assignment-4 Part B

(Total Points: 45)

Instructions:

- Download the attached python file assignment_4.py (make sure to **NOT** change the name of this file).
- Follow the instructions and **replace** all TODO comments in the scaffolding code.
- Test your code as much as you can to make certain it is correct.
- Run flake8 in addition to testing your code; I expect professional and clear code with minimal flake8 warnings and having McCabe complexity (<10) from all of you.
- Create a write up with formatted code and screenshots of your output, running the McCabe complexity command and error free console.
- Save the write-up as a PDF and submit it along with your python code (file name assignment_4.py) as separate attachments before the due date on blackboard.

  Note: Running flake 8
  flake8 path/to/your/file (for warnings and errors)
  flake8 --max-complexity 10 path/to/your/file (for complexity)

Problem 1 (Max points 15):

A numeric sequence of a_i is ordered if $a_1 < a_2 < ... < a_N$. Let the subsequence of the given numeric sequence (a_1, a_2, ..., a_N) be any sequence (a_{i1}, a_{i2}, ..., a_{iK}), where $1 <= i_1 < i_2 < ... < i_K <= N$. For example, sequence (1, 7, 3, 5, 9, 4, 8) has ordered subsequences, e. g., (1, 7), (3, 4, 8) and many others. All longest ordered subsequences are of length 4, e. g., (1, 3, 5, 8).

Your program, when given the numeric sequence, must find the length of its longest ordered subsequence. The input list contains the elements of sequence - N integers in the range from 0 to 10000 each. 1 <= N <= 1000. Your output must contain a single integer that which is the length of the longest ordered subsequence of the given sequence.

Sample Input

```
[1, 7, 3, 5, 9, 4, 8]
```

Sample Output

```
4
```

Problem 2 (Max points 15):

Due to recent rains (I know it's very rare here), water has pooled in various places in the campus, which is represented by a rectangle of N x M (1 <= N <= 100; 1 <= M <= 100) squares. Each square contains either water ('#') or dry land ('-'). A pond is a connected set of squares with water in them, where a square is considered adjacent to all eight of its neighbors. The problem is to figure out how many ponds have formed in the campus, given a diagram of the campus. The campus is represented by a grid represented by a list of N lines of characters separated by ",". Each line contains M characters per line representing one row of the grid (campus). Each character is either '#' or '-'. The characters do not have spaces between them. Write a program to compute and return the number of ponds in the campus.

Sample Input

```
["#--------##-",

 "-###-----###",

 "----##---##-",

 "---------##-",

 "---------#--",

 "--#------#--",

 "-#-#-----##-",

 "#-#-#-----#-",

 "-#-#------#-",

 "--#-------#-"]
```

Sample Output

```
3
```

(There are three ponds: one in the upper left, one in the lower left,and one along the right side)

Problem 3 (Max points 15):

A supermarket has a set ‘Prod’ of products on sale. It earns a profit px for each product x∈Prod sold by a deadline dx that is measured as an integral number of time units starting from the moment the sale begins. Each product takes precisely one unit of time for being sold. A selling schedule is an ordered subset of products Sell ≤ Prod such that the selling of each product x∈Sell, according to the ordering of Sell, completes before the deadline dx or just when dx expires. The profit of the selling schedule is Profit(Sell)=$\Sigma_{x \in Sell}$px. An optimal selling schedule is a schedule with a maximum profit.
For example, consider the products Prod={a,b,c,d} with (pa,da)=(50,2), (pb,db)=(10,1), (pc,dc)=(20,2), and (pd,dd)=(30,1). The possible selling schedules are listed in table 1. For instance, the schedule Sell={d,a} shows that the selling of product d starts at time 0 and ends at time 1, while the selling of product a starts at time 1 and ends at time 2. Each of these products is sold by its deadline. Sell is the optimal schedule and its profit is 80.

schedule	profit
{a}	50
{b}	10
{c}	20
{d}	30
{b,a}	60
{a,c}	70
{c,a}	70
{b,c}	30
{d,a}	80
{d,c}	50

Write a program that reads sets of products from the input and computes the profit of an optimal selling schedule for each set of products. Your input must be a list of n pairs (pi, di) of integers, that designate the profit and the selling deadline of the i-th product. Note: 0 < n < 100, 1 <= pi <= 1000 and 1 <= di <= 1000. For output, the program returns the profit of an optimal selling schedule for the set.

Sample Input 1

```
[(50, 2), (10, 1), (20, 2), (30, 1)]
```

Sample Output 1

```
80
```

Sample Input 2

```
[(20, 1), (2, 1), (10, 3), (100, 2), (8, 2), (5, 20), (50, 10)]
```

Sample Output 2

```
185
```

Note

The sample input contains two product sets. The first set encodes the products from table 1. The second set is for 7 products. The profit of an optimal schedule for these products is 185.