fariha@cse.uiu.ac.bd

Greedy Algorithms

A greedy algorithm always makes the choice that looks best at this moment.

Practice problems:

1. Fractional knapsack

```
Function Fractional-Knapsack (W, v[n], w[n])
1.  Order item-list by $v_i/w_i$ descending
2.  while w > 0 and as long as there are items remaining
3.      pick item with maximum $v_i/w_i$
4.      $x_i \leftarrow \min(1, w/w_i)$
5.      remove item i from list
6.      $w \leftarrow w - x_i w_i$
w: the amount of space remaining in the knapsack (initially w = W)
```

Sample input W n item, value, weight …	Sample output
15 4 silver-dust 300 4 gold-dust 2000 8 salt 80 10 sugar 89 10	Taking gold-dust 8.0 kg -- 2000.0 taka Taking silver-dust 4.0 kg -- 300.0 taka Taking sugar 3.0 kg -- 26.7 taka Total profit 2326.7 taka
25 4 silver-dust 300 4 gold-dust 2000 8 salt 80 10 sugar 89 10	Taking gold-dust 8.0 kg -- 2000.0 taka Taking silver-dust 4.0 kg -- 300.0 taka Taking sugar 10.0 kg -- 89.0 taka Taking salt 3.0 kg -- 24.0 taka Total profit 2413.0 taka

2. Activity Selection Problem

Greedy algorithms for this problem:

- Sort by start time
- Sort by finish time (this one gives the optimal answer)
- Sort by interval

The following two pseudo codes assume that the activities are sorted according to finish time.

Recursive version:

```
RECURSIVE-ACTIVITY-SELECTOR$(s, f, k, n)$
$m = k + 1$
while $m \leq n$ and $s[m] < f[k]$        // find the first activity in $S_k$ to finish
    $m = m + 1$
if $m \leq n$
    return $\{a_m\} \cup$ RECURSIVE-ACTIVITY-SELECTOR$(s, f, m, n)$
else return $\emptyset$
```

For-loop version:

```
GREEDY-ACTIVITY-SELECTOR$(s, f)$
$n = s.length$
$A = \{a_1\}$
$k = 1$
for $m = 2$ to $n$
    if $s[m] \geq f[k]$
        $A = A \cup \{a_m\}$
        $k = m$
return $A$
```

Activity selection in different scenario: Version 1

Suppose you are a coder. You write code for money and charge each customer the same amount, **M**, irrespective of how many hours it takes to write the code. Your customers have sent you **n** coding requests for tomorrow. Each coding request contains the customer id (c_i), the start time (s_i) and the duration (d_i) of the coding task. Write a code to maximize your income tomorrow. Note that you can only write code for one customer at a time.

Sample input	Sample output
M N c_1, s_1, d_1 … c_n, s_n, d_n	
10 4 a 2 8 b 3 4 d 8 1 c 7 1	Profit: 3x10=30 Chosen tasks: b c d

fariha@cse.uiu.ac.bd

Activity selection in different scenario: Version 2

Note that you need **X** hour break between two code writing tasks.

Sample input	Sample output
M, X N c_1, s_1, d_1 … c_n, s_n, d_n	
10 1 4 a 2 8 b 3 4 d 8 1 c 7 1	Profit: 2x10=30 Chosen tasks: b d

3. Maximize your payment

Suppose you are temporarily working as a photoshop editor. You work **10 hours a day** from 12:00 PM to 10 PM. Each of your customers sends you a task request for the next day. The task request contains the payment (p_i), the duration (d_i), and the deadline (dl_i). You must complete a task within the given deadline to get the payment. **Write the following greedy algorithm to maximize your payment.** Note that you can move on to a new task leaving the current task partially complete, and then come back to it later.

Algorithm:

1. Sort the tasks by descending order of *payment per hour*.
2. Choose the task with the maximum *payment per hour* and do it in the last possible hours. Make that hour occupied.
3. Choose the next maximum paid (*payment per hour*) task, find the last possible time that is not occupied, do it at that time and make that hour occupied. If no such time exists, you cannot do this task.
4. Move to the next maximum profitable task (based on *payment per hour*) and repeat step 3.

Sample Input:

Customer	Last time (PM)	Time needed (hour)	Payment (Tk)
a	4	2	2000
b	1	1	1000
c	1	1	4000
d	2	2	3000

Sample Output:

c, a
6000

Customer	Last time (PM)	Time needed (hour)	Payment (Tk)
a	2	1	1000
b	1	1	190
c	2	1	270
d	1	1	250
e	3	1	150

c, a, e
1420

4. Greedy Coin Change

Consider the problem of making change for **N** cents using the fewest number of coins. Assume that each coin's value is an integer. Write a greedy algorithm to make change consisting of quarters (25 cents), dimes (10 cents), nickels (5 cents), and pennies (1 cent).

Sample input N	Sample output

`173`	`25 cents --- 6`
	`10 cents --- 2`
	`1 cents --- 3`
	`Total 11 coins`

Consider the problem of making change for **N** cents using the fewest number of coins. Assume that each coin's value is an integer. Write a greedy algorithm to make change consisting of coins $c_1,\ c_2,\ \ldots,\ c_d$.

Sample input N d $c_1,\ c_2,\ \ldots,\ c_d$	Sample output

`173`
`10 1 25 5`
	`25 cents --- 6`
	`10 cents --- 2`
	`1 cents --- 3`
	`Total 11 coins`

5. Determine the smallest set of unit-length closed intervals

Given a set $x1 \le x2 \le \ldots \le xn$ of points on the real line, give an algorithm to determine the smallest set of unit-length closed intervals that contains all of the points. A closed interval includes both its endpoints; for example, the interval $[1{:}\ 25;\ \ 2{:}\ 25]$ includes all xi such that$1.\ 25 \le xi \le 2.\ 25$.

Sample input n x1, x2, …, xn	Sample output

`6`
`5.22`
`6.1`	

2.2 2.5 3.25 4.8	
5 5.22 6.1 2.2 2.5 3.25	

6. Huffman Encoding

```
HUFFMAN($C$)
1  $n = |C|$
2  $Q = C$
3  for $i = 1$ to $n - 1$
4      allocate a new node $z$
5      $z.left = x =$ EXTRACT-MIN($Q$)
6      $z.right = y =$ EXTRACT-MIN($Q$)
7      $z.freq = x.freq + y.freq$
8      INSERT($Q, z$)
9  return EXTRACT-MIN($Q$)      // return the root of the tree
```

Huffman Decoding

7. More practice problems: https://leetcode.com/tag/greedy/