Greedy Algorithms

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

We hope that a locally optimal choice will lead to a globally optimal solution.

For some problems, it works.

Greedy algorithms tend to be easier to code

A Simple Example

Pick k numbers out of n numbers such that the sum of these k numbers is the largest.

Algorithm

FOR i = 1 to k

Pick out the largest number and

Delete this number from the input. ENDFOR

```
1. for i=1 to k do
2.         pick out the largest number
3.         delete this number from the input
4. end for
```

Optimization Problems

An optimization problem is the problem of finding the best solution from all feasible solutions

- Fractional knapsack: we maximize our profit
- Activity selection: we maximize the number of activities
- Shortest path problem: we minimize the path length.
- Minimum spanning tree: we minimize the spanning tree weight

Practice problems:

PROBLEM 01. Fractional knapsack

The weights and values of **n** items are given. *The items are such that you can take a whole item or some fraction of it (divisible).* You have a knapsack to carry those items, whose weight capacity is **W**. Due to the capacity limit of the knapsack, it might not be possible to carry all the items at once. In that case, pick items such that the profit (total values of the taken items) is maximized.

Write a program that takes the weights and values of **n** items, and the capacity **W** of the knapsack from the user and then finds the items which would maximize the profit using a greedy algorithm.

sample input n weight, value … W	**sample output**
4 4 20 3 9 2 12 1 7 5	item 4: 1.0 kg 7.0 taka item 3: 2.0 kg 12.0 taka item 1: 2.0 kg 10.0 taka profit: 29 taka

Possible greedy strategies:

- Pick the lightest item first, then pick the next lightest item and so on.
- Pick the costliest (per-unit value wise) item first, then pick the next costliest item and so on. **(optimal answer)**
- Pick the costliest (total value wise) item first, then pick the next costliest item and so on.

pseudocode (version 1):

```
Function FractionalKnapsack(W, v[n], w[n])
    5.  sort items by vi/wi descending // vi, wi = value and weight of the ith item
    6.  cap_left = W, profit = 0 // cap_left = capacity left
    7.  i = 1
    8.  while cap_left > 0 and i <= n do
    9.          if cap_left ≥ wi then
    10.                 profit = profit + vi
    11.                 cap_left = cap_left - wi
    12.         else:
    13.                 profit = profit + vi * cap_left/wi
    14.                 cap_left = 0
    15.         i = i+1
    16.         end if
    17. end while
```

pseudocode (version 2):

```
Function FractionalKnapsack(W, v[n], w[n])
1.  sort items by vi/wi descending // vi, wi = value and weight of the ith item
2.  cap_left = W, profit = 0 // cap_left = capacity left
3.  i = 1
4.  while cap_left > 0 and i <= n do
5.          fraction = min(1.0, cap_left/wi) // fraction = fraction taken from ith item
6.          cap_left = cap_left - fraction * wi
7.          profit = profit + fraction * vi
8.          i++
9.  end while
```

ALTERNATIVE QUESTION 01.1: Thieves in warehouse

There are **n** boxes of **n** different items in a warehouse. Each box has a label that says the name (**m_i**), total weight (**w_i**) in kg and the total value (**v_i**) in taka of that item (**i**). *All items are divisible.* Suppose, **k** thieves have come to steal from the warehouse, each with a knapsack of capacity **W_i**. Given each thief wants to maximize his/her profit, how many thieves will be needed to empty the warehouse? Write a code to solve this problem using a greedy algorithm.

Sample input	Sample output
n **m_1, v_1, w_1** **…** **m_n, v_n, w_n** **k** **W_1 W_2 … W_k**	
4 silver-dust 300 4 gold-dust 2000 8 salt 80 10 sugar 89 10 2 15 15	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 Thief 1 profit: 2326.7 taka Taking sugar: 7.0 kg 62.3 taka Taking salt: 8.0 kg 64.0 taka Thief 2 profit: 126.3 taka Total 2 thieves stole from the warehouse. Still following items are left salt 2.0 kg 16.0 taka
4 silver-dust 300 4 gold-dust 2000 8 salt 80 10 sugar 89 10 4 8 10 6 10	Taking gold-dust: 8.0 kg 2000.0 taka Thief 1 profit: 2000.0 taka Taking silver-dust: 4.0 kg 300.0 taka Taking sugar: 6.0 kg 53.4 taka Thief 2 profit: 353.4 taka Taking sugar: 4.0 kg 35.6 taka Taking salt: 2.0 kg 16.0 taka Thief 3 profit: 51.6 taka Taking salt: 8.0 kg 64.0 taka Thief 4 profit: 64.0 taka Total 4 thieves stole from the warehouse.

ALTERNATIVE QUESTION 01.2: Maximize your marks

Write a code for the following scenario using greedy algorithm:

You are attending your mid-term of the course "CS101". The total marks of the exam is M and the total time is T minutes. You have to answer N questions, where the i-th question carries m_i marks and takes t_i minutes for you to answer. The marks you receive will be proportional to the percentage of your answer compared to the full answer, e.g., if a question contains 100 marks and you complete 30% of it, you will get 30 marks.

a) Find the maximum marks you can get in this exam.
b) Print the questions you have to answer for that.
c) Find the maximum marks you can get in this exam if you are allowed to take the same exam in a group with your one friend (as long as a question is answered, both of you get marks irrespective of who answered it) and your friend's answering capacity is exactly the same as you.

Sample Input M T N m_1 t_1 m_2 t_2 … m_n t_n	**Sample Output**
120 20 5 20 10 20 5 30 5 30 6 20 40	Maximum 88 marks answering alone ques 3 100% done -- 30 marks ques 4 100% done -- 30 marks ques 2 100% done -- 20 marks ques 1 40% done -- 8 marks Maximum 107 marks answering with a friend

PROBLEM 02. Activity Selection Problem [CLRS 16.1]

Problem Description: Suppose we have a set $S = \{a_1, a_2, \ldots, a_n\}$ of n proposed activities that wish to use a resource, such as a lecture hall, which can serve only one activity at a time. Each activity a_i has a start time s_i and a finish time f_i, where $0 \le s_i < f_i < \infty$. If selected, activity ai takes place during the half-open time interval $[s_i, f_i)$. Activities a_i and a_j are compatible if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap. That is, a_i and a_j are compatible if $s_i \ge f_j$ or $s_j \le f_i$. In the activity-selection problem, we wish to select a maximum-size subset of mutually compatible activities.

Possible greedy strategies:

- Select the activity that starts first, next select the activity that starts first and does not conflict with the already picked activities
- Select the activity that ends first (this one gives the optimal answer)
- Select the activity that has the shortest duration first

Pseudocode (version 1):

```
Function Greedy-Activity-Selector (activities):
1.  sort activities by finish time ascending
2.  n = activities.length
3.  A= {activities[1]} // A = selected activities
4.  k = 1 // k = last chosen activity
5.  for m=2 to n do
6.          if activities[m].start_time >= activities[k].finish_time then
7.                  A.add( activities[m] )
8.                  k = i
9.          end if
10. end for
11. return A
```

Pseudocode (version 2):

```
Function Greedy-Activity-Selector (s, f):
1.  sort activities by finish time ascending
2.  n = s.length
3.  A = {a_1}
4.  k = 1
5.  for m=2 to n do
6.          if s[m] ≥ f[k] then
7.                  A = A ∪{a_m}
8.                  k = m
9.  end for
10. return A
```

ALTERNATIVE QUESTION 02.1:

Suppose you are managing a multipurpose hall of your university, where seminars, lectures, and even cultural events are held. **N** clubs have sent you booking requests for their events tomorrow. Each booking request contains the club id (**c_i**), the start time (**s_i**) and the duration (**d_i**) of the events. Approve the booking requests such that you can accommodate maximum events tomorrow, without creating conflict. Note that, after an event you need **X** hour to clean up and prepare for the next event.

Sample input	**Sample output**
N c_1 s_1 d_1 c_2 s_2 d_2 … X	
4 a 2 8 b 3 4 d 8 1 c 7 1 0	Chosen clubs: b c d
4 a 2 8 b 3 4 d 8 1 c 7 1 1	Chosen clubs: b d

PROBLEM 03. 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 and there are an infinite number of coins for each coin type.** Write a greedy algorithm to make change consisting of coins $c_1, c_2, \ldots, c_d$.

Sample input N d $c_1, \quad c_2, \quad \ldots, c_d$	Sample output
173 4 10 1 25 5	25 cents --- 6 10 cents --- 2 1 cents --- 3 Total 11 coins

PROBLEM 04. Finding Minimum Stops [Link1]

Suppose you were to drive from A to B, which is **D** miles away, along a straight road. Your gas tank, when full, holds enough gas to travel **m** miles, and you have a map that gives distances between gas stations along the route. Let **d1** < **d2** < … < **dn** be the locations of all the gas stations along the route where **di** is the distance from St. Louis to the gas station. You can assume that the distance between neighboring gas stations is at most **m** miles. Your goal is to make as few gas stops as possible along the way. Give the most efficient algorithm you can to determine at which gas stations you should stop.

Write a code to solve this problem using a **greedy algorithm**. Keep the time complexity of your code O(n).

Sample input **D** **m** **n** **d1 d2 … dn**	Sample output
20 10 8 2 4 5 8 12 14 16 19	stop at gas station 4 (8 miles) stop at gas station 7 (16 miles)
20 10 4 2 8 12 14	Can't reach destination

PROBLEM 05. Determine the smallest set of unit-length closed intervals

Given a set $x1 \leq x2 \leq \cdots \leq xn$ 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 $1.25 \leq xi \leq 2.25$.

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	

PROBLEM 06. 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/