Tutorial (week 11)

All the answers can be given with pseudocode or with Python. Remember that most of the time, many solutions are possible.

1. Let $S = \{a, b, c, d, e, f, g\}$ be a collection of objects with benefit-weight values, $a : (12, 4)$, $b : (10, 6)$, $c : (8, 5)$, $d : (11, 7)$, $e : (14, 3)$, $f : (7, 1)$, $g : (9, 6)$. What is an optimal solution to the fractional knapsack problem for S assuming we have a sack that can hold objects with total weight 18 ? Show your work.

   Solution:
   First, compute the benefit/weight ratio for each object: $a : 3$, $b : 1.67$, $c : 1.6$, $d : 1.57$, $e : 4.67$, $f : 7$, $g : 1.5$. Them sort them decreasingly according to that ratio: $f : 7$, $e : 4.67$, $a : 3$, $b : 1.67$, $c : 1.6$, $d : 1.57$, $g : 1.5$.

   As greedy choice, you would pick first the object with the highest ratio, that is f, which leaves a weight of $18 - 1 = 17$. Then, we would pick the second-highest ratio, that is e, which leaves a weight of $17 - 3 = 14$. Then, we would pick the third-highest ratio, that is a, which leaves a weight of $14 - 4 = 10$. Then, we would pick the fourth-highest ratio, that is b, which leaves a weight of $10 - 6 = 4$. Then, we would pick the fifth-highest ratio, that is c, but we only have 4 of weight remaining, so we can only take 4 of it, while 5 was originally available. At this point, the bag is full and we have a total value of $7 + 14 + 12 + 10 + 8 * (4/5) = 49.4$.

2. Suppose we are given a set of tasks specified by pairs of the start times and finish times as $T = \{(1, 2), (1, 3), (1, 4), (2, 5), (3, 7), (4, 9), (5, 6), (6, 8), (7, 9)\}$. Solve the task scheduling problem for this set of tasks.

   Solution:
   As greedy algorithm, we will consider the tasks one a time, ordered by start time. We assign a new task to any empty machine, and if none is available we create a new one.

   We first assign a machine M_1 for task (1, 2), one machine M_2 for task (1, 3), and one machine M_3 for task (1, 4). Then, task (2, 5) is sent to M_1, task (3,7) is sent to M_2 and task (4, 9) is sent to M_3. Finally, tasks (5, 6) and (6,8) are sent to M_1, and task (7, 9) is sent to M_2.

   Thus, the minimum number of machines required for this task schedule is 3.

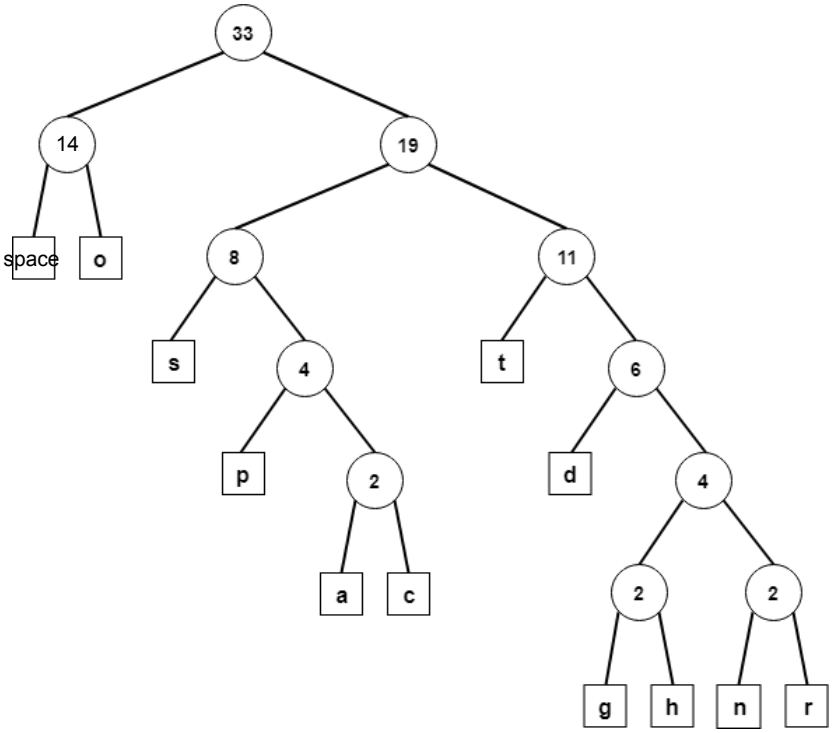
3. Draw the frequency table and Huffman tree for the following string:

   ```
   "dogs do not spot hot pots or cats"
   ```

   Solution:
   The frequency table is as follows:
   space character: 7
   o: 7
   t: 5
   s: 4

d: 2
p: 2
a: 1
c: 1
g: 1
h: 1
n: 1
r: 1

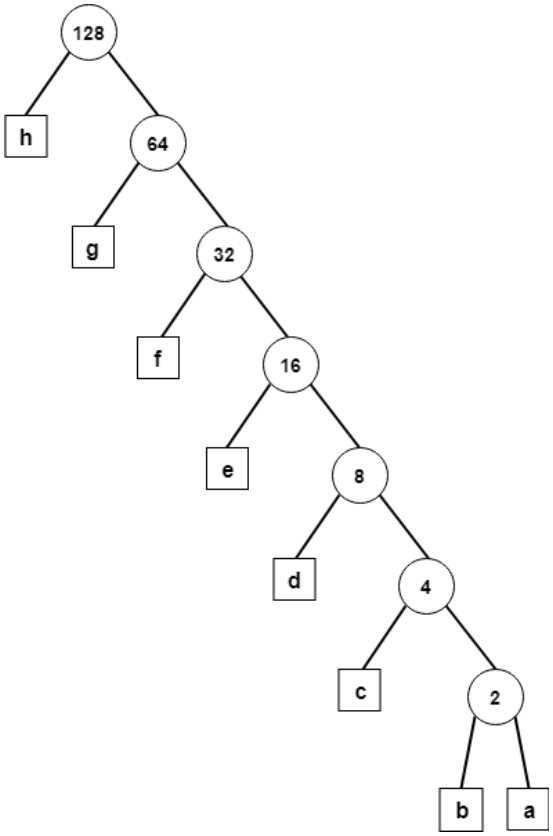
A possible resulting tree for that frequency table would looks like:



4. Give an example set of 8 characters and their associated frequencies so that, in the Huffman tree for this set, every internal node has an external-node child.

Solution:
A set of characters with the following frequencies will work: $\{1, 1, 2, 4, 8, 16, 32, 64\}$.

The resulting tree would look like:



5. Provide an example instance of the fractional knapsack problem where a greedy strategy based on repeatedly choosing as much of the highest-benefit item as possible results in a suboptimal solution.

   Solution:
   With a knapsack of weight 10, consider items with weight-benefit pairs, $(1, 40)$ and $(10, 50)$. This greedy choice picks all of item 2, with benefit 50, whereas choosing item 1 first, and 9 units of item 2, gives benefit $40 + 45 = 95$.

6. Suppose you are given an instance of the fractional knapsack problem in which all the items have the same weight. Show that you can solve the fractional knapsack problem in this case in $O(n)$ time, instead of $O(n \log n)$.

   Solution:
   Suppose the weight of each item is w and we have a knapsack of size W. Then we need to fill the knapsack with the top $\lfloor W/w \rfloor$ items based on their benefits, and then possibly a fraction of the item with benefit rank $\lfloor W/w \rfloor$. Thus, if we used linear-time selection to find the item with benefit rank $\lfloor W/w \rfloor$ in linear time, using for example the quick-select selection algorithm from previous tutorial (exercise 8), then we can scan through the items to find the ones with benefits greater than this item, which all go in the knapsack, and then we can determine the fraction of this item that should go in the knapsack as well. All of this computation therefore runs in $O(n)$ time.

7. A native Australian named Anatjari wishes to cross a desert carrying only a single water bottle. He has a map that marks all the watering holes along the way. Assuming he can walk k miles on one bottle of water, design an efficient algorithm for determining where Anatjari should refill his bottle in order to make as few stops as possible. Argue why your algorithm is correct.

Solution:
Anatjari should use a greedy algorithm. He should draw a line between every pair of watering holes that are within k miles of one another, for he can get from one to the other with one bottle of water. Anatjari should then use a path that uses the fewest number of lines and leads from his present position to a watering hole that is within with k miles of the other side. There can be no path with fewer stops, for he is including in his minimization all pairs of watering holes that can be reached with one bottle of water.

8. Describe an efficient greedy algorithm for making change for a specified value using a minimum number of coins, assuming there are three denominations of coins (called dimes, nickels, and pennies), with values 10, 5, and 1, respectively. Argue why your algorithm is correct.

Solution:
First give as many dimes as possible, then nickels and finally pennies.

In order to prove the correctness of the algorithm, assume that for a change X the greedy algorithm outputs a dimes, b nickels, and c pennies, such that $10a + 5b + c = X$. Let the optimal solution be a' dimes, b' nickels, and c' pennies.

Clearly, we have that $b' \leq 1$ (otherwise we can decrease b' by 2 and increase a' by 1, improving the solution). Similarly, $c' \leq 4$.

From $0 \leq c' \leq 4$ and $X = (2a' + b') * 5 + c'$, we have $c' = X \bmod 5$. Similarly, $c = X \bmod 5$, and hence $c = c'$. Now, let $Y = (X - c)/5$. From $0 \leq b' \leq 1$ and $Y = 2a' + b'$, we have $b' = Y \bmod 2$. Similarly, $b = Y \bmod 2$, and hence $b = b'$. Thus, $a = a'$, $b = b'$, $c = c'$ and the solution given by the greedy algorithm is indeed optimal.

9. Give an example set of denominations of coins so that a greedy change making algorithm will not use the minimum number of coins.

Solution:
If the denominations are \$0.25, \$0.24, \$0.01, then a greedy algorithm for making change for 48 cents would give 1 quarter and 23 pennies ... while two coins of \$0.24 would be a better solution.

10. In the art gallery guarding problem we are given a line L that represents a long hallway in an art gallery. We are also given a set $X = x_0, x_1, ..., x_{n-1}$ of real numbers that specify the positions of paintings in this hallway. Suppose that a single guard can protect all the paintings within distance at most 1 of his or her position (on both sides). Design an algorithm for finding a placement of guards that uses the minimum number of guards to guard all the paintings with positions in X.

Solution:
We can use a greedy algorithm, which seeks to cover all the designated points on L with the fewest number of length-2 intervals (for such an interval is the distance one guard can protect). This greedy algorithm starts with x_0 and covers all the points that are within distance 2 of x_0. If x_i is the next uncovered point, then we repeat this same covering step starting from x_i. We then repeat this process until we have covered all the points in X.

Hints

- **Question 3: Don't forget to include the space character.**
- **Question 4: Think about powers of 2.**

- **Question 5: Think about how to create items with large weights and high benefit.**
- **Question 6: You may use the fact that it is possible to solve the selection problem for n numbers in $O(n)$ time (for example using quick-select algorithm seen in Exercise 8 of the last tutorial.)**