INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact factor: 4.295
(Volume 5, Issue 5)
Available online at: www.ijariit.com

Greedy algorithm

Kishan Senjaliya
kishu2107@gmail.com
G. H. Patel College of Engineering and Technology, Anand, Gujarat

ABSTRACT

This research will help you go the through all the greedy algorithm and how greedy algorithm approaches on different kind of problems and give the optimal solution at a particular instant of step. We will go through different Greedy problems like making change problem, minimal spanning tree problems and knapsack problem in detail and help you clear out how this method work.

Keywords— *Greedy Algorithm, Fractional Knapsack, Making change problem, Huffman code, Computer science*

1. INTRODUCTION

First of all what is a greedy algorithm or a greedy approach, it basically chooses the optimal solution or optimal choice from the given set of choices to make it locally optimal and in aim of making the problem globally optimal. Optimal solution should contain sub problems also to be optimal. It chooses the optimal choice ignoring the ramification which can occur in the future. In order to make a problem globally optimal, greedy algorithm chooses to have two sets. First set contains the solution which algorithm found it optimal and stored it. Second set contains the solution which algorithm considered it but does found it optimal and stored in second set. Greedy approach is quite straightforward, efficient and easy to learn and covers wide range of problems. But greedy approach does not always result in an optimal solution. The greedy algorithm consists of four functions.

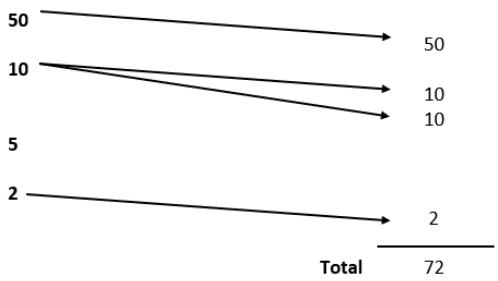
- **Solution Function:** A function that checks whether chosen set of items provides a solution.
- **Feasible Function:** A function that checks the feasibility of a set.
- **Selection Function:** The selection function tells which of the candidates is the most promising.
- **Objective Function:** An objective function, which does not appear explicitly, but gives the value of a solution.

2. TYPES OF GREEDY PROBLEM

(a) Making change
(b) Fractional knapsack
(c) Huffman code

2.1 Making Change problem

Making Change problem is the problem which we go through in day to day life like paying 74rs to a shopkeeper with the different denominations of coins or notes available with us. Most of solve this kind of problem in daily routine by doing it instantaneously, but unconsciously we use Greedy algorithm. For example considering available coin denomination are of {2, 5, 10, and 50} and we want pay amount of 72rs with the lowest number of coins. For that we use one 50rs coin, two 10rs coin and one 2rs coin to make total amount of 72rs. So in this we make the optimal solution by using the largest denomination to add up so to make total amount and thus at every stage we add the already chosen largest coin value available that does not take us past the amount to be paid and similarly we use greedy choice to make the whole problem optimal.



Algorithm
MAKE-CHANGE (n)
C ← {100, 25, 10, 5, 1} // constant.
Sol ← {}; // set that will hold the solution set.
Sum ← 0 sum of item in solution set

WHILE sum not = n
x = largest item in set C such that sum + x ≤ n
IF no such item **THEN**

RETURN "No Solution"
S ← S {value of x}
sum ← sum + x

RETURNS

2.2 Fractional knapsack problem
To understand this problem easily and in a practical way, first assume you have a sack which contains its maximum weight has X and you have some number of different object which has a particular different value and weight for different objects. So, your main aim is to fill the sack with the objects to yield maximum value by keeping in mind the crucial sack's maximum weight. There are three plausible selection methods:
(a) Selecting the highest value object first.
(b) Selecting the lowest weight object first.
(c) Object whose value per unit weight is as high as possible

Here is a very simple example to make you understand:
- We have 5 objects and weight capacity of knapsack is 10.
- For each object the weight and value are mentioned in the below table.

Object	**1**	**2**	**3**	**4**	**5**
Weight(w)	1	2	3	4	5
Value(v)	18	40	56	30	75
v/w	18	20	18.66	7.5	15

Now the solution table for three different case:

Selection	**x**					**value**
Max v	0	1	1	0	1	171
Min w	1	1	1	1	0	144
Max v/w	1	1	1	0	0.8	**174**

So the most optimal solution was given by max v/w ratio which is 174.

Algorithm
Fractional Knapsack (Array W, Array V, int M)
 for i <- 1 to size (V)
  calculate cost[i] <- V[i] / W[i]
 Sort-Descending (cost)
 i ← 1
 while (i <= size(V))
  if W[i] <= M
   M ← M – W[i]
   total ← total + V[i];
  if W[i] > M
   i ← i+1

2.3 Huffman Code
Huffman code is special type of optimal prefix code generally used to effectively compress data savings from 20% to 90% depending on the characteristics if the data being compressed. The main goal behind this algorithm is to assign variable-length codes to input characters. The most frequent character gets the smallest code and the least frequent character gets the largest code. Huffman's greedy algorithm uses a table giving how often each character occurs to build up an optimal way of representing each character as a binary string.

In Prefix codes, the codes (bit sequences) are assigned in such a way that the code assigned to one character is not a prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bit stream.

2.3.1 There are mainly two major parts in Huffman Coding

(a) Build a Huffman Tree from input characters.

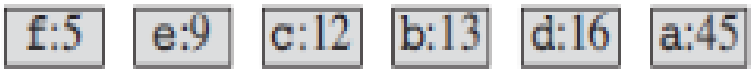
(b) Traverse the Huffman Tree and assign codes to characters.

Example

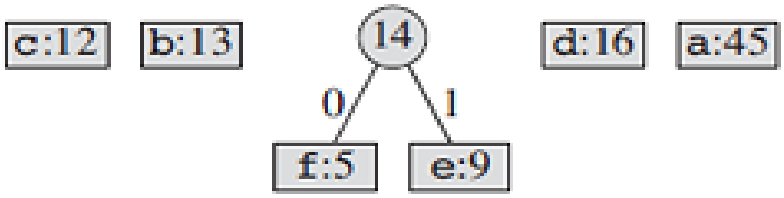
If we have 100,000 character data that we wish to store in compressed form and we have 6 different character which appears. The table below gives the frequency of each character from the data.

Characters	**a**	**b**	**c**	**d**	**e**	**f**
Frequency (in thousand)	45	13	12	16	9	5

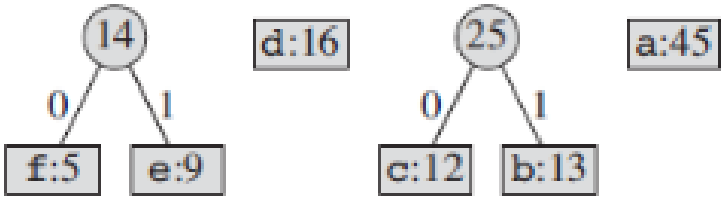
Step 1: Arrange elements in ascending order



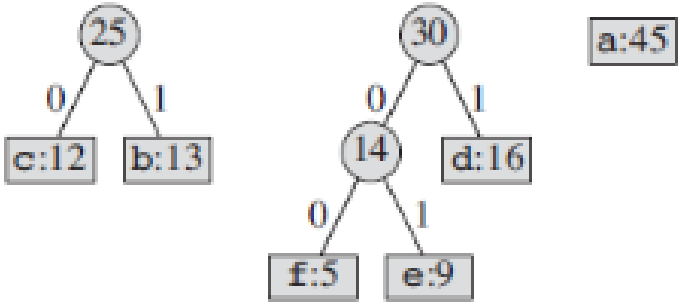
Step 2:



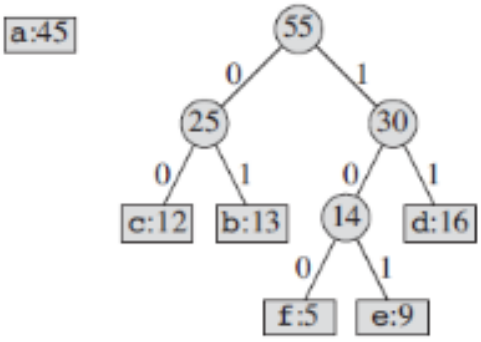
Step 3:



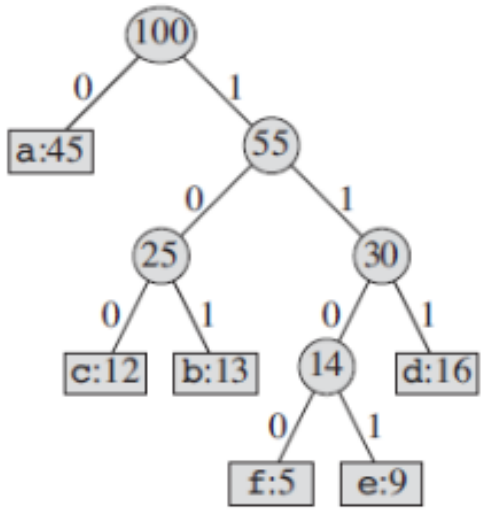
Step 4:



Step 5:



Step 6:



Frequency (in thousands)	**a**	**b**	**c**	**d**	**e**	**f**
	45	13	12	16	9	5
Huffman code-word	0	101	100	111	1101	1100

Algorithm

Step 1: HUFFMAN(C)
Step 2: n=|C|
Step 3: Q=C
Step 4: fori=1ton-1
Step 5: allocate a new node z
Step 6: z.left = x = EXTRACT-MN(Q)
Step 7: z.right = y = EXTRACT-MIN(Q)
Step 8: z.freq = x.freq + y.freq
Step 9: INSERT(Q,z)
Step 10: return EXTRACT-MIN(Q) // return the root of the tree

3. REFERENCES

[1] https://www.geeksforgeeks.org/greedy-algorithms/
[2] https://en.wikipedia.org/wiki/Greedy_algorithm
[3] https://brilliant.org/wiki/greedy-algorithm/
[4] AJudith Gal-Ezer, Ela Zur for the efficiency of greedy algorithms
[5] Introduction to Algorithms 3ed by Cormen, Leiserson, Rivest, and Clifford
[6] http://www.darshan.ac.in/Upload/DIET/Documents/CE/Analysis%20of%20Algorithms_12112014_054619AM.pdf

