Greedy Algorithm

INSTRUCTOR: DR. TARUNPREET BHATIA

ASSISTANT PROFESSOR, CSED

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY

Contents

Greedy Technique

- Primarily used in Optimization problems where in we would like to find the best of all possible solutions.
- In other words we need to find the solution which has the optimal (maximum or minimum) value satisfying the given constraints.
- The Greedy approach helps in constructing a solution for a problem through a sequence of steps (piece by piece) where each step is considered to be a partial solution. This partial solution is extended progressively to get the complete solution.
- In the greedy approach each step chosen has to satisfy the constraints given in the problem. Each step is chosen such that it is the best alternative among all feasible choices that are available.

Characteristics of Greedy approach

- **Greedy choice property:** The global optimal solution is found by selecting locally optimal choices, or the ones that appear to be the best at the time. The current decision may be influenced by prior decisions, but it is independent by future decisions. The choice of each step in a greedy approach is done based on the following:
  - It must be feasible
  - It must be locally optimal
  - It must be irrevocable
  - A locally optimal choice is considered globally optimal
- **Optimal substructure:** We say given problem exhibits optimal substructure if the optimal solution to the given problem contains the optimal solution to its subproblems too. In the problem which possesses the optimal substructure, best next choice always leads to an optimal solution.

Pillars of Greedy Approach

- A candidate set, from which a solution is created
- A selection function, which chooses the best candidate to be added to the solution
- A feasibility function that is used to determine if a candidate can be used to contribute to a solution
- An objective function, which assigns a value to a solution, or a partial solution, and
- A solution function, which will indicate when we have discovered a complete solution

Does greedy choice always work?

- In some cases making a decision that looks right at that moment gives the best solution (Greedy), but in other cases, it doesn’t.

Greedy Algorithms Done so Far.....

Dijkstra's algorithm for finding the shortest path in a graph

- Always takes the *shortest* edge connecting a known node to an unknown node

Kruskal's algorithm for finding a minimum-cost spanning tree

- Always tries the *lowest-cost* remaining edge

Prim's algorithm for finding a minimum-cost spanning tree

- Always takes the *lowest-cost* edge between nodes in the spanning tree and nodes not yet in the spanning tree

Interval Scheduling problem (Activity Selection)

- You are given **n** activities with their start and finish times. Select the maximum number of activities that can be performed by a single person, assuming that a person can only work on a single activity at a time.
- More formally, Given set $S = \{A_1, \ldots, A_n\}$ of activities and activity start and finish times. Find a maximum-size subset of mutually compatible activities.

  $S_{ij} = \{A_k \in S : f_i \leq s_k < f_k \leq s_j\}$
- Activities are compatible if their time intervals do not overlap, i.e., A_i is compatible with A_j is $s_i \geq f_j$ or $s_j \geq f_i$.

Greedy Template

Consider jobs in some order. Take each job provided it's compatible with the ones already taken.

Activity	A1	A2	A3	A4	A5	A6
Start Time	0	1	3	5	5	8
Finish Time	6	2	4	7	9	9

Greedy Template

Consider jobs in some order. Take each job provided it's compatible with the ones already taken.

Activity	A1	A2	A3	A4	A5	A6
Start Time	0	1	3	5	5	8
Finish Time	6	2	4	7	9	9
Duration						

Counter Example for Duration

Activity	A1	A2	A3
Start Time			
Finish Time			
Duration			

Applications of Activity Selection

- Scheduling events in a room having multiple competing events
- Scheduling and manufacturing multiple products having their time of production on the same machine
- Scheduling meetings in one room

Basic Idea

- The greedy choice is to always pick the next activity whose finish time is the least among the remaining activities and the start time is more than or equal to the finish time of the previously selected activity.
- Follow the given steps to solve the problem:

1. Sort the activities according to their finishing time
2. Select the first activity from the sorted array and print it
3. Do the following for the remaining activities in the sorted array

   3a. If the start time of this activity is greater than or equal to the finish time of the previously selected activity then select this activity and print it

Algorithm

Assume: Activities are sorted in ascending order of their finishing time

```
GREEDY- ACTIVITY SELECTOR (s, f)
1. n = length[s]
2. A = {$A_1$}
3. i = 1                //Index of last selected activity
4. for m = 2 to n
5.        do if s[m] $\geq$ f[i]
6.        then A = A $\cup$ {$A_m$}
7.        i = m
8. return A
```

Example 2

i	1	2	3	4	5	6	7	8	9	10	11
s[i]	1	3	0	5	3	5	6	8	8	2	12
f[i]	4	5	6	7	9	9	10	11	12	14	16

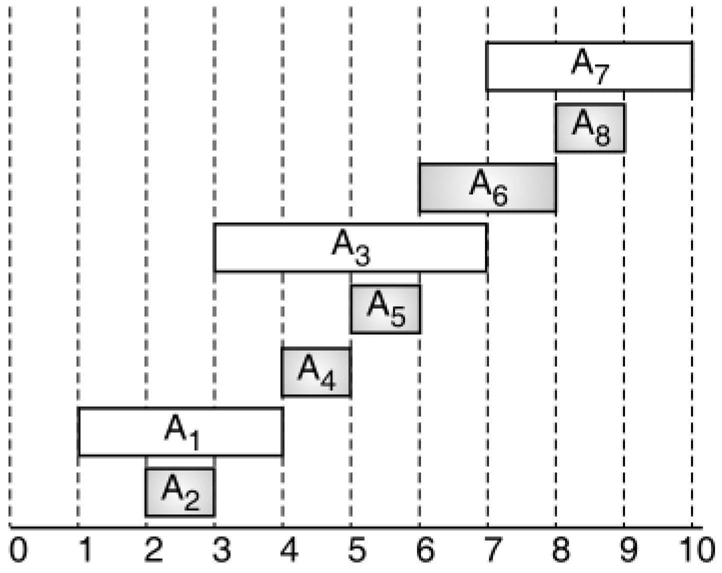
Example 3

Given following data of activities A_1 to A_8, determine the optimal schedule for activity selection using greedy algorithm.

s[i]	1	2	3	4	5	6	7	8
f[i]	4	3	7	5	6	8	10	9

Sort all activities by their finishing time and then check compatibility

i	1	2	3	4	5	6	7	8
id	A2	A1	A4	A5	A3	A6	A8	A7
s[i]	2	1	4	5	3	6	8	7
f[i]	3	4	5	6	7	8	9	10



Final schedule is A = <A_2, A_4, A_5, A_6, A_8>

Try this!

Let A = {(1, 3),(2, 4),(3, 5),(2, 7),(4, 6),(5, 6),(3, 7),(5, 8),(6, 10),(7, 9),(8, 10)} denote the (start, finish) times for a collection of 11 tasks.

a) Solve the Interval Scheduling Problem for this collection of tasks (i.e. find the maximum number of tasks that can be scheduled on a single machine, and give a set of compatible tasks that achieves this maximum).

b) Solve the Total Interval Scheduling for this collection of tasks (i.e. find the minimum number of machines required to complete all tasks, and give a schedule for doing so).

Solution (a)
T = {(1, 3),(2, 4),(3, 5),(2, 7),(4, 6),(5, 6),(3, 7),(5, 8),(6, 10),(7, 9),(8, 10)}

Solution (b)

T = {(1, 3),(2, 4),(3, 5),(2, 7),(4, 6),(5, 6),(3, 7),(5, 8),(6, 10),(7, 9),(8, 10)}

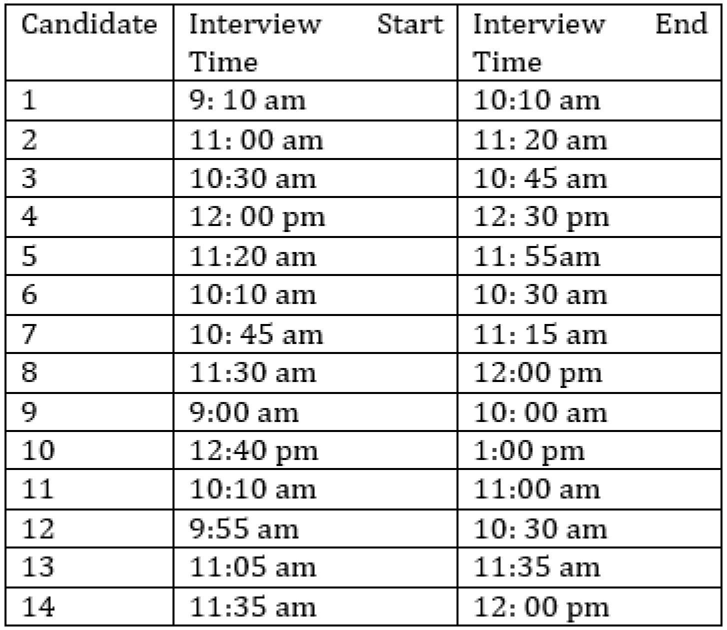
Total Interval Scheduling

1. Maintain a set of free (but already used) lecture halls *F* and currently busy lecture halls *B*.
2. Sort the classes by start time.
3. For each new start time which you encounter, remove a lecture hall from *F*, schedule the class in that room, and add the lecture hall to *B*. If *F* is empty, add a new, unused lecture hall to *F*. When a class finishes, remove its lecture hall from *B* and add it to *F*.

Solution (b)
T = {(1, 3),(2, 4),(3, 5),(2, 7),(4, 6),(5, 6),(3, 7),(5, 8),(6, 10),(7, 9),(8, 10)}

Try this!

An interview is scheduled on a particular day where each candidate is to be interviewed by a single interviewer. The start time and end time of the interview for each candidate is given in the table. Design and apply the efficient algorithm for determining the minimum number of interviewers to interview all the candidates, with a minimum of 3 candidates for each interviewer. You need to also show the candidate list for each interviewer.



Candidate	Interview Start Time	Interview End Time
1	9: 10 am	10:10 am
2	11: 00 am	11: 20 am
3	10:30 am	10: 45 am
4	12: 00 pm	12: 30 pm
5	11:20 am	11: 55am
6	10:10 am	10: 30 am
7	10: 45 am	11: 15 am
8	11:30 am	12:00 pm
9	9:00 am	10: 00 am
10	12:40 pm	1:00 pm
11	10:10 am	11:00 am
12	9:55 am	10: 30 am
13	11:05 am	11:35 am
14	11:35 am	12: 00 pm

Complexity Analysis

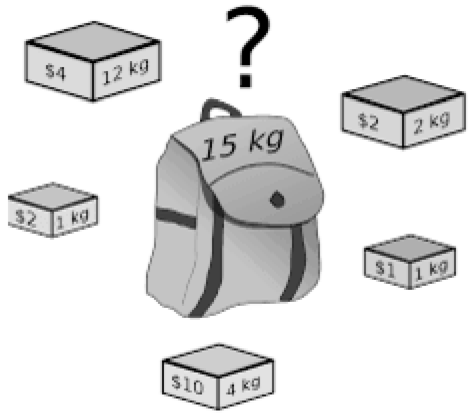
- When activities are sorted by their finish time: O(N)
- When activities are not sorted by their finish time, the time complexity is O(N log N) due to complexity of sorting

Partial Knapsack Problem

Fractional Knapsack problem is defined as, “Given a set of items having some weight and value/profit associated with it. The knapsack problem is to find the set of items such that the total weight is less than or equal to a given limit (size of knapsack) and the total value/profit earned is as large as possible.”

More formally, we are given n objects and a knapsack. Object i has weight w_i and the knapsack has a capacity M. If a fraction x_i, $0 <= x_i <= 1$ of object i is placed into the knapsack then a profit of $p_i x_i$ is earned. The profits & weights are positive.

In general, for n items, knapsack has 2^n choices. So brute force approach runs in $O(2^n)$ time.



Greedy template

Several choices:

- Select the item based on the maximum profit.

Item i	Weight (kg) w_i	Profit p_i
I1	5	30
I2	10	40
I3	15	45
I4	22	77
I5	25	90

Knapsack capacity is 60kg

Greedy template

Several choices:

- Select the item based on the minimum weight.

Item i	Weight (kg) w_i	Profit p_i
I1	5	30
I2	10	40
I3	15	45
I4	22	77
I5	25	90

Knapsack capacity is 60kg

Greedy template

Several choices:

- Select the item based on the ratio of profit/weight.

Item i	Weight (kg) w_i	Profit p_i
I1	5	30
I2	10	40
I3	15	45
I4	22	77
I5	25	90

Knapsack capacity is 60kg

Basic Approach

1. For each item, compute its profit/weight ratio.

2. Arrange all the items in decreasing order of their profit/weight ratio.

3. Take the item with the highest ratio and add them until we cannot add the next item as a whole.

4. At the end, add next item as much (fraction) as you can.

Item i	Weight (kg) w_i	Profit p_i
I1	5	30
I2	10	40
I3	15	45
I4	22	77
I5	25	90

Partial Knapsack Algorithm

Input: Set S of items with weight w_i and benefit/profit p_i with total weight W

Output: Amount x_i of each item i to maximize profit with weight at most W and total profit

```
1.  for each item i in S {
2.      $x_i=0$
3.      $v_i=p_i/w_i$
4.  }
5.  Sort all the items in **decreasing order of the $v_i$**
6.  w=0, total_profit=0
7.  while w<=W {
8.      remove item i with highest $v_i$
9.          $x_i=\min(w_i, W-w)$
10.         total_profit = total_profit + $v_i*x_i$
11.         $w=w+x_i$
12. }
```

Example 1

A thief enters a house for robbing it. He can carry a maximal weight of 12 kg into his bag. There are 5 items in the house with the following weights (in kg) and values. What items should thief take if he can even take the fraction of any item with him?

Item i	Weight w_i	Profit p_i	
I1	2	15	
I2	5	12	
I3	3	9	
I4	4	16	
I5	6	17	

Solution

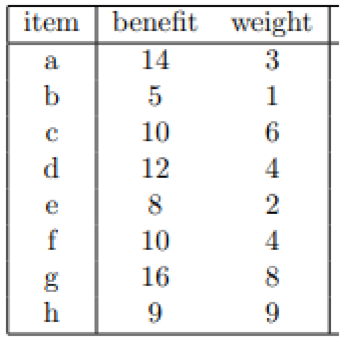
Decreasing order of v_i

Item i	Weight w_i	Profit p_i	$v_i=p_i/w_i$
I1	2	15	7.5
I4	4	16	4
I3	3	9	3
I5	6	17	2.8
I2	5	12	2.4

Item i considered	Amount of x_i in Knapsack	Knapsack weight w	Total_profit

Try this!

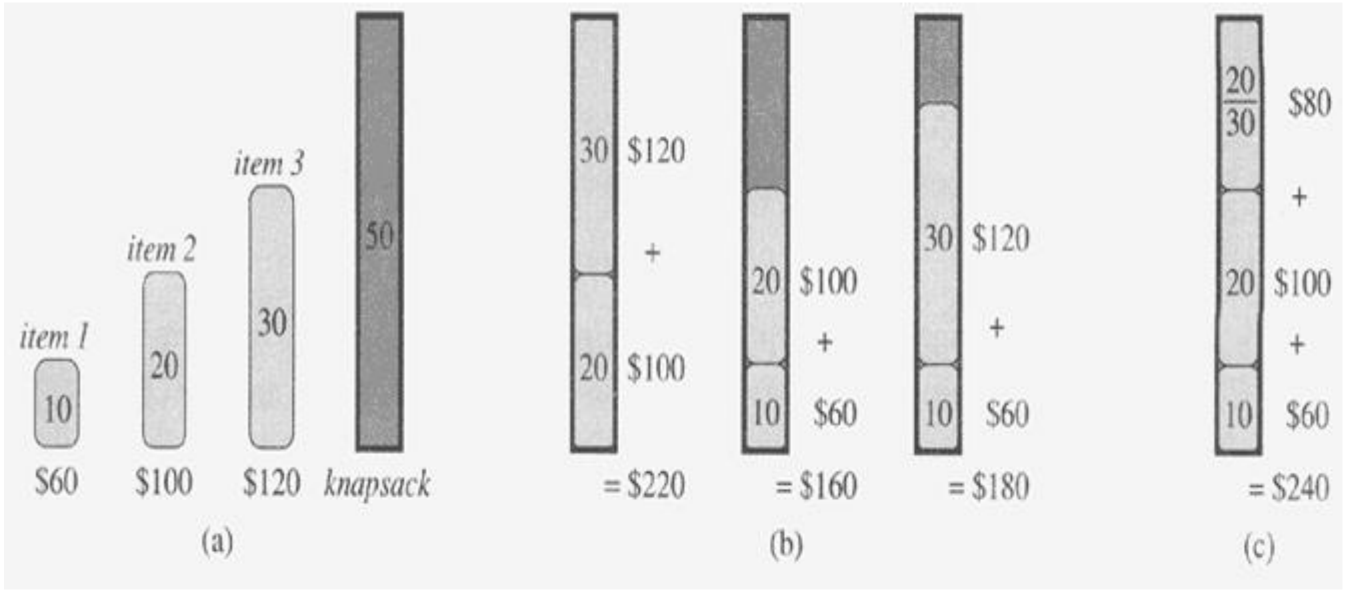
Consider the set of items S = {a, b, c, d, e, f, g, h} where the benefits/profits and weights are given in the table below.



item	benefit	weight
a	14	3
b	5	1
c	10	6
d	12	4
e	8	2
f	10	4
g	16	8
h	9	9

Solve the Fractional Knapsack Problem for this set of items, where the maximum total allowed weight is W_{max} = 15.

Greedy algorithm doesn’t give optimal solution for 0-1 Knapsack



(a) Thief must select a subset of items with knapsack capacity of 50 pounds

(b) Possible subsets for 0/1 Knapsack

(C) Fractional Knapsack solution

Applications

- In logistics, it can be used to determine the most efficient way to load a truck with a given set of items.
- In finance, it can be used to choose which investments to make in order to maximise return while staying within a budget.

Huffman Coding

- Huffman coding is a lossless data compression algorithm.
- The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters.
- The character which occurs most frequently gets the smallest code.
- The character which occurs least frequently gets the largest code.
- Formally, Given a set of n characters from the alphabet A (each character $c \in A$) and their corresponding frequency *freq(c)*, find a binary code for each character such that $\sum_{c \in A} freq(c) * |binarycode(c)|$ is minimum, where $|binarycode(c)|$ represents the length of binary code of character c.

Prefix code/Prefix-free code

- The variable-length codes assigned to input characters are Prefix free Codes, means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not the prefix of code assigned to any other character.
- This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream.

Steps for creating Huffman Tree

1. Create a leaf node for each character of the text. Leaf node of a character contains the occurring frequency of that character.
2. Arrange all the nodes in increasing order of their frequency value.
3. Considering the first two nodes having minimum frequency, Create a new internal node. The frequency of this new node is the sum of frequency of those two nodes. Make the first node as a left child and the other node as a right child of the newly created node.
4. Keep repeating Step-02 and Step-03 until all the nodes form a single tree.
5. The tree finally obtained is the desired Huffman Tree. Leaves of the binary tree will represent the bit strings. The nodes with larger frequency will be nearer to the root and with lesser frequency will be towards the leaves.

Huffman Coding Rule

- **Convention 1: If you assign weight ‘0’ to the left edges, then assign weight ‘1’ to the right edges.**
- Convention 2: If you assign weight ‘1’ to the left edges, then assign weight ‘0’ to the right edges.
- Any of the above two conventions may be followed. But follow the same convention at the time of decoding that is adopted at the time of encoding.
- We will follow Convention 1.

Algorithm

```
HUFFMAN(C)
1. n ← |C|
2. Q ← C            //Q is priority queue on the basis of frequency
3. for i ← 1 to n − 1
4.     do allocate a new node z
5.     left[z] ← x ←EXTRACT-MIN(Q)
6.     right[z] ← y ←EXTRACT-MIN(Q)
7.     f [z] ← f [x] + f [y]
8.     INSERT(Q, z)
9. return EXTRACT-MIN(Q)
```

Time Complexity

O(N logN),where n is the number of unique characters.

Example 1

File f has one million characters. Only a, b, c, d, e, f chars come in the file. Assume frequency of occurrence of a, b, c, d, e, f is 45%, 13%, 12%, 16%, 9%, 5% . Determine:

a) Huffman Code for each character

b) Calculate bits saved

c) Average code length

Frequency of occurrence of a, b, c, d, e, f is 45%, 13%, 12%, 16%, 9%, 5%

Character	Probability	Fixed length code	HuffmanBinarycode	|Binarycode|
a	0.45	000	0	1
b	0.13	001	101	3
c	0.12	010	100	3
d	0.16	011	111	3
e	0.09	100	1101	4
f	0.05	101	1100	4

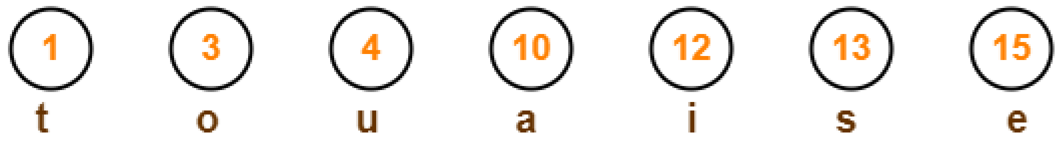
What about decoding?

1. We start from the root and do the following until a leaf is found.
2. If the current bit is 0, we move to the left node of the tree.
3. If the bit is 1, we move to right node of the tree.
4. If during the traversal, we encounter a leaf node, we print the character of that particular leaf node and then again continue the iteration of the encoded data starting from step 1.

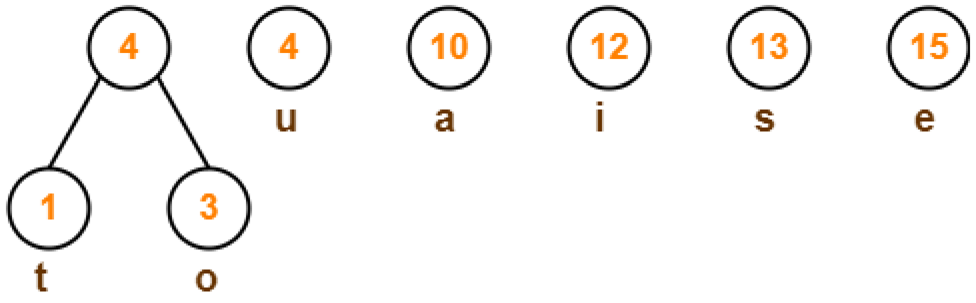
Example 2

Characters	a	e	i	o	u	s	t
Frequency	10	15	12	3	4	13	1

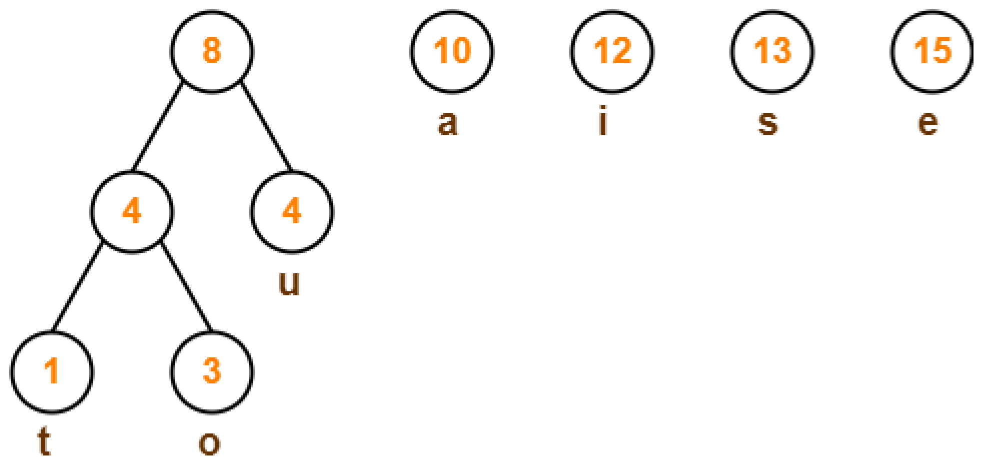
Step-01:



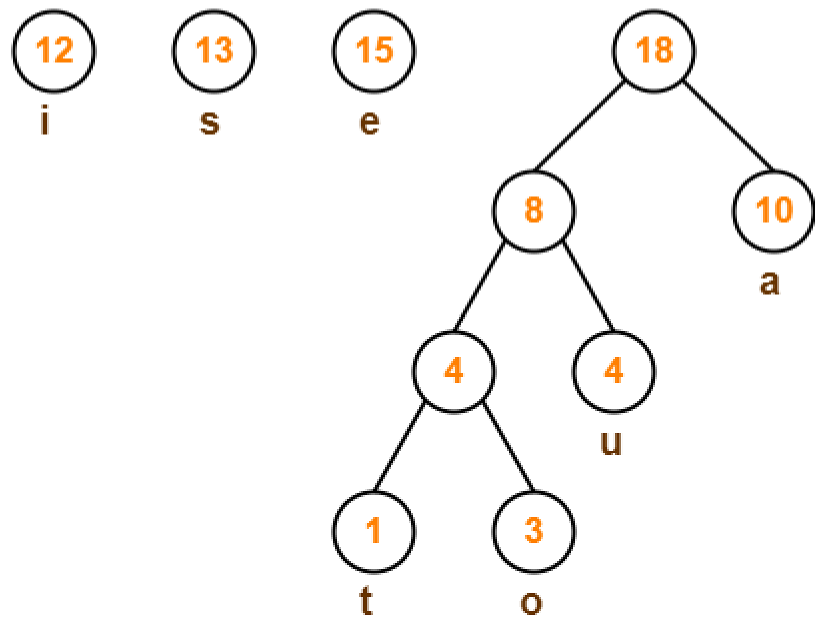
Step-02:



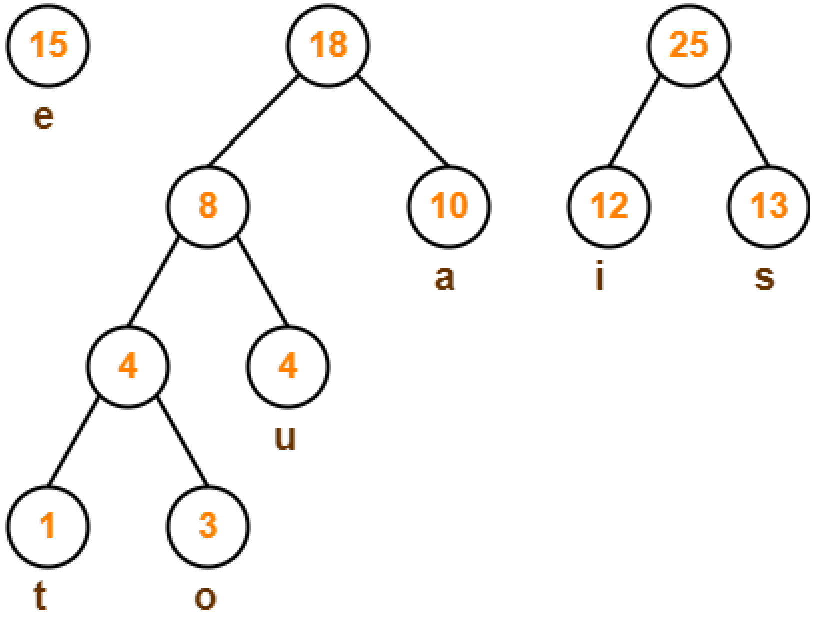
Step-03:



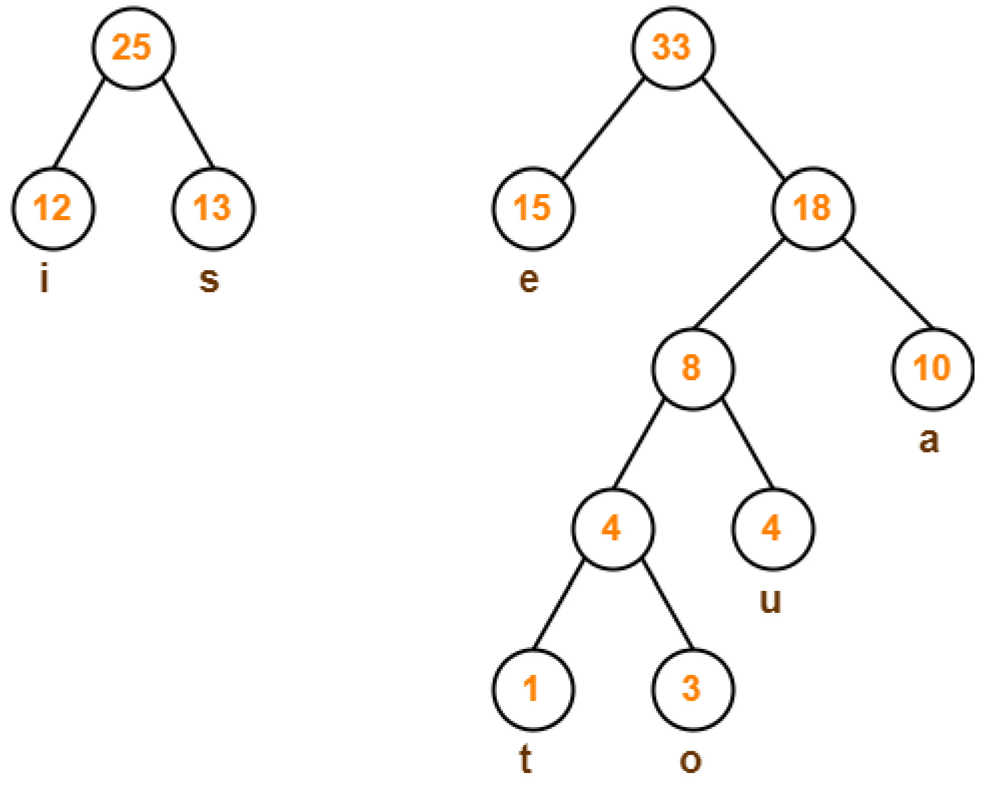
Step-04:



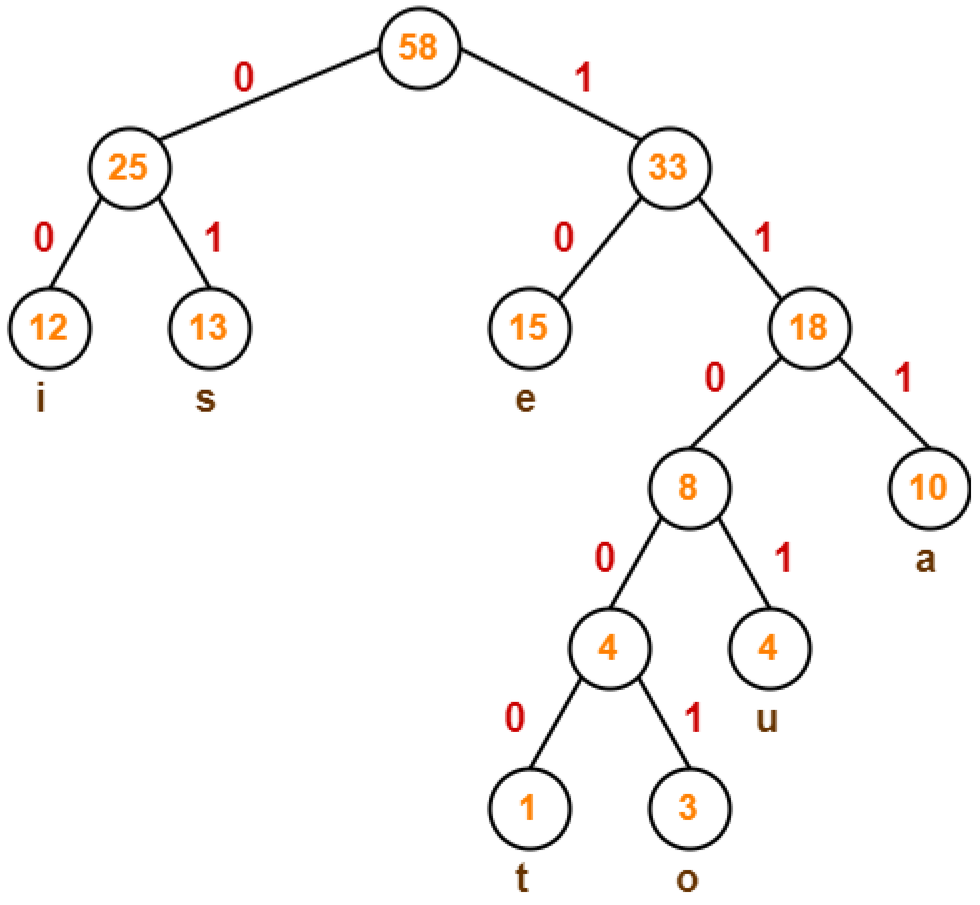
Step-05:



Step-06:



Step-07:



Huffman Code for each character is:

- a = 111
- e = 10
- i = 00
- o = 11001
- u = 1101
- s = 01
- t = 11000

Average code length

= ∑ (frequency_i x code length_i) / ∑ (frequency_i)

= { (10 x 3) + (15 x 2) + (12 x 2) + (3 x 5) + (4 x 4) + (13 x 2) + (1 x 5) } / (10 + 15 + 12 + 3 + 4 + 13 + 1)

= 2.52

Try this!

A networking company uses a Huffman compression technique to encode the message before transmitting over the network. Consider the following characters with their frequency:

Character	Frequency
D	18
A	37
T	29
S	50
R	17
U	30
C	6
E	13

a) Show stepwise a tree that Huffman's greedy algorithm could produce for these characters.

b) Encode the message "DATASTRUCTURES" before transmitting on a network.

c) If the codeword for character "C" is reversed, will there be any effect on decoding the message at receiver. Justify your answer with example.

Try this!

Consider the text message as shown in the Table 1. Apply and verify the text compression algorithm which is based on idea that more number of common letters required fewer bits and less common letters required more number of bits. Assume that the blank space is considered as “-” with frequency 1. Draw the appropriate data structure to find the length of message after compression. How many bits are required for transferring this message before and after compression? Also, deduce the compression ratio. [MST 2023]

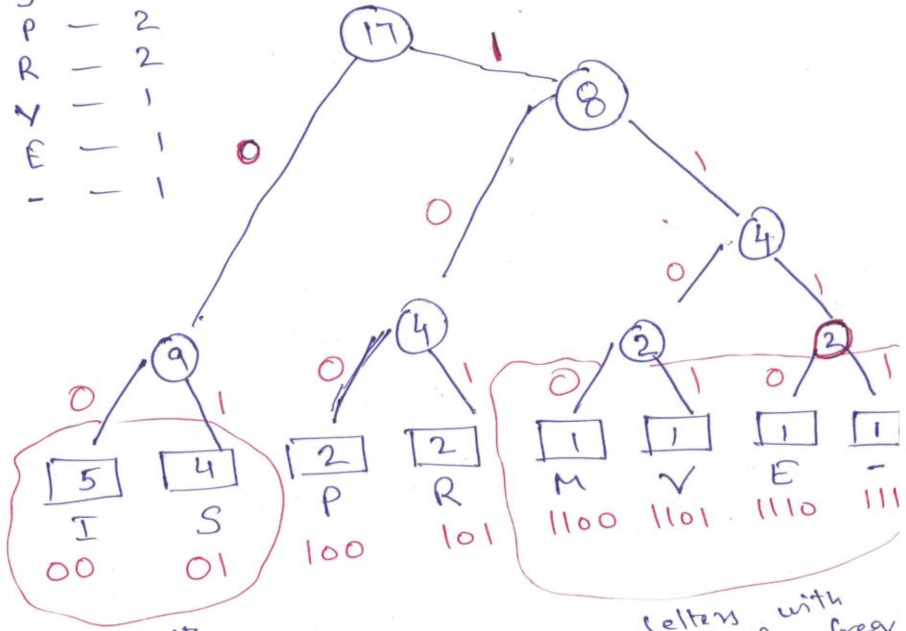
Table 1

M	I	S	S	I	S	S	I	P	P	I		R	I	V	E	R

Solution MISSISSIPPI RIVER

freaueny 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

M	1
I	5
S	4
P	2
R	2
V	1
E	1
-	1



Letters with greater frequency have shorter representation

Letters with ~~[illegible]~~ less freq have longer representation

No. of bits required before Compression = 17×8 = 136 bits

No. of bits required after Compression = 46 bits.

Compression Ratio = $\frac{136}{46} = \frac{2.956}{1}$

(OR)

in terms of % = $1 - \frac{1}{\frac{136}{46}} = \frac{136-46}{136} = \frac{90}{136}$

$= 66.17\%$

Job-Scheduling/Job Sequencing with deadline problem

- Job scheduling is the problem of scheduling jobs out of a set of N jobs on a single processor which maximizes profit as much as possible. Consider N jobs, each taking unit time for execution.
- Each job/task t_i is having some profit and deadline associated with it. Profit earned only if the job is completed on or before its deadline. Otherwise, we have to pay a profit as a penalty. Each job has deadline $d_i \geq 1$ and profit $p_i \geq 0$. At a time, only one job can be active on the processor.
- The job is feasible only if it can be finished on or before its deadline. A feasible solution is a subset of N jobs such that each job can be completed on or before its deadline.
- An optimal solution is a solution with maximum profit.
- In real life, job scheduling algorithms are widely used in the fields of optimal routing in networks.

Brute Force Approach

- The simple and inefficient solution is to generate all subsets of the given set of jobs and find the feasible set that maximizes the profit. For N jobs, there exist 2^N schedules, so this brute force approach runs in $O(2^N)$ time.

Job i	Profit p_i	Deadline d_i
J1	100	2
J2	10	1
J3	15	2
J4	20	1

Example 1

Job i	Profit p_i	Deadline d_i
1	10	2
2	20	1
3	25	2
4	50	1
5	40	3

Example 2

Job i	Profit p_i	Deadline d_i
1	10	2
2	20	1
3	25	2
4	50	1
5	40	3
6	30	3

Basic Approach

- Brute force approach is to generate all subsets of a given set of jobs and check individual subsets for the feasibility of jobs in that subset. Keep track of maximum profit among all feasible subsets.
- The greedy method involves sorting the jobs in decreasing order of their profit and then placing the jobs one by one in the free slot available just before the deadline so that we can have the space left for the jobs with lesser deadlines.

Algorithm

Step 1: Sort profit p_i into non-increasing order. After sorting $p_1 \geq p_2 \geq p_3 \geq \ldots \geq p_i$.

Step 2: Add the next job i to the solution set if i can be completed by its deadline. Assign i to time slot [r-1, r], where r is the largest integer such that $1 \leq r \leq d_i$ and [r-1, r] is free.

Step 3: Stop if all jobs are examined. Otherwise, go to step 2.

Example 3

Job i	p_i	d_i	
J1	20	2	
J2	15	2	
J3	10	1	
J4	5	3	
J5	1	3	

Try this!

Job i	d_i	p_i	
1	4	70	
2	2	60	
3	4	50	
4	3	40	
5	1	30	
6	4	20	
7	6	10	

Solution

Job i	d_i	p_i	
1	4	70	assign to [3, 4]
2	2	60	assign to [1, 2]
3	4	50	assign to [2, 3]
4	3	40	assign to [0, 1]
5	1	30	reject
6	4	20	reject
7	6	10	assign to [5, 6]

Solution = {4, 2, 3, 1, 7, 5, 6}

Total penalty incurred = 30 + 20= 50

Scheduling to Minimizing Lateness

- Given a set of n jobs all of which must be scheduled on a single resource (one job at a time) such that all jobs arrive at time s and each job has a deadline d_j and a length t_j, minimize the maximum lateness of the resulting schedule.
- We need to assign a start time s_j to each job j.
- Jobs are allowed to be late i.e., a job j may finish at time $f_j = s_j + t_j > d_j$.
- If job j is late, its lateness is defined to be $l_j = f_j – d_j$. Otherwise, $l_j = 0$.
- So the objective is to find a schedule that minimizes the maximum lateness, $L = \max(l_j)$

Greedy template

Select jobs according to some natural order. Which order minimizes the maximum lateness?

A. [shortest processing time] Ascending order of processing time t_j

B. [earliest deadline first] Ascending order of deadline d_j

C. [smallest slack] Ascending order of slack: $d_j – t_j$

D. None of the above

Greedy Algorithm- Earliest deadline first

```
EARLIEST-DEADLINE-FIRST (n, t[], d[]) {
    SORT jobs by deadline and renumber so that $d_1 \le d_2 \le \ldots \le d_n$
    t = 0;
    for j = 1 to n
    {
        Assign job j to interval $[t, t + t_j]$
        $s_j = t$;
        $f_j = t + t_j$;
        $l_j = \max(0, f_j - d_j)$
        $t = t + t_j$;
    }
    $L = \max(l_j)$
    Print intervals [s1, f1], [s2, f2], …, [sn, fn] and maximum lateness L
```

```
for j = 1 to n{
  Assign job j to interval [t, t + $t_j$ ]
  $s_j$ = t;
  $f_j$ = t + $t_j$;
  t = t + $t_j$;}
```

Example 1

Job j	J1	J2	J3	J4	J5	J6
Time t_j	3	2	1	4	3	2
Deadline d_j	6	8	9	9	14	15

Try this!

Job j	J1	J2	J3	J4	J5
Time t_j	3	2	1	4	2
Deadline d_j	4	7	8	8	10

Coin Change Problem

Given a set of n coins (infinite supply of each of the denominations) and a value, we have to find the minimum number of coins which satisfies the value. Assume that each coin's value is an integer.

Example 1

coins[] = {5,10,20,25}

value = 50

Total coins = 2 (25+25)

Example 2

coin[] = {25,20,10,5}

value = 70

Total coins needed = 3 (25+25+20).

Algorithm

```
COIN-CHANGE-GREEDY(coins, value)
{
    Sort the array of coins in decreasing order
    i = 0;
    while (value)
    {
        if coins[i] > value
            i++;
        else
        {
            print coins[i];
            value = value - coins[i];
        }
    }
}
```

Example

Coins = {1, 2, 5, 10, 20, 50, 100, 500}

n = 93

Analysis of the algorithm

As value (V) is decreased by coins[i] at the end of the while loop, we can say that for most of the cases it will take much less than O(V) time.

So, we can say that our algorithm has a O(V) running time.

Issue with Greedy Algorithm Approach

For example, consider this denominations.

coins = {1, 5, 6, 9}

If we have to reach a sum of 11, the greedy algorithm will end up the denomination 9, 1, 1 i.e. 3 coins to reach the value of 11.

However, there is a more optimal solution. And that is by using the denominations 5 & 6. Using them, we can reach 11 with only 2 coins.

For general input we will use Dynamic Programming.