Greedy Technique.

Consider the situation,

There is very intelligent guy who knows programming. He is very rich & has an infinite supply of 20$, 10$ & 5$ coins

Now he has a friend who needs some money (₹35) & that too in minimum no of coins. He has a Greedy choice

① 7 * $5 ② 10$ + 5 * 5$ ③ $20+5+5+5 . & soon

And he got a final solution.

he has to give a coin of largest value that does not take him past

for $35, Largest denomination $20

Now $15 are short to the amount so the next largest he can give is 10$. & he gave 1 5$ coin

Note:- At each iteration, add coin of the largest value that does not take us past the amount to be paid

Definition of Greedy Algorithm:-

An algorithmic paradigm that follows the problem solving approach of making the locally optimal choice at each stage with the hope of finding a global optimum

Pros:- Simple easy to implement, run fast.

Cons:- very often, they don't provide a globally optimum solution

Properties of Greedy:-

1) **Greedy Choice property** :- A global optimum can be arived at by selecting a local optimum

2) **Optimal Substructure** :- An optimal solution of the problem contains an optimal solution to subproblems

Applications:-

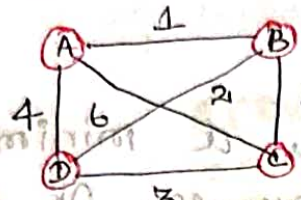
1) Prim's Algorithm
2) Kruskal's Algorithm } → for finding minimum Spanning Tree
3) Huffman Coding
4) 0/1 Knapsack problem
5) Container Loading problem

Ms.S.T. SHENBAGAVALLI, M.E.,
Assistant Professor

PRIM's ALGORITHM

Definition :-

It is an algorithm used to find a minimum cost Spanning tree for Connected weighted undirected graph.



Connected Weighted Undirected ⇒

Minimum Spanning tree

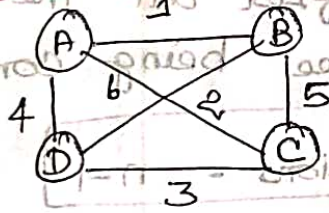
Definition of Spanning tree :-

A Spanning tree of a connected graph is its connected acyclic sub graph (i.e., a tree) that contains all the vertices of a graph.

Definition of Minimum Spanning tree :-

A Minimum Spanning tree of a weighted connected graph is its Spanning tree of the smallest weight.

Brute force Approach to Solve Prim's Algorithm :-



There are 2^n possibilities.

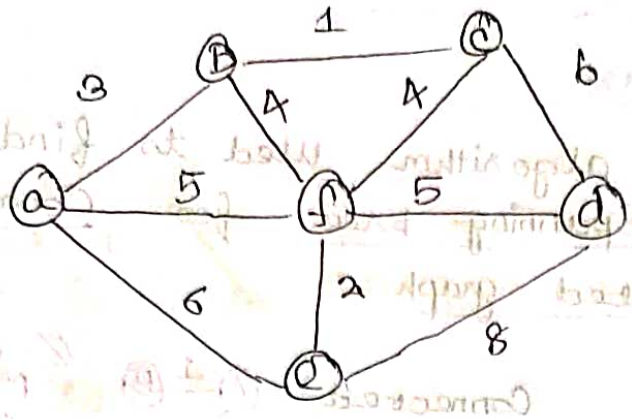
∴ 16 possibilities. &
List all the possibilities & Choose the smallest.

Drawback :-

1) Number of Spanning trees grows exponentially with the input size.

Ms.S.T. SHENBAGAVALLI, M.E.,
Assistant Professor

Prim's Algorithm example:-



Procedure:-

① Prim's algorithm constructs a minimum spanning tree through a sequence of expanding subtrees.

2. The initial subtree in such a sequence consists of a single vertex selected arbitrarily from the set V of the graph's vertices.

3. On each iteration, we expand the current tree in the greedy manner by simply attaching to it the nearest vertex not in that tree. (which is having smallest weight)

4. The algorithm stops after all the graph's vertices have been in the tree being constructed.

Total number of iterations = n-1

Important Note:-

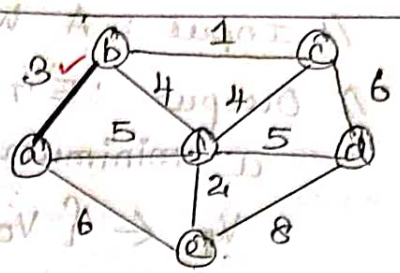
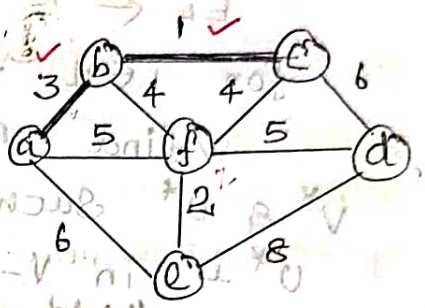
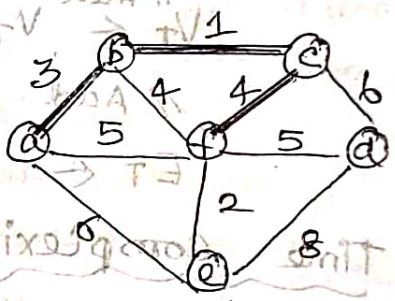
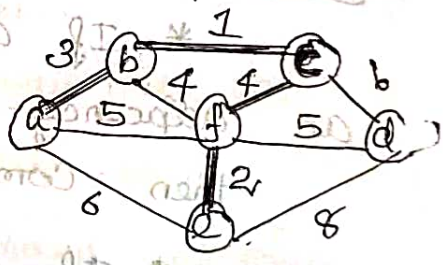
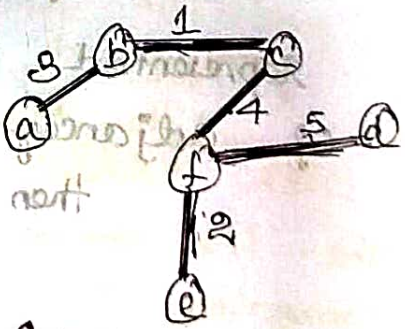
* The Aim of this algorithm to provide each vertex not in the current tree with the information about the shortest edge connecting the vertex to a tree vertex.

* It will be done by 2 Labels.

1) Name of the nearest vertex

2) The length of the corresponding edge

* Vertices that are not adjacent mentioned by $(-, \infty)$

Tree vertices	Remaining vertices	Illustration
a(-,-)	**b(a,3)**, c(-,∞), d(-,∞), e(a,6) f(a,5)	
b(a,3)	**c(b,1)**, d(-,∞), e(-,∞) f(b,4)	
c(b,1)	d(c,6), e(-,∞), **f(b,4)**	
f(b,4)	**d(f,5)**, **e(f,2)**	
e(f,2)	d(f,5) d(e,8)	
d(f,5)		Cost of MST ⇒ 3+1+4+5+2

* Bold letters are shows the selected Vertices & edges.

Algorithm Of PRIM's :-

Algorithm Prim(G)

// Prim's algorithm for constructing a minimum spanning tree

// Input : A Weighted Connected graph $G\{V, E\}$

// Output : E_T the set of edges composing a minimum spanning tree of G.

$V_T \leftarrow \{v_0\}$ set of visited vertices.

$E_T \leftarrow \phi$

for $i \leftarrow 1$ to $|V| - 1$ do

// find minimum edge 'e*' between Vertices v^* & u^* such that v^* is in V_T & u^* is in $V - V_T$

// Add u to V_T

$V_T \leftarrow V_T \cup \{u^*\}$

// Add the edge to the spanning tree

$E_T \leftarrow E_T \cup \{e^*\}$

Time Complexity :

* If a graph is represented as an adjacency matrix, then complexity is V^2. i.e two for loops for go through the matrix

* If a graph is represented as binary heap & in the form of adjacency list, then the time complexity = $O|E| \log V$

E → Edges V = Vertices

Ms. S. T. SHENBAGAVALLI, M.E.,
Assistant Professor

Kruskal's Algorithm

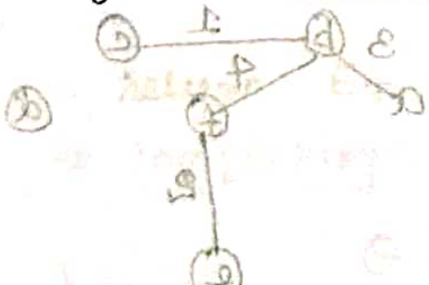
* It is used to find Minimum Spanning tree.
* This is discovered by 2nd year graduate student Joseph kruskal.
* In this algorithm always the minimum cost edge has to be selected.
* But it is not necessary that selected optimum edge is adjacent.

Differences blw Prim's & kruskal's Algorithm:

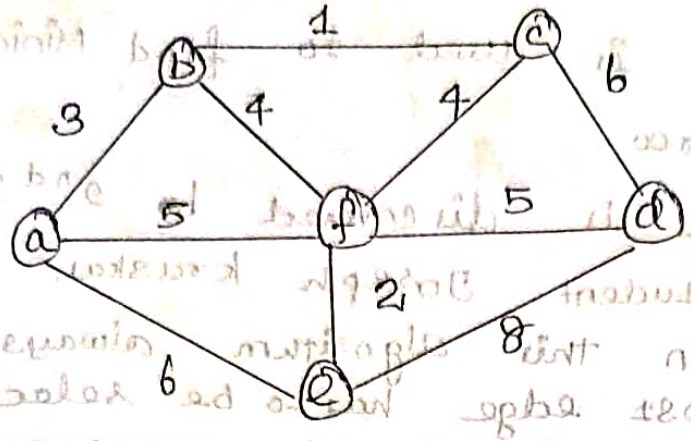
Prim's Algorithm	kruskal's Algorithm
Obtaining Minimum Spanning tree by selecting the adjacent vertices of already selected vertices	It is not necessary that selected optimum edge is adjacent.

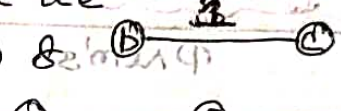
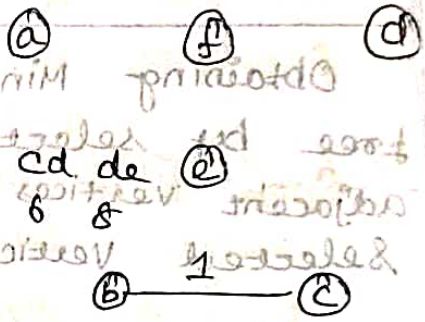
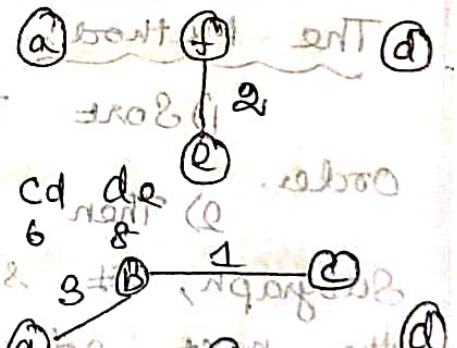
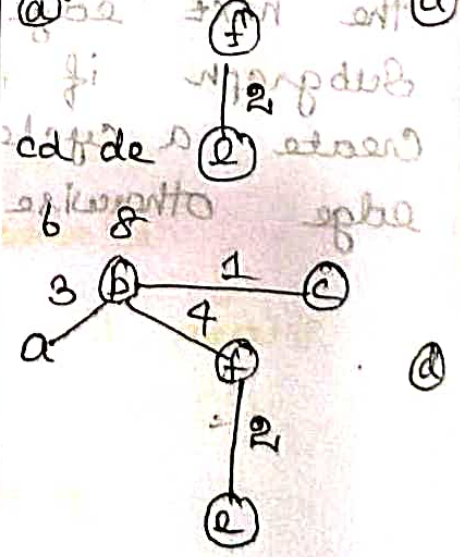
The Method:-

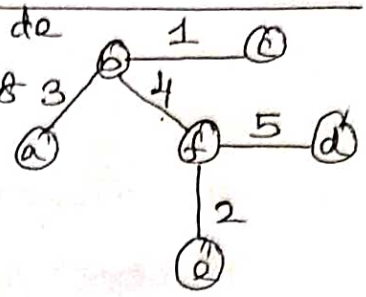
1) Sort the edges in non decreasing order.
2) Then starting with the empty subgraph, it scans this sorted list, adding the next edge on this list to the current subgraph if such an inclusion does not create a cycle and simply skipping the edge otherwise.



Example for kruskal Algorithm:-



Tree edges	Sorted List of edges	Illustration
	bc fe ab bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
bc 1	bc fe ab bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
fe 2	bc fe ab bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	
ab 3	bc fe ab bf cf af df ae cd de 1 2 3 4 4 5 5 6 6 8	

Tree edges	Sorted List of edges	Illustration
bf 4	bc fe ab bf ef af df ae cd de 1 2 3 4 4 5 5 6 6 8	
df 5	Cost of MST = 15	

Algorithm for kruskal :-

ALGORITHM KRUSKAL(G)

// kruskal's algorithm for constructing a MST

// Input: A weighted connected graph $G = \{V, E\}$

// Output: E_T, the set of edges composing a Minimum Spanning Tree of G

Sort E in nondecreasing order of the edge weights $w(e_{i_1}) \leq \ldots \leq w(e_{i_{|E|}})$

$E_T \leftarrow \phi$; // no edges selected

ecounter $\leftarrow 0$ // no edges selected (set of edges)

$k \leftarrow 0$ // Initialize the number of processed edge.

While ecounter $< |V| - 1$ do

$k \leftarrow k + 1$

if $E_T \cup \{e_{i_k}\}$ is acyclic

$E_T \leftarrow E_T \cup \{e_{i_k}\}$;

ecounter $\leftarrow$ ecounter $+ 1$;

return E_T.

Time complexity :-

$O(|E| \log |E|)$

E = Total No of edges

Huffman Coding

Greedy Algorithms | Set 3 (Huffman Coding)

Definition

Huffman coding is a lossless data compression algorithm. The idea is to assign variable-legth codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code and the least frequent character gets the largest code.

The variable-length codes assigned to input characters are Prefix Codes, means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not 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.

Let us understand prefix codes with a counter example. Let there be four characters a, b, c and d, and their corresponding variable length codes be 00, 01, 0 and 1. This coding leads to ambiguity because code assigned to c is prefix of codes assigned to a and b. If the compressed bit stream is 0001, the de-compressed output may be "cccd" or "ccb" or "acd" or "ab".

See this for applications of Huffman Coding.

There are mainly two major parts in Huffman Coding

1) Build a Huffman Tree from input characters.

2) Traverse the Huffman Tree and assign codes to characters.

Steps to build Huffman Tree

Input is array of unique characters along with their frequency of occurrences and output is Huffman Tree.

1. Create a leaf node for each unique character and build a min heap of all leaf nodes (Min Heap is used as a priority queue. The value of frequency field is used to compare two nodes in min heap. Initially, the least frequent character is at root)

2. Extract two nodes with the minimum frequency from the min heap.

3. Create a new internal node with frequency equal to the sum of the two nodes frequencies. Make the first extracted node as its left child and the other extracted node as its right child. Add this node to the min heap.

4. Repeat steps#2 and #3 until the heap contains only one node. The remaining node is the root node and the tree is ~~complete~~ Complete

Huffman Trees

We can use a fixed length encoding that assigns to each character a bit string of the same length m ($m \geq \log_2 n$)

* Using a variable-length encoding which assigns code words of different lengths to different characters, introduces a problem that fixed-length encoding does not have.

* In a prefix code, no code word is a prefix of a code word is of another character. Hence with such an encoding, we can simply scan a bit string until we get the first group of bits that is a code word for some character, replace these bits by this character, and repeat this operation until the bit string's end is reached.

* Huffman Algorithm

Step 1: Initialize n one-node trees and label them with the characters of the alphabet. Record the frequency of each character in its tree's root to indicate

the trees weight.

Step 2: Repeat the following operation until single tree is obtained. Find two trees with the smallest weight. Make them the left and right subtree of a new tree and record the sum of their weights in the root of the new tree as its weight.

* A tree constructed by the above algorithm is called a Huffman tree. It defines in the manner described - a Huffman code.

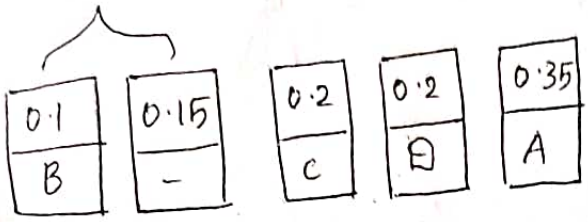
EXAMPLE: Consider the five-character alphabet {A, B, C, D, -} with the following occurrence probabilities.

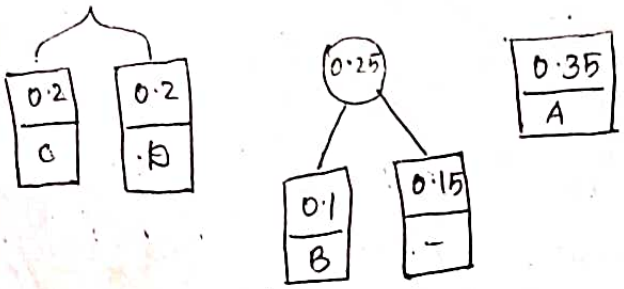
character	A	B	C	D	-
probability	0.35	0.1	0.2	0.2	0.15

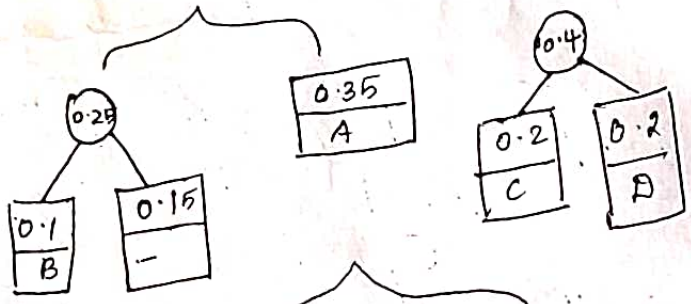
The Huffman tree construction for this input. The resulting code words are as follows:

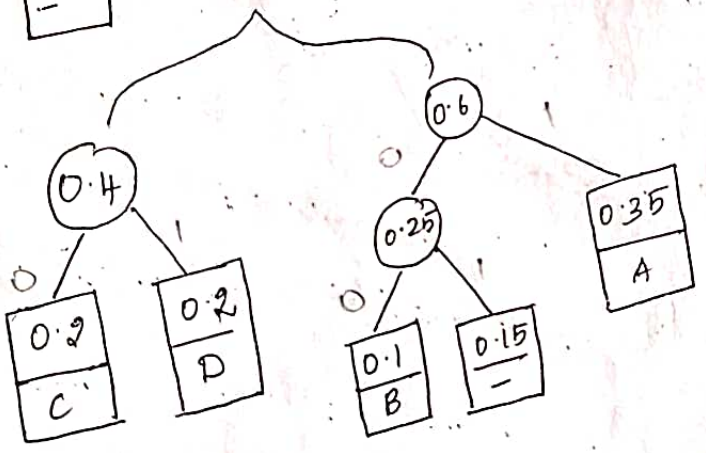
character	A	B	C	D	-
Probability	0.35	0.1	0.2	0.2	0.15
code word	11	100	00	01	101

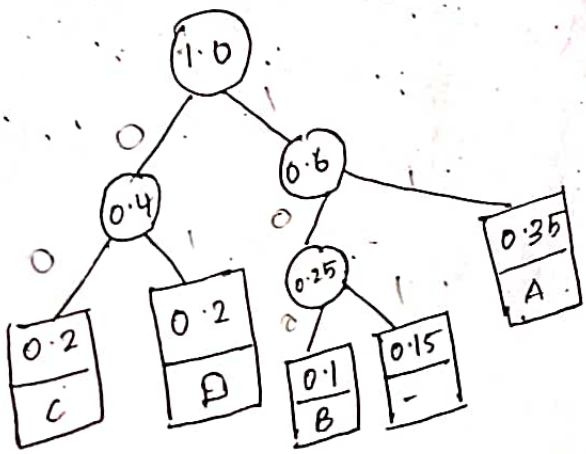
Hence DAD is encoded as 011101 and 10011011011101 is decoded as BAD_AD











with the occurrence probabilities given and the codeword length obtained the expected number of bits per character in this code is

$2 \cdot 0 \cdot 35 + 3 \cdot 0 \cdot 1 + 2 \cdot 0 \cdot 2 + 2 \cdot 0 \cdot 2 + 3 \cdot 0 \cdot 15 = 2 \cdot 25$

* Huffman's code achieve the compression ratio, a standard measure of a compression algorithm's effectiveness, of $(3 - 2 \cdot 25)/3 \cdot 100\% = 25\%$.

* In otherwords we should expect the huffman's encoding of a text will use 25% less memory than its fixed length encoding.

* Huffman's encoding is one of the most important file compression methods. In addition to its simplicity and versatility it yields an optimal, i.e., minimal - length encoding

* The simplest version of huffman compression calls in fact, for a preliminary scanning of a given text to count the frequencies of character occurrences in it

* suppose we have n positive numbers $w_1, w_2 \ldots w_n$. which have to be assigned to n leaves of a binary tree one per node. If we define the weighted path length as the sum $\sum_{i=1}^{n} l_i w_i$ where l_i is the length of the simple Path from the root to the i^{th} leaf.

0/1 knapsack Problem

Items Greedy Method

0 → not selected 1 → item is selected

Objects	1	2	3	4	5	6	7
Profit	10	5	15	7	6	18	3
weight	2	3	5	7	1	4	1

knapsack weight $m = 15$

$n = 7$

Formula:

maximize $\sum_{1 \le i \le n} p_i x_i$

Constraint $\sum_{1 \le i \le n} w_i x_i \le m$

Obj	1	2	3	4	5	6	7
Profit	10	5	15	7	6	18	3
weight	2	3	5	7	1	4	1
P/w	5	1.3	3	1	6	4.5	3

Take maximum P/w value P/w = 5 Corresponding items

1) item 5 {0, 0, 0, 0, 1, 0, 0}
P/w = 6 weight 15 - 1 = 14 → remaining Capacity

2) item 1 {1, 0, 0, 0, 1, 0, 0}
P/w = 5 weight 14 - 2 = 12,

3) item 6 {1, 0, 0, 0, 1, 1, 0}
P/w = 4.5 weight 12 - 4 = 8,

4) item 3 {1, 0, 1, 0, 1, 1, 0}

P/w = 3

8 - 5 = 3 //

5) item 7 {1, 0, 1, 0, 1, 1, 1}

P/w = 3

3 - 1 = 2

6) item 2

2 - 3 item 2 is not taken

7) item 4 2 - 7 item 4 is not taken

items taken

5, 1, 6, 3, 7

Arrange in ascending order 1, 3, 5, 6, 7

Profit

$\sum_{1 \le i \le n} P_i x_i = 10 \times 1 + 15 \times 1 + 6 \times 1 \times 18 \times 1 \times 3 \times 1$

$\Rightarrow 10 + 15 + 6 + 18 + 3$

Profit = 52 //

weight $\sum_{1 \le i \le n} w_i x_i$ = ~~2 x 1 + 5 x 1 + 1 x 1 + 4 x~~

$(2 \times 1) + (5 \times 1) + (1 \times 1) + (4 \times 1) + (1 \times 1)$

$\Rightarrow 2 + 5 + 1 + 4 + 1$

weight = 13 //