Greedy Algorithms:

Greedy algorithms solve problems by making the choice that seems best at the particular moment. Many optimization Problems can be solved using a greedy algorithm. Some problems have no efficient solution, but a greedy algorithm may provide a solution that is close to optimal.

AN ACTIVITY-SELECTION PROBLEM:

Our first Example is the problem of scheduling a resource among Several Competing activities.

Suppose $S = \{1, 2, \ldots n\}$ is the set of n proposed activities. The activities share a resource, which can be used by only one activity at a time e.g. a Tennis Court, a lecture Hall etc.

Each activity i has a start time s_i and a finish time f_i, where $s_i < f_i$ ie. they do not overalap.

The activity-selection problem selects the maximum size set of mutually compatible activities → choose the maximum set of activities using Greedy choice algorithm.

If we will select the activites in increasing order of their finish time ie.

$$f_1 \le f_2 \le f_3 \le \ldots \le f_n$$

The running time of algorithm GREEDY-ACTIVITY-SELECTOR is $\Theta(n \log n)$ as sorting can be done in $O(n \log n)$.

GREEDY-ACTIVITY-SELECTOR (S, f)

```
1. n ← length[S]
2. A ← {1}
3. J ← 1
4. for I ← 2
5.     do if si ≥ fj
6.         then A ← A U {i}
7.             J ← i
8. return A.
```

Example: Given 10 activities along with their start and finish time as

$S = (A_1, A_2, A_3, A_4, A_5, A_6, A_7, A_8, A_9, A_{10})$

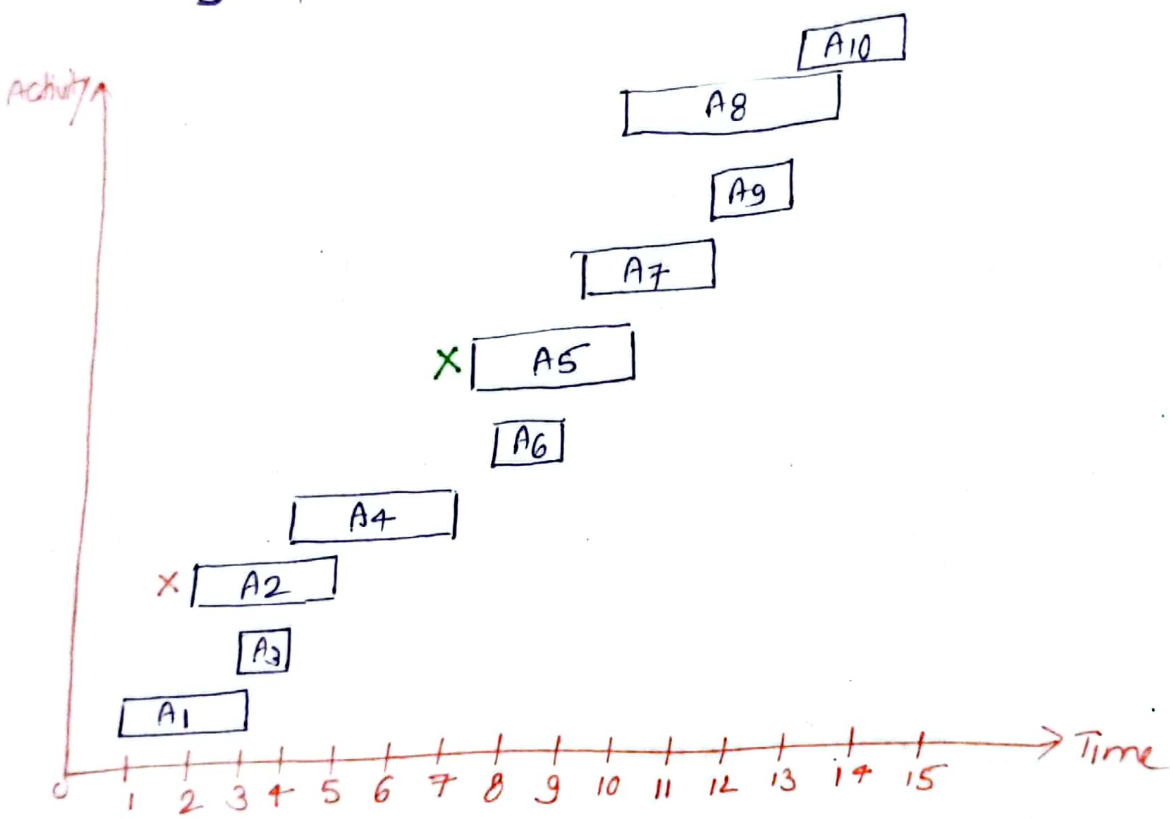
$s_i = (1, 2, 3, 4, 7, 8, 9, 9, 11, 12)$

$f_i = (3, 5, 4, 7, 10, 9, 11, 13, 12, 14)$

Computes a schedule where the largest number of activities takes place

Sol^n Arranging the activities in increasing order of finish time

Activity	A_1	A_3	A_2	A_4	A_6	A_5	A_7	A_9	A_8	A_{10}
Start	1	3	2	4	8	7	9	11	9	12
Finish	3	4	5	7	9	10	11	12	13	14



Thus, the final activity schedule is

$(A_1, A_3, A_4, A_6, A_7, A_9, A_{10})$

Scanned with CamScanner

KNAPSACK PROBLEMS:

We want to pack n items in your luggage.

- → The ith item is worth v_i dollars and weights w_i pounds
- → Take as valuable a load as possible, but cannot exceed W pound
- → v_i, w_i, W are integers.

0-1 Knapsack problem:

→ Each item is taken or not taken

False (Not taken) True (Taken)

→ Cannot take a fractional amount of an item or take an item more than once.

Fractional Knapsack Problem:

→ fractions of items can be taken rather than having to make a binary (0-1) choice for each item.

Fractional Knapsack (Array V, Array ω, int W)

```
for i = 1 to Size(V)
      do P[i] = V[i]/ω[i]
Sort : Descending(P)
i ← 1
while (W>0)
      do amount = min(W, ω[i])
      solution[i] = amount
      W = W - amount
      i ← i+1
return solution
```

Example: Consider 5 items along their respective weights and values

$I = (I_1, I_2, I_3, I_4, I_5)$

$\omega = (5, 10, 20, 30, 40)$

$V = (30, 20, 100, 90, 160)$

The capacity of Knapsack W = 60. Find the solution to the fractional knapsack problem.

Solution:

Item	w_i	v_i
I_1	5	30
I_2	10	20
I_3	20	100
I_4	30	90
I_5	40	160

Taking value per weight ratio i.e. $p_i = v_i/w_i$

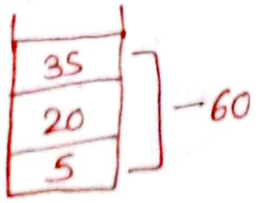
Item	w_i	v_i	$p_i = v_i/w_i$
I_1	5	30	$30/5 = 6$
I_2	10	20	$20/10 = 2$
I_3	20	100	$100/20 = 5$
I_4	30	90	$90/30 = 3$
I_5	40	160	$160/40 = 4$

Now arrange the value of p_i in decreasing order

Item	w_i	v_i	$p_i = v_i/w_i$
I_1	5	30	6
I_3	20	100	5
I_5	40	160	4
I_4	30	90	3
I_2	10	20	2

First we choose item I_1 whose weight is 5, then choose item I_3 whose weight is 20. Now we total weight in knapsack is $5+20=25$

Now, the next item is I_5 and its weight is 40, but we want only 35, so we choose fractional part of it ie.



The value of fractional part of I_5 is

$$\frac{160}{40} \times 35 = 140$$

Thus the maximum value is $= 30 + 100 + 140 = 270$

Huffman CODES:-

Data can be encoded Efficiently using Huffman codes. It is a widely used and very Effective technique for compressing data. Huffman's greedy algorithm uses a table of the frequencies of occurrence of Each character to build up an optimal way of representing Each character as a binary string. The total running time of Huffman on a set of is $O(n \log n)$.

(I) Fixed Length code: Each letter represented by an equal number of bits. With a fixed length code, at least 3 bits per character.

(II) Variable-length code can do considerably better than a fixed-length code, by giving frequent characters short code words and infrequent characters long code words.

$$\text{Number of bits} = \sum_{i=0}^{n} q_i d_i$$

Where $(q_1 d_1 + q_2 d_2 + q_3 d_3 + q_4 d_4 + q_5 d_5) * 1000$

The algorithm builds the Tree T corresponding to the optimal code in a bottom-up manner.

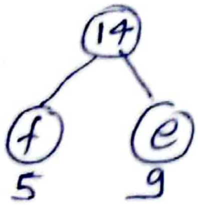
Example Suppose only 6 characters appear in the file:

	a	b	c	d	e	f	Total
Frequency	45	13	12	16	9	5	100

How can we represent the data in a compact way?

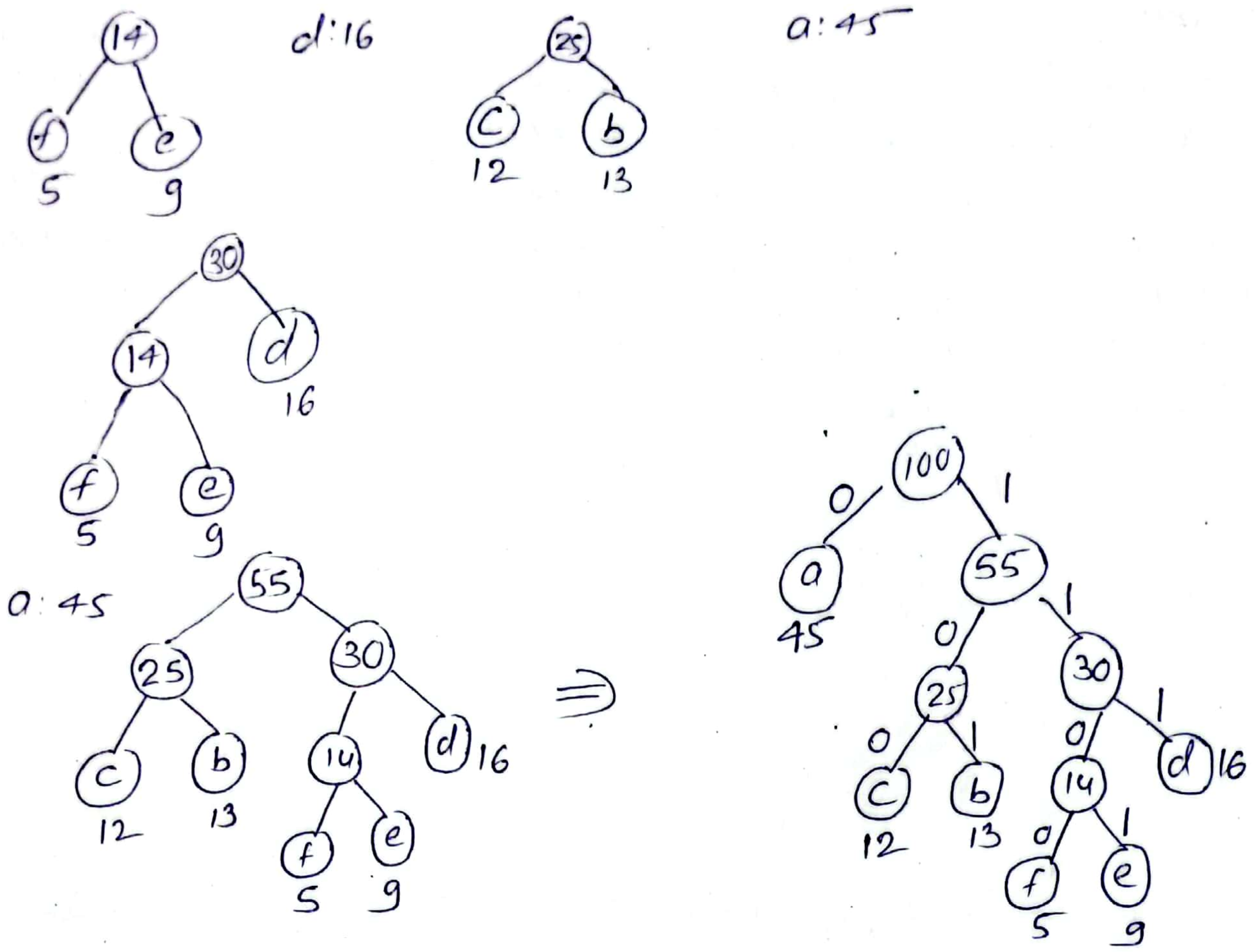
Huffman code or Tree is made by MIN HEAP. Extract two nodes with the minimum frequency from the min heap.

f:5 e:9 c:12 b:13 d:16 a:45



character	frequency
c	12
b	13
internal node	14
d	16
a	45

Scanned with CamScanner



$a = 0$

$b = 101$

$c = 100$

$d = 111$

$e = 1101$

$f = 1100$

total Number of bits:

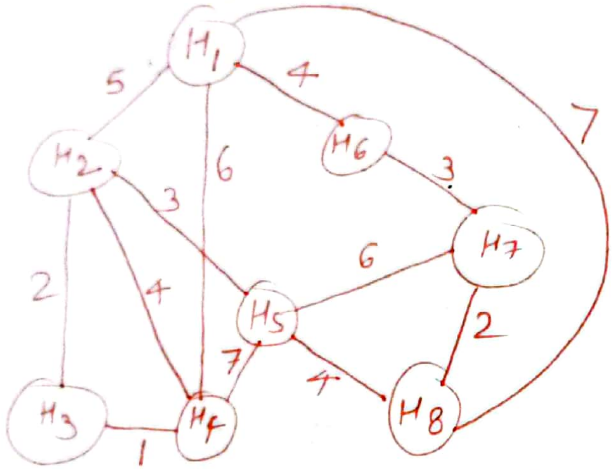
$$= (45 \times 1 + 13 * 3 + 12 * 3 + 16 * 3 + 9 * 4 + 5 * 4) * 1000$$

$$= 224{,}000 \text{ bits/msg} = 2.24 * 10^5 \text{ bits.}$$

Travelling Sales Person Problem:

In this problem a salesman need to visit n cities in such a manner that all cities must be visited at once and in the end he returns to the city from where he started with minimum cost.

Example: A newspaper agent daily drops the newspaper to the area assigned in such a manner that he has to cover all the houses in the respective area with minimum travel cost. Compute the minimum travel cost. The area assigned to the agent where he has to drop the newspaper is shown in the figure.



Sol: The cost-adjacency matrix of graph G is as follows: $Cost_{ij} =$

	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8
H_1	0	5	0	6	0	4	0	7
H_2	5	0	2	4	3	0	0	0
H_3	0	2	0	1	0	0	0	0
H_4	6	4	1	0	7	0	0	0
H_5	0	3	0	7	0	0	6	4
H_6	4	0	0	0	0	0	3	0
H_7	0	0	0	0	6	3	0	2
H_8	7	0	0	0	4	0	2	0

So, using the greedy strategy we get the following:

$H_1 - H_6 - H_7 - H_8 - H_5 - H_2 - H_3 - H_4 - H_1$

$4 + 3 + 2 + 4 + 3 + 2 + 1 + 6 = 25$

Thus the minimum travel cost: 25

"A" EXTRAC-MIN(Q) {A}

Relax all edges leaving A {A}

'C' EXTRACT-MIN(Q) = {A, C}

Relax all edges leaving C - {A, C}

"E" EXTRACT_MIN(Q) = {A, C, E}

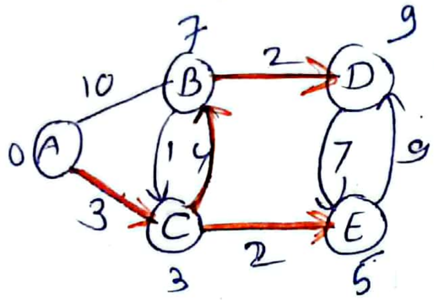
Relax all edges leaving E : {A, C, E}

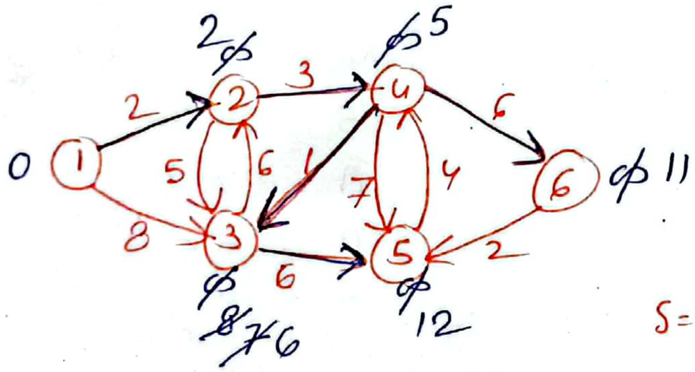
"B" EXTRACT_MIN(Q) : {A, C, E, B}

Relax all edges leaving B: {A, C, E, B}

"D" EXTRACT-MIN(Q) S={A, C, E, B, D}

Q: A	B	C	D	E
0	∞	∞	∞	∞
	10	3	-	-
	7		11	5
	7		11	
			9	



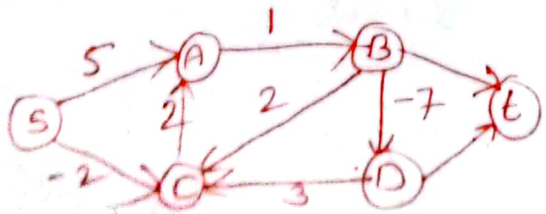


S = {1, 2, 4, 3, 5, 6}

Solve the shortest path problems using Dijkstra's algorithm count the number of distance undates.

BELLMAN - FORD ALGORITHM:

If a Graph G = (V, E) Contains a negative weight cycle, then some shortest paths may not Exist.

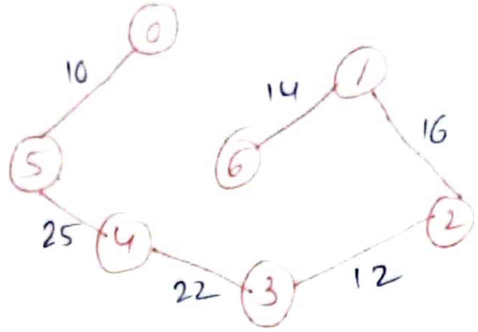


Negative weight cycle.

~~By EXTRACT-MIN() remove 6 because key(6) = 1~~

Now becomes Q contains only one vertex 6. By EXTRACT_MIN remove it.

Thus the final spanning tree is



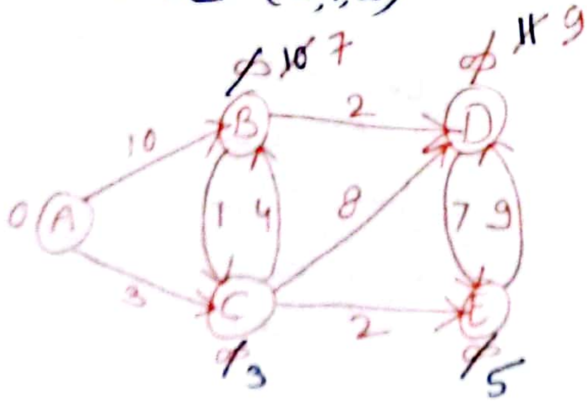
Singe-Source Shortest Paths:-

Dijkstra's Algorithm:

Dijkstra's algorithm maintains a set of S of vertices whose final shortest-path weights from the source s have already been determined.

```
DIJKSTRA(G,w,s)
INITIALIZE-SINGLE-SOURCE(G,s)
S ← φ
Q ← V[G]
while Q ≠ φ
     do u ← EXTRACT-MIN(Q)
        S ← S U {u}
        for Each vertex v ∈ Adj[u]
            do RELAX(u,v,w)
```

Example:

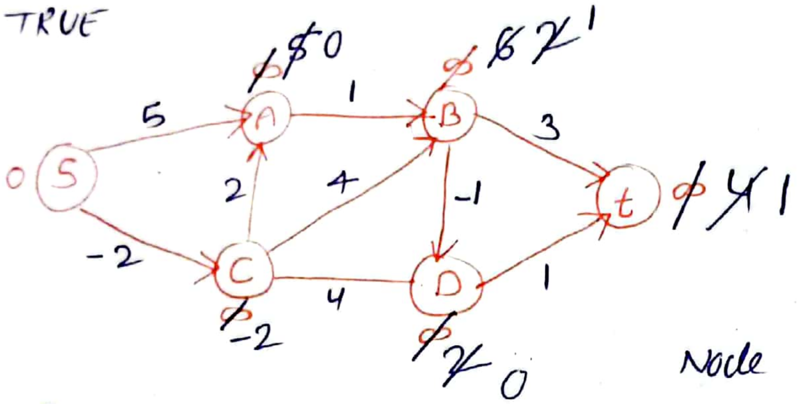


Q: A	B	C	D	E
0	∞	∞	∞	∞

Bellman-Ford algorithm finds all shortest-path lengths from a source $s \in V$ to all $v \in V$ or determine that a negative-weight cycle exists.

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i <- 1 to |V[G]| - 1
      do for each edge (u,v) ∈ E[G]
            do RELAX(u, v, w)
for each edge (u,v) ∈ E[G]
      do if d[v] > d[u] + w(u,v)
            then return FALSE
return TRUE
```

Example



$S - A = 5$

$S - C = -2$

$A - B = 1$

$C \to D = 4$

$C \to A = 2$

$C \to B = 4$

$B \to D = -1$

$B \to t = 3$

$D - t = 1$

Node		Path
S	0	
A	~~∅~~ ~~5~~ 0	A
B	~~∅~~ ~~6~~ ~~2~~ 1	B
C	~~∅~~ -2	C
D	~~∅~~ ~~2~~ 0	D
t	~~∅~~ ~~4~~ 1	t