Huff man Codes

Huffman codes compress the data very effectively. Depending on the characteristics of data being compressed, savings of 20% to 90% are typical. Considering data to be a sequence of characters, Huffman's greedy algorithm uses a table giving how often a character occurs to build up an optimal way of representing each character as a binary string.

Suppose we have 10^5 characters in data file. Suppose only 6 characters appear in the file.

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

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

a	b	c	d	e	f
000	001	010	011	100	101

Space required for 10^5 characters $= 3 \times 10^5$ bits

(ii) Variable length Code : Gives frequent characters short code words and infrequent characters long code words.

a	b	c	d	e	f
0	101	100	111	1101	1100

No. of bits $= (45 \times 1 + 13 * 3 + 12 \times 3 + 16 \times 3 + 9 * 4 + 5 * 4) * 1000$

$= 2.24 \times 10^5$ bits

Memory space approximately saved = 25%

Huffman Codes

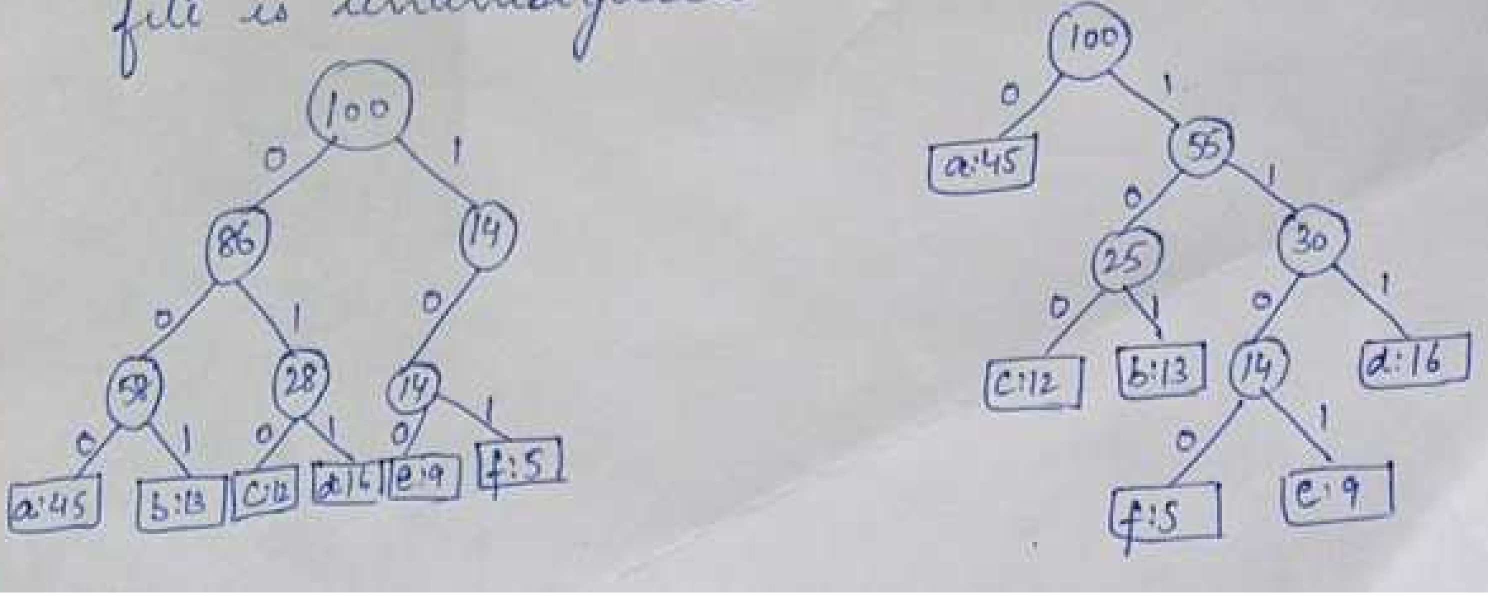
	a	b	c	d	e	f
frequency	45	13	12	16	9	5
fixed length codeword	000	001	010	011	100	101
Variable length codeword	0	101	100	111	1101	1100

Prefix codes

In huffman codes, we consider only the codes in which no codeword is also a prefix of some other code word. Such codes are called prefix codes.

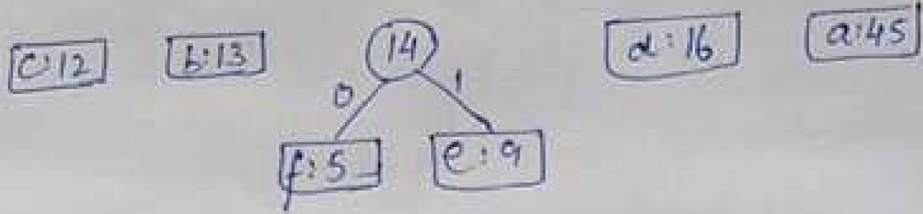
3 character code 'abc' will be encoded as "0·101·100 = 0101100" where '·' denotes concatenation.

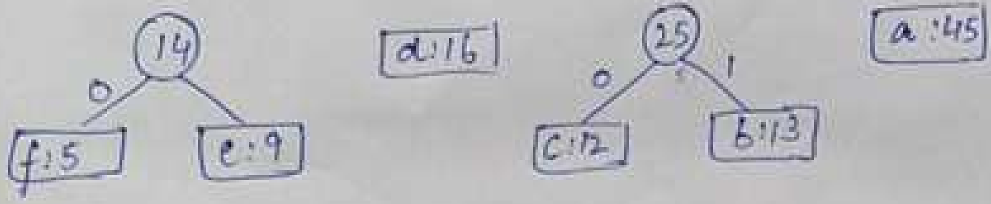
Prefix codes are desirable because they simplify decoding. Since no codeword is a prefix of any other, the codeword that begins an encoded file is unambiguous.

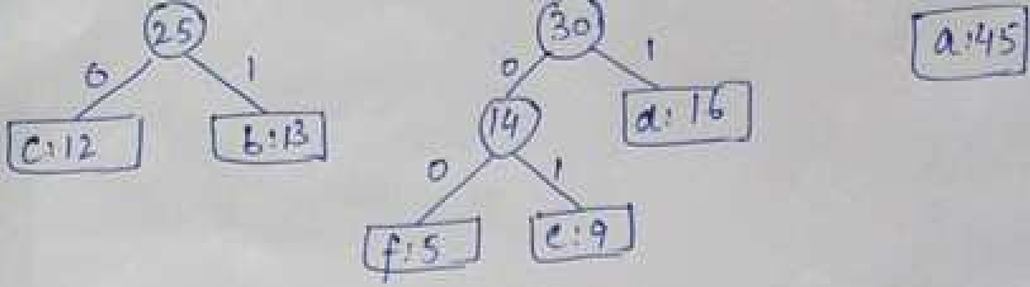


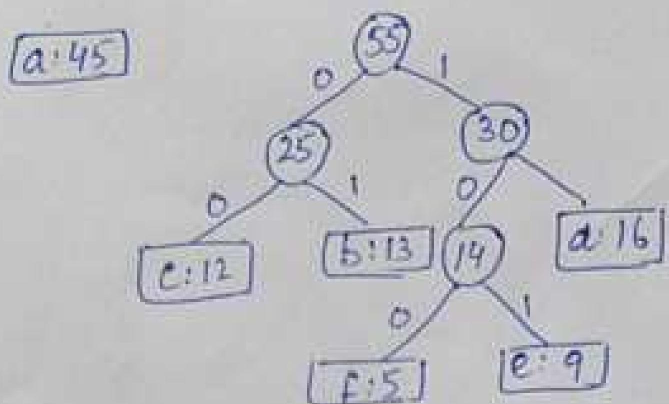
Constructing huffman code

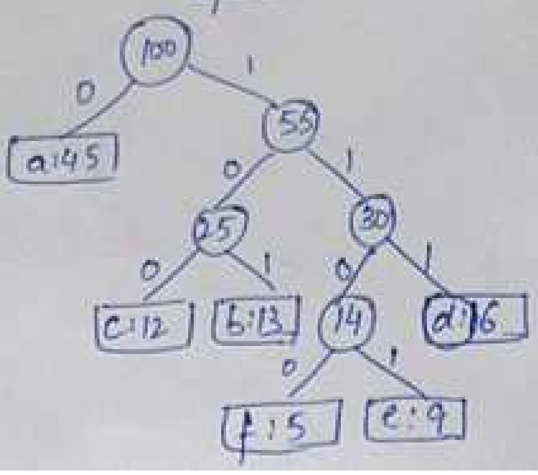
① 

② 

③ 

④ 

⑤ 

⑥ 

Let C be the set of n characters and each character $c \in C$ is an object with attribute $c.freq$ giving its frequency. The algorithm builds the tree T corresponding to the optimal code in bottom-up manner.

```
HUFFMAN (C)
  n = |C|
  Q = C
  for i = 1 to n-1
        allocate a new node z
        z.left = x = EXTRACT-MIN (Q)
        z.right = y = EXTRACT-MIN (Q)
        z.freq = x.freq + y.freq
        INSERT (Q, z)
  return EXTRACT-MIN (Q)
```

Single source shortest path: Given a graph $G = (V, E)$, we want to find a shortest path from a given source vertex $s \in V$ to every vertex $v \in V$.

Dijkstra's Algorithm: Dijkstra is a greedy algorithm that solves the single source shortest path problem for a directed graph $G = (V, E)$ with non-negative edge weights.

```
                      source
Dijkstra (G, w, s)
{  INITIALIZE_SINGLE_SOURCE (G, s)
   S ← φ
   Q ← V[G]
   while (Q ≠ φ)
   {  u ← Extract_min (Q)
      S ← S U {u}
      for (each vertex v ∈ Adj[u])
      {  RELAX (u, v, w)
      }
   }
}
```

INITILIZE-SINGLE-SOURCE (G, S)

{ for each vertex $V \in V[G]$

{ $d[V] \leftarrow \infty$

$\pi[V] \leftarrow$ NIL

}

$d[S] \leftarrow 0$

}

{$\pi[V]$ means Predecessor of V}

RELAX (U, V, w)

{ if $(d[V] > d[u] + w(u,v))$

{ $d[V] \leftarrow d[u] + w(u,v)$

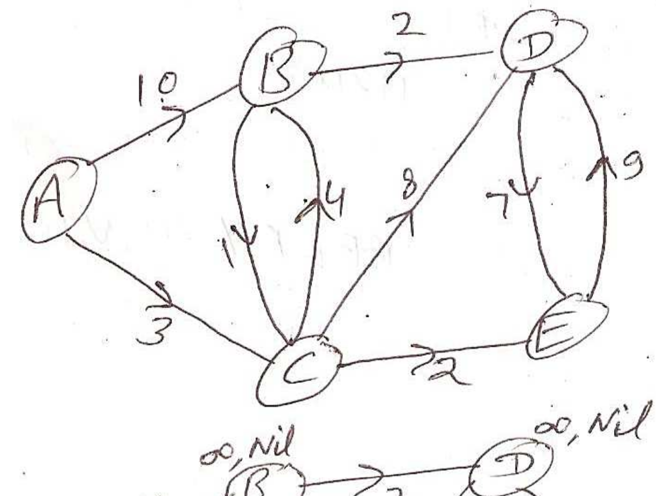
$\pi[V] \leftarrow u$

}

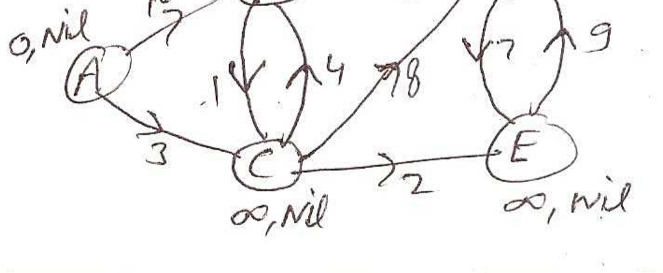
}

Time complexity $= O(V^2)$

Example:



initialize:

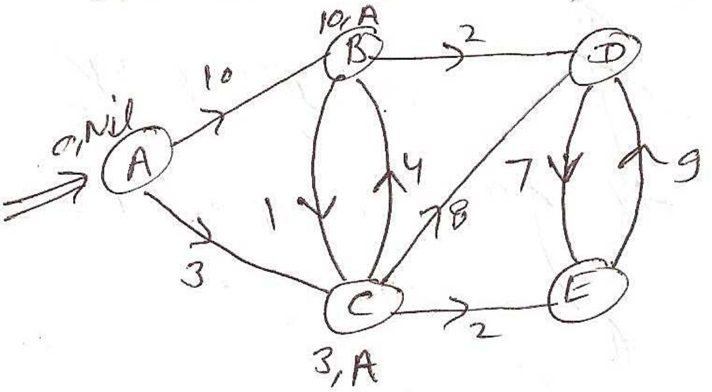


Q: A B C D E

0 ∞ ∞ ∞ ∞

S: { }

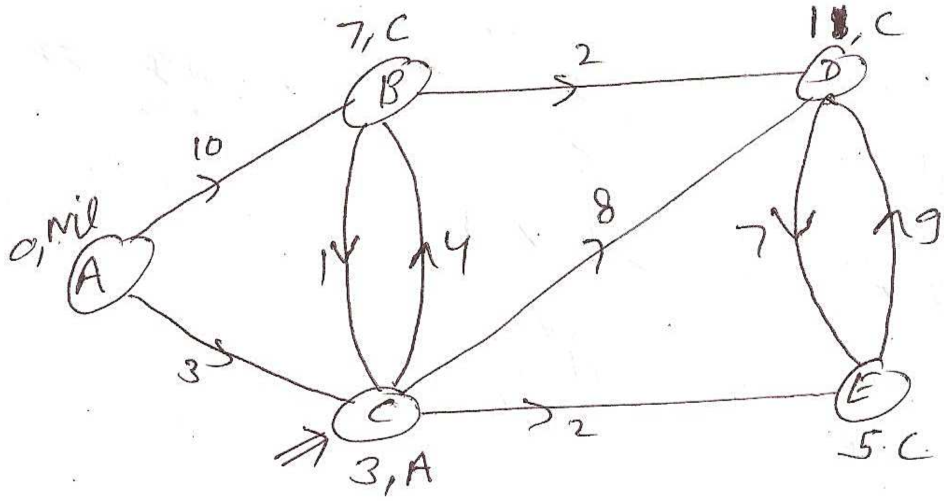
Extract A From Q & Relax all edges leaving A:



Q: ~~A~~, B C D E
0 10 3 ∞ ∞

S: {A}

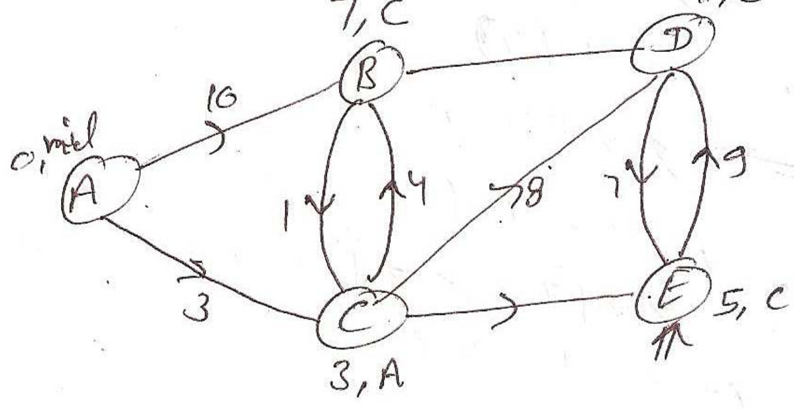
Now Extract C From Q & Relax All edges leaving C



Q: ~~A~~ B ~~C~~ D E
0 7 3 11 5

S: {A, C}

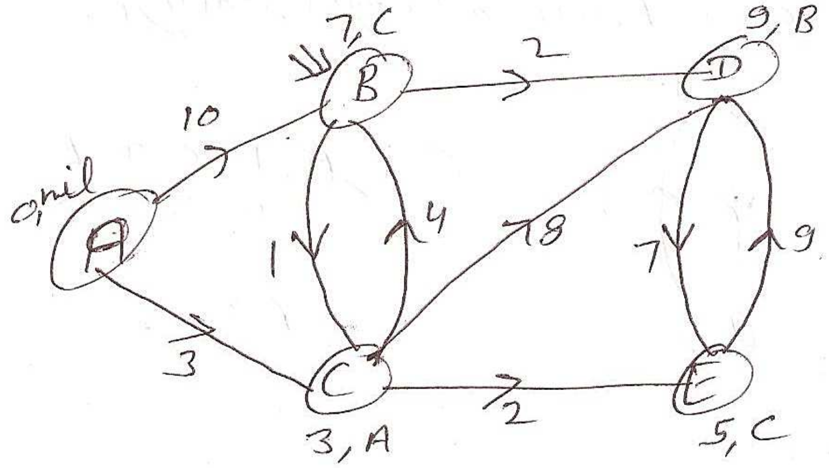
Now Extract E From Q & Relax all edges leaving E:



Q: ~~A~~ B ~~C~~ D ~~E~~
0 7 3 11 5

S: {A, C, B}

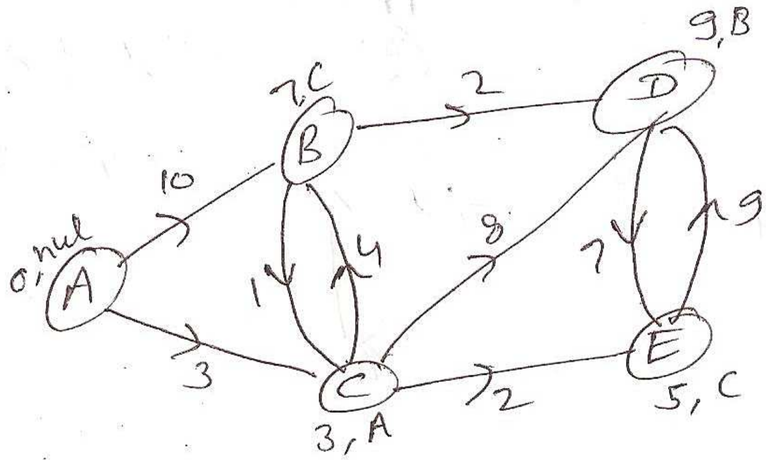
Extract B & Relax all edges leaving B:



Q : ~~A B C D E~~
0 7 3 9 5

S : {A, B, C, E}

Extract D & Relax all edges leaving D:



Q : ~~A B C D E~~
0 7 3 9 5

S : {A B C D E}

Result:

$A \to A : (0)$

$A \to B : \{\pi[B] = C$ & $\pi[C] = A\}$
$\Rightarrow$ Path $A \to C \to B$ (7)

$A \to E : \{\pi[E] = C$ & $\pi[C] = A\}$
$\Rightarrow$ Path $= A \to C \to E$ (5)

$A \to D : \{\pi[D] = B$ & $\pi[B] = C$ & $\pi[C] = A\}$
$\Rightarrow$ Path $= A \to C \to B \to D$ (9)

$A \to C : \{\pi[C] = A\}$
$\Rightarrow$ Path $= A \to C$ (3)

BELLMAN FORD ALGORITHM:

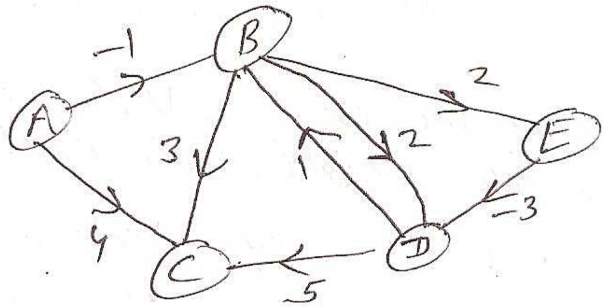
This algo finds all shortest path lengths from a source $s \in V$ to all $v \in V$ or determines that a negative weight cycle exists. Thus the Bellman Ford algo solves the single source shortest paths problem in the more general case in which edge weight can be negative.

Algorithm return boolean value:

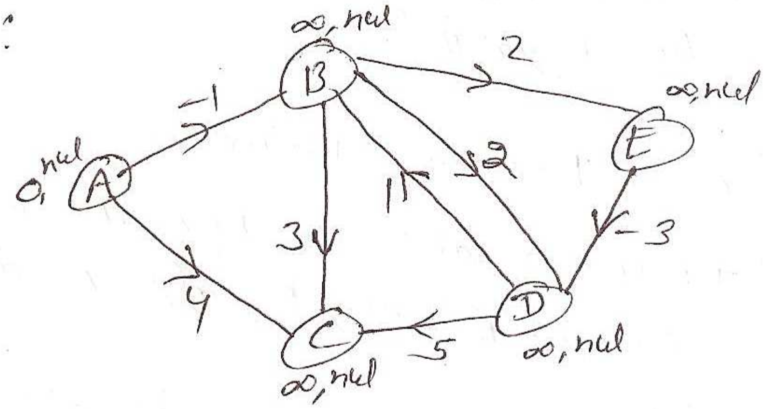
* if return "TRUE" means no negative weight cycle (or -ve edge not create problem) so algo produce shortest path.
* if return "FALSE" means there is -ve weight cycle so Algo indicate no solution.

```
BELLMAN_FORD (G, w, s)
{ INITIALIZE_SINGLE_SOURCE (G, s)
  for (i=1 to |V[G]| - 1)
    { RELAX (u, v, w)
    }
  for (each edge (u,v) ∈ E[G])
  { if (d[v] > d[u] + w(u,v))
     {
      return (false)
     }
  }
  return (TRUE)
}
```

ex.



initialize:



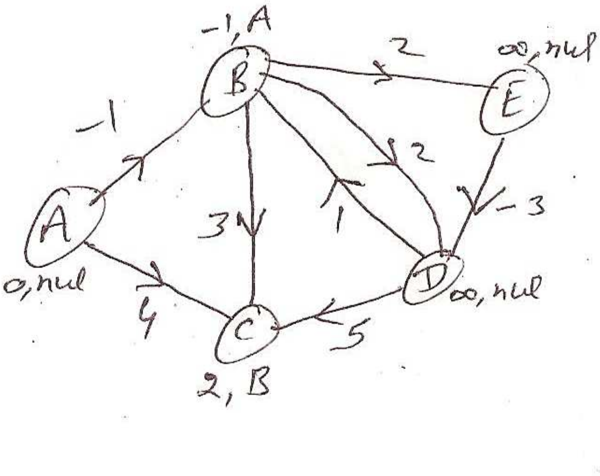
A	B	C	D	E
0	∞	∞	∞	∞

Let order of edge: (B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D)
there 5 vertex so first loop run i=1 to 4 means.
these edges are RELAX four time in this sequence.

for i = 1.

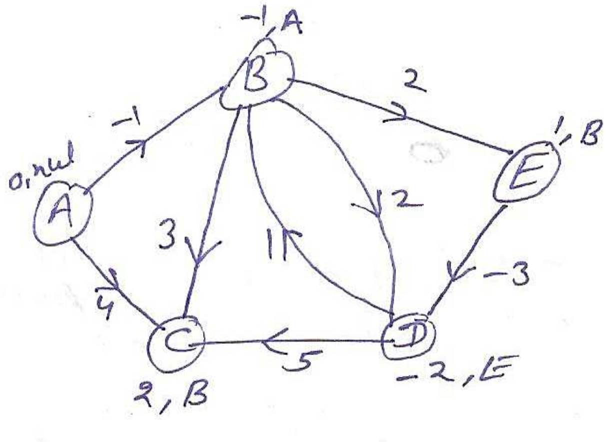
RELAX on	A 0, nul	B ∞, nul	C ∞, nul	D ∞, nul	E ∞, nul
BE	0, nul	∞, nul	∞, nul	∞, nul	∞, nul
DB	″	″	″	″	″
BD	″	″	″	″	″
AB	″	-1, A	″	″	″
AC	″	″	4, A	″	″
DC	″	″	″	″	″
BC	″	″	2, B	″	″
ED	″	″	″	″	″

Result after loop i=1



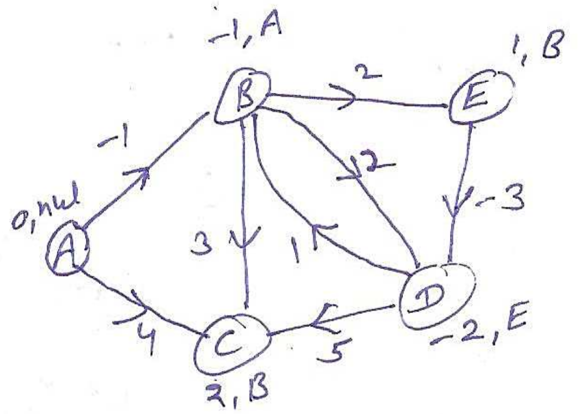
for $i=2$

RELAX on	A	B	C	D	E
	0,nul	-1,A	2 B	∞ nul	∞ nul
BE	"	"	"	"	1,B
DB	"	"	"	"	"
BD	"	"	"	1,B	"
AB	"	"	"	"	"
AC	"	"	"	"	"
DC	"	"	"	"	"
BC	"	"	"	"	"
ED	0,nul	-1,A	2 B	-2 E	1,B



For $i=3$

RELAX on	A	B	C	D	E
	0,nul	-1,A	2,B	-2,E	1,B
BE	"	"	"	"	"
DB	"	"	"	"	"
BD	"	"	"	"	"
AB	"	"	"	"	"
AC	"	"	"	"	"
DC	"	"	"	"	"
BC	"	"	"	"	"
ED	0,nul	-1,A	2,B	-2,E	1,B



No changes found in table for $i=3$ so same Result will find for $i=4$

Now find boolean value (by 2nd for loop)

for BE : if $(d(E) > d(B) + w(B,E))$ then Return False

$(1 > (-1+2))$ condition not true

DB : $(-1 > (0+(-1)))$ condition not true

BD : $(-2 > (-1+2))$ condition not true

AB : $(-1 > (0+(-1)))$ "

AC : $(2 > (0+4))$ "

DC : $(2 > (-2+5))$ "

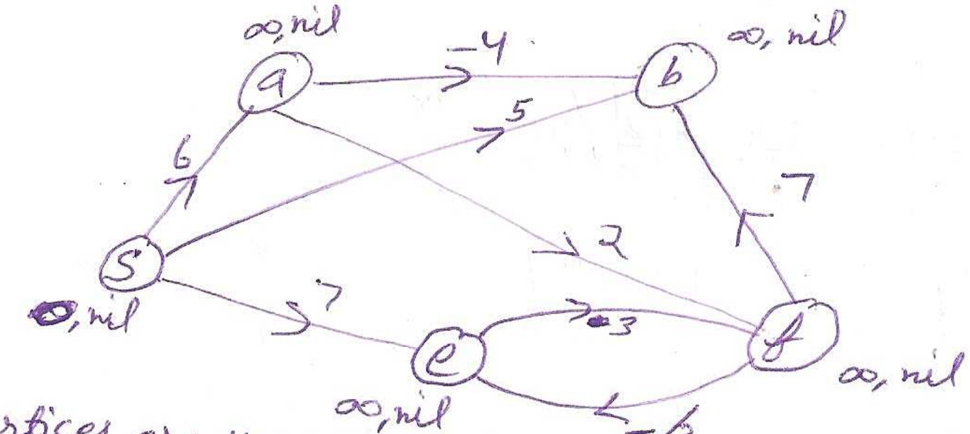
BC : $(2 > (-1+3))$ "

ED : $-2 > (1+(-3))$ "

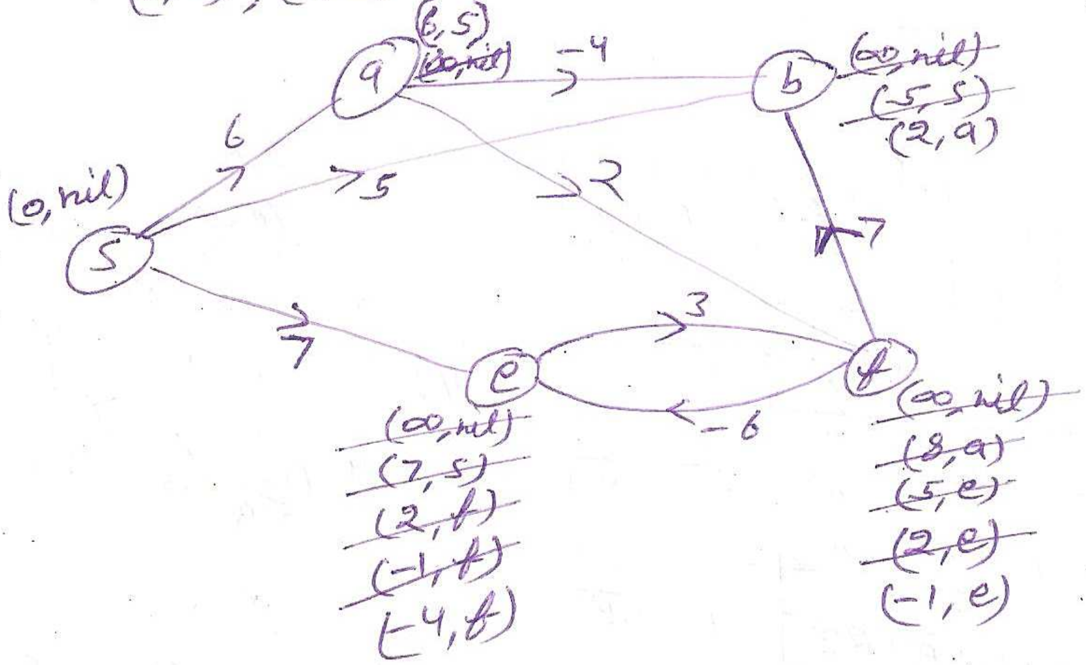
no one return false so finaly return True & solution is correct

shortest value & Path found Process same as Dijkstra.

Q.



5 vertices are there so first for loop run four times.
Let order of edge is (a,b), (a,f), (f,b), (f,e), (e,f), (S,a), (S,b), (S,e)



Now find boolean value (by 2^{nd} for loop)

for (f, e) : $d(e) > d(f) + w(f,e)$
$(-4 > (-1 + (-6)))$ ⟸ condition true

So Return false boolean value.

* means There is a -ve weight cycle so no solution

Dynamic Programming

Dynamic is an algorithm design method that can be used when the solutions to a Problem can be viewed as the result of a sequence of can be viewed as the result of a sequence of decisions.

* in this we find all Possible feassible solutions and out of these solution select an optimal solution.
* Generally the complexity of these type of algo is exponential i.e. (2^n) or (n^n)
* This technique is like Divide & conquer and solves Problems by dividing them into subproblems.
* Dynamic Programming is used when the subProblems are not independent. means one subproblem's result also depends on other subproblems
* Dynamic Programming is an approach developed to solve sequential or multistage, decision Problems. hence the name "Dynamic" Programming.

Application:

(i) Knapsack Problem
(ii) shortest Path Problem
(iii) matrix chain multiplication
(iv) Longest common sequence
(v) Resource allocation Problem

All Pair shortest Paths :

its aims to compute the shortest Path from each vertex v to every other u.

* we can expect to get a naive implementation of $O(n^3)$ if we use Dijkstra for example, i.e. running a $O(n^2)$ Process n times. & if use Bellman-Ford algorithm, it will take about $O(n^4)$
* but Both are memory expensive, as we need one spanning tree for each vertex. so these are impractical in terms of memory consumption.

* so output require in tabular form.
  Three approaches are there
  (i) matrix multiplication $O(V^3 \log V)$
  (ii) Floyd - Warshall $O(V^3)$
  (iii) Johnson $O(V^2 \log V + VE)$

Floyd - warshall Algorithm :

* Find shortest Path between all pair of vertices
* -ve weight may be present but no negative weight cycle.
* complexity $O(V^3)$

Algorithm: matrix of edge weight

FLOYD-WARSHALL (W)

```
{   n ← rows[W]
    D^0 ← W
    for (K ← 1 to n)
      { for (i ← 1 to n)
          { for (j ← 1 to n)
              { d_ij^(K) ← min(d_ij^(K-1), d_iK^(K-1) + d_Kj^(K-1))
              }
          }
      }
    Return D^n
}
```

Constructing a shortest Path:

for This we construct $\pi^{(0)}, \pi^{(1)}, \pi^{(2)} \ldots \pi^{(n)}$ & $\pi^{(n)}$ show the predecessor of vertex j on a shortest path from vertex i with all intermediate vertices in the set.

$$\pi_{ij}^{(0)} = \begin{cases} NIL & \text{if } (i = j) \text{ or } w_{ij} = \infty \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty \end{cases}$$

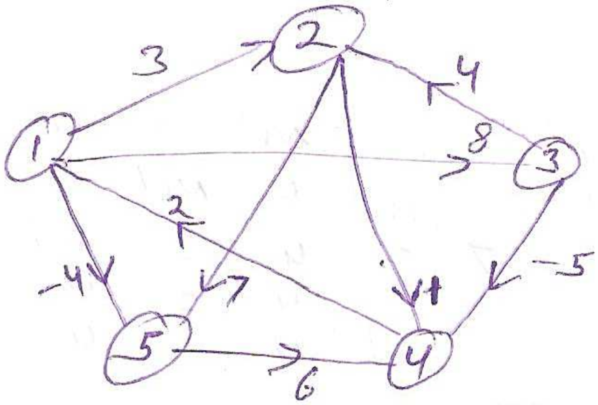
$$\pi_{ij}^{(K)} = \begin{cases} \pi_{ij}^{(K-1)} & \text{if } d_{ij}^{(K-1)} \leq d_{iK}^{(K-1)} + d_{Kj}^{(K-1)} \\ \pi_{Kj}^{(K-1)} & \text{if } d_{ij}^{(K-1)} > d_{iK}^{(K-1)} + d_{Kj}^{(K-1)} \end{cases}$$

or

$$\pi_{ij}^{(K)} = \begin{cases} \pi_{Kj}^{(K-1)} & \text{if } d_{ij}^{(K-1)} \Rightarrow d_{ij}^{(K)} \\ \pi_{ij}^{(K-1)} & \text{if } d_{ij}^{(K-1)} = d_{ij}^{(K)} \end{cases}$$

no change

Q.



$$D^{0} = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}, \quad \pi^{[0]} = \begin{bmatrix} nil & 1 & 1 & nil & 1 \\ nil & nil & nil & 2 & 2 \\ nil & 3 & nil & nil & nil \\ 4 & nil & 4 & nil & nil \\ nil & nil & nil & 5 & nil \end{bmatrix}$$

$$D^{1} = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \quad \pi^{(1)} = \begin{bmatrix} nil & 1 & 1 & nil & 1 \\ nil & nil & nil & 2 & 2 \\ nil & 3 & nil & nil & nil \\ 4 & 1 & 4 & nil & 1 \\ nil & nil & nil & 5 & nil \end{bmatrix}$$

$$D^{2} = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \quad \pi^{(2)} = \begin{bmatrix} nil & 1 & 1 & 2 & 1 \\ nil & nil & nil & 2 & 2 \\ nil & 3 & nil & 2 & 2 \\ 4 & 1 & 4 & nil & 1 \\ nil & nil & nil & 5 & nil \end{bmatrix}$$

$$D^{3} = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix} \quad \pi^{(3)} = \begin{bmatrix} nil & 1 & 1 & 2 & 1 \\ nil & nil & nil & 2 & 2 \\ nil & 3 & nil & 2 & 2 \\ 4 & 1 & 4 & nil & 1 \\ nil & nil & nil & 5 & nil \end{bmatrix}$$

$$D^{4} = \begin{bmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix} \quad \pi^{4} = \begin{bmatrix} nil & 1 & 4 & 2 & 1 \\ 4 & nil & 4 & 2 & 1 \\ 4 & 3 & nil & 2 & 1 \\ 4 & 3 & 4 & nil & 1 \\ 4 & 3 & 4 & 5 & nil \end{bmatrix}$$

$$D^{(5)} = \begin{bmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix} \quad \pi^{(5)} = \begin{bmatrix} \text{Nil} & 3 & 4 & 5 & 1 \\ 4 & \text{Nil} & 4 & 2 & 1 \\ 4 & 3 & \text{Nil} & 2 & 1 \\ 4 & 3 & 4 & \text{Nil} & 1 \\ 4 & 3 & 4 & 5 & \text{Nil} \end{bmatrix}$$

* $D^{(5)}$ show the shortest weight matrix

like $d^{(5)}_{(2,5)} = -1$ means shortest path length from 2 to 5 is -1

Now find Path:

$\pi^{(5)}_{(2,5)} = 1$

$\Rightarrow \pi^{(5)}_{(2,1)} = 4$

$\Rightarrow \pi^{(5)}_{(2,4)} = 2$

$\Rightarrow \pi^{(5)}_{(2,2)} = \text{nil}$

path

so path is shortest $2 \to 4 \to 1 \to 5$