Notes
on
Design and Analysis
of
Algorithm

By

Dr. Satyasundara Mahapatra

(Module –III)

Divide and Conquer with Examples Such as Sorting, Matrix Multiplication, Convex Hull and Searching. Greedy Methods with Examples Such as Optimal Reliability Allocation, Knapsack, Minimum Spanning Trees – Prim's and Kruskal's Algorithms, Single Source Shortest Paths - Dijkstra's and Bellman Ford Algorithms.

Divide & conquer: Matrix Multiplication

$$T(n) = 8T(n/2) + n^2$$

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$C = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

$$c_{11} = a_{11} b_{11} + a_{12} b_{21}$$

$$c_{12} = a_{11} b_{12} + a_{12} b_{22}$$

$$c_{21} = a_{21} b_{11} + a_{22} b_{21}$$

$$c_{22} = a_{21} b_{12} + A_{22} B_{22}$$

Suppose, we wish to compute $C = A \times B$, where A, B & C are $n \times n$ matrices, assuming that n exact ponter of 2. we divide each of A, B & C, into 4 $(n/2 \times n/2)$ matrices. Each of 4 equation specify two multiplication of $n/2 \times n/2$ matrices & their addition of $n/2 \times n/2$ Productions.

The divide and conquer strategy represented by the following equation (recursive).

$$T(n) = 8T(n/2) + n^2$$

$$\Rightarrow T(n) = \Theta(n^3).$$

Strossen discovered a recursive approach, that require 7 recursive multiplications of $n/2 \times n/2$ and n^2 scalar addition/ substraction. Then

The recursive relation is

$$T(n) = 7T(n/2) + n^2$$

$$= O(n^{2.81}).$$

Strassen method has 4 steps:-

1) Divide the input matrices A & B into $n/2 \times n/2$ matrices.

2) Using $\Theta(n^2)$ scalar additions & subtraction compute 14 $(n/2 \times n/2)$ matrices. i.e $a_1 b_1, a_2 b_2, a_3 b_3, \ldots, a_7, b_7$

3) Recursively compute the seven matrix product, $P_i = a_i b_i$ where $i = 1, 2, \ldots, 7$.

4) Compute the result matrix of C.

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$P = (a_{11} + a_{22})(b_{11} + b_{22})$$

$$Q = (a_{21} + a_{22}) b_{11}$$

$$R = a_{11}(b_{12} - b_{22})$$

$$S = a_{22}(b_{21} - b_{11})$$

$$T = (a_{11} + a_{12}) b_{22}$$

$$U = (a_{21} - a_{11})(b_{11} + b_{12})$$

$$V = (a_{12} - a_{22})(b_{21} + b_{22})$$

$c_{11} = P + S - T + V$

$c_{12} = R + T$

$c_{21} = Q + S$

$c_{22} = P + R - Q + U$

Ques: The strassen algo to compute the product of two sq. matrix.

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad b = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

$a_{11} = 1 \quad a_{12} = 2 \quad a_{21} = 3 \quad a_{22} = 4$

$b_{11} = 5 \quad b_{12} = 6 \quad b_{21} = 7 \quad b_{22} = 8$

$P = (a_{11} + a_{22})(b_{11} + b_{22}) = (1+4)(5+8) = 65$

$Q = (a_{21} + a_{22}) b_{11} = (3+4)5 = 35$

$R = a_{11}(b_{12} - b_{22}) = 1(6-8) = -2$

$S = a_{22}(b_{21} - b_{11}) = 4(7-5) = 8$

$T = (a_{11} + a_{12}) b_{22} = (1+2)8 = 24$

$U = (a_{21} - a_{11})(b_{11} + b_{12}) = (3-1)(5+6) = 22$

$V = (a_{12} - a_{22})(b_{21} + b_{22}) = (2-4)(7+8) = -30.$

$c_{11} = P + S - T + V = 65 + 8 - 24 - 30 = 19$

$c_{12} = R + T = -2 + 24 = 22$

$c_{21} = Q + S = 35 + 8 = 43$

$c_{22} = P + R - Q + U = 65 - 2 - 35 + 22 = 50.$

Ques: Multiply the given matrices by Strossen Algo.

$$A = \begin{bmatrix} 5 & 6 & 7 & 8 \\ 8 & 9 & 2 & 1 \\ 2 & 4 & 6 & 8 \\ 4 & 8 & 7 & 8 \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

Ans.

$$A = \left[\begin{array}{cc|cc} 5 & 6 & 7 & 8 \\ 8 & 9 & 2 & 1 \\ \hline 2 & 4 & 6 & 8 \\ 4 & 8 & 7 & 8 \end{array}\right] \qquad B = \left[\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{array}\right]$$

A_1 A_2 A_3 A_4 — B_1 B_2 B_3 B_4

$$C = \begin{bmatrix} C_1 & C_2 \\ C_3 & C_4 \end{bmatrix}$$

Apply strossen Algo 4 times to complete the multiplication.

Ques: Why strossen Algo is better than normal Matrix multiplication algo?

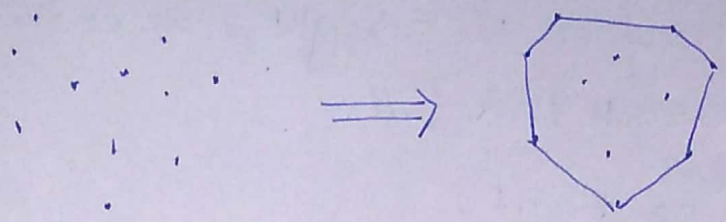
Ans: Strossen Algo decrease exactly one multiplication as compare to Matrix Multiplication and increases addition & Substraction. So complexity of Strossen is less.

$$T(n) = 7(n/2) + n^2 = \theta(n^{2.81})$$

$$T(n) = 8(n/2) + n^2 = \theta(n^3).$$

Convex Hull {Divide and Conquer}

A convex hull is the smallest convex polygon containing all the given points. as given below.



or Given set of n points, find a convex polygon using these points. for which points lies inside it.

use Divide & Conquer algorithm.

Quick Hull (S)

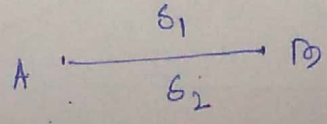
// Find convex hull from the set S of n points.

Convex Hull = { } // Empty set.

1. Find left and right most points, say A & B and add A & B to Convex Hull.

2. Segment AB divides the remaining (n-2) points into two groups S_1 and S_2 :

Where S_1 are points in S that are on the right side of the oriented line form A to B and S_2 are points in the S that are on the right side of the oriented line from B to A.



3. Find Hull (S1, A, B)
4. Find Hull (S2, B, A).

Find Hull (S_K, P, Q).

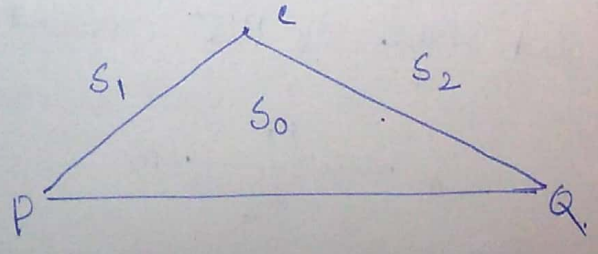
// { Find points on convex hull from the set S_K Points that are on the right side of the oriented from P to Q } //.

If S_K has no point.

Then return.

- From the given set of points in S_K, find farthest point, say C, from segment PQ.
- Add point C to convex hull at the location between P and Q. Three points P, Q, and C partition the remaining points of S_K into 3 subsets: S_0, S_1, and S_2.
- where S_0 are points inside triangle PCQ, S_1 are points on the right side of the oriented line from P to C, and S_2 are the points on the right side of the oriented line from C to Q.

- Find Hull (S_1, P, C)
- Find Hull (S_2, C, Q).



Time complexity of Quick Hull.

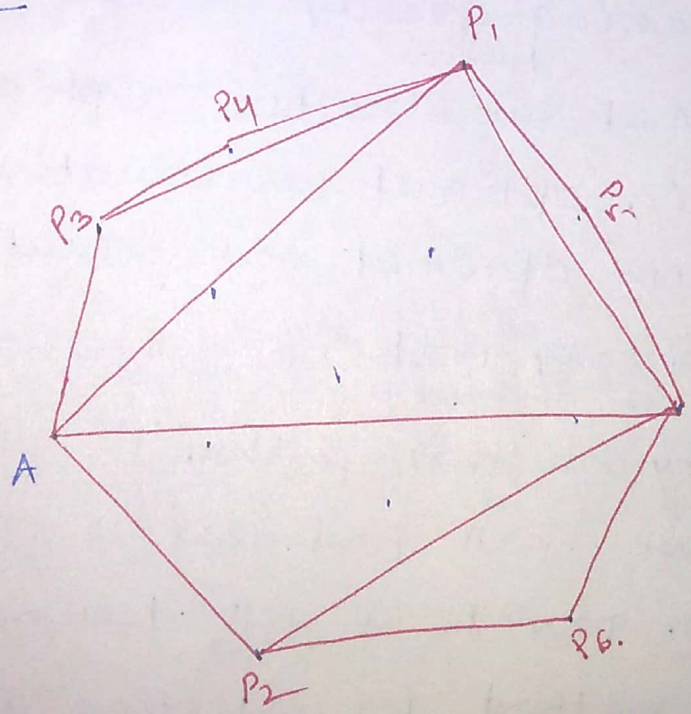
$$T(n) = \underbrace{T(l)}_{\text{Point in left side of AB.}} + \underbrace{T(n-l)}_{\text{Points in the right side of AB.}} + \underbrace{O(n)}_{\text{To find the farthest Point}}$$

$$= 2T\left(\frac{n}{2}\right) + O(n).$$

$T(n) = O(n \lg n)$ in average case.

Since it is a problem like Quick Sort. the worst case complexity is $T(n) = O(n^2)$.

Example.



The convex hull A P3 P4 P1 P5 B P6 P2

Greedy Algorithm.

Greedy Algorithm solve problem by making the choice that seems best at the particular moment. Many optimization problem can be solve by using greedy Algorithm. Some problem have no efficient solution but a greedy algorithm may provide a solution i.e closed to optimal.

A greedy algo work if the problem have following two properties:-

1 → Greedy choice property: A globally optimal solution can be arrived by making a locally optimal soln. In other words an optimal solution can be obtain by making greedy choice.

2 → Optimal Sub structure: Optimal soln. contain optimal sub solution & it contains optimal sub solution & so on

(Unders Greedy Problem)

Activity slection problem:

We shall find that a greedy algorithm provide a well designed & simple method for selecting a maximum size set of mutually compatible activities. It can be observed that the process of

selecting the activities become faster, if we assume that the I/P activities are in order by increasing finishing time. i.e.

$$f_1 \leq f_2 \leq f_3 \leq \cdots \leq f_{10}.$$

e.g. Given 10 activities with their start & finish compute a schedule where the largest no. of activities takes place.

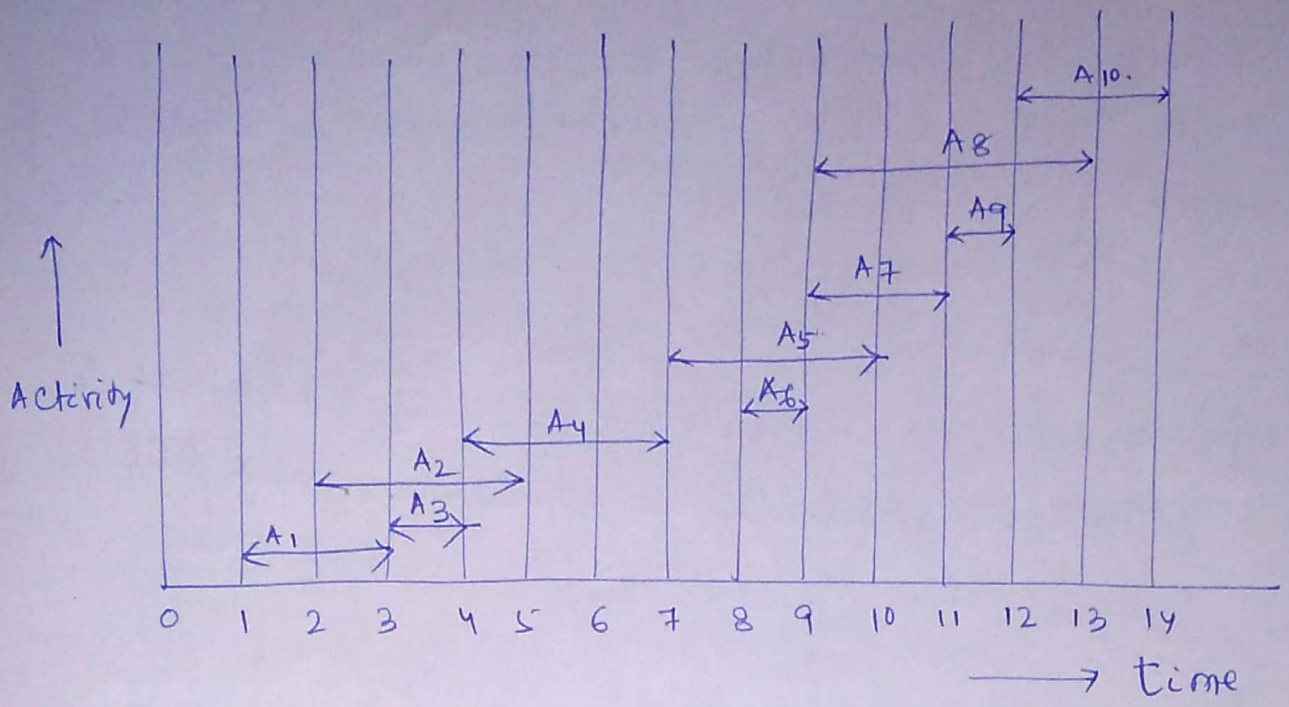
Activities.	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}.
s_i	1	2	3	4	7	8	9	9	11	12
f_i	3	5	4	7	10	9	11	13	12	14.

Solution

First arrange the following activities in increasing order of their finishing time.

i.e.

Activities	A_1	A_3	A_2	A_4	A_6	A_5	A_7	A_9	A_8	A_{10}.
s_i	1	3	2	4	8	7	9	11	9	12
f_i	3	4	5	7	9	10	11	12	13	14



One solution is

$\langle A_1, A_3, A_4, A_5, A_7, A_9, A_{10} \rangle$

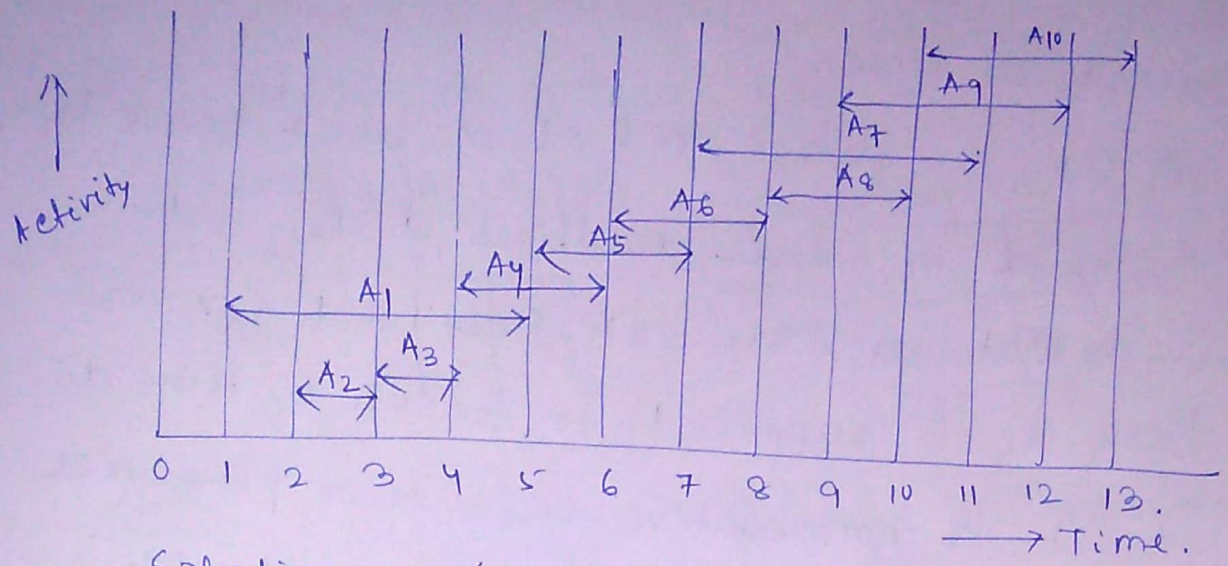
i.e Maximum 7 activities.

Question. find the optimal set in the given activity selection problem.

Activities	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	5	6	7	8	9	10.
f_i	5	3	4	6	7	8	11	10	12	13.

Now arrange the above activities in increasing order of their finishing time.

Activities	A_2	A_3	A_1	A_4	A_5	A_6	A_8	A_7	A_9	A_{10}.
s_i	2	3	1	4	5	6	8	7	9	10.
f_i	3	4	5	6	7	8	10	11	12	13.



Solution – $\langle A_2, A_3, A_4, A_6, A_8, A_{10} \rangle$

i.e. 6 activities.

GreadyActivity Selector (s, f)

1. n = length [s]
2. A = {1} // select first activity
3. j = 1
4. for i = 2 to n
5. if ($s_i \geq f_i$)
6. then A = A ∪ {ai}
7. j = i
8. return A.

* Task scheduling problem:

This is the problem of optimally scheduling unit time task on a single processor where each task has a deadline and penalty that must be paid if deadline is missed.

A unit time task is a job such a program to be run on a computer that require exactly

one unit of time.

→ A set $S = \{a_1, a_2, \ldots a_n\}$ of n unit time task

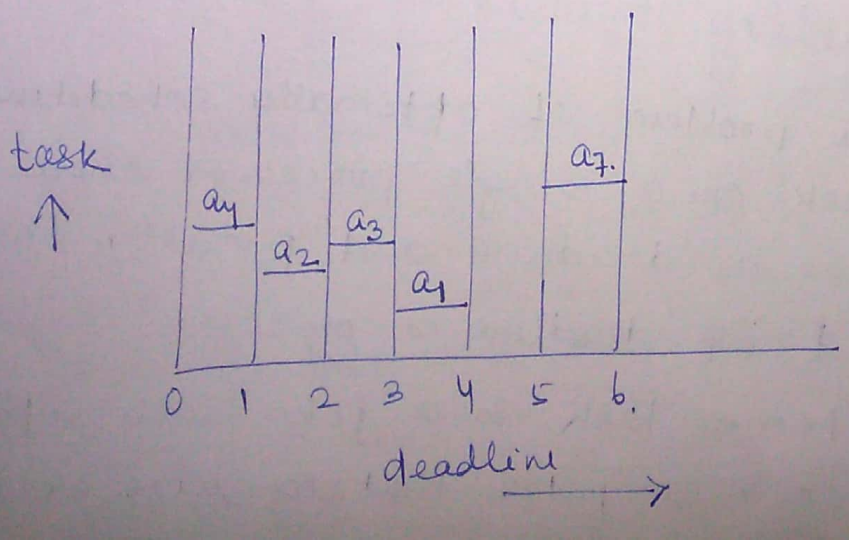
→ A set of n integer deadlines $d_1, d_2, \ldots, d_n$, such that each d_i satisfies $1 \leq d_i \leq n$ and task a_i is supposed to finish by time d_i

→ A set of nonnegative weights or penalties $w_1, w_2, \ldots, w_n$, such that a penalty w_i is occured if task a_i is not finished by time i.e its deadline.

Ques: find the optimal schedule for the following task with given weight (penalties) & deadlines

Task	a_1	a_2	a_3	a_4	a_5	a_6	a_7
d_i	4	2	4	3	1	4	6
p_i / w_i	70	60	50	40	30	20	10

According to greedy algorithm, first sort the job in decreasing order of their penalties, so that minimum penalty will be charged.



a_4	a_2	a_3	a_1		a_7
1	2	3	4	5	6

So loss = $a_5 + a_6$.

= 30 + 20

= 50.

Ques: Let the no. of task = 4

task	a_1	a_2	a_3	a_4
di	2	1	2	1
Pi/wi	100	10	15	27

→ after sort

tasks	a_1	a_4	a_3	a_2
di	2	1	2	1
Pi/wi	100	27	15	10.

Sol.

a_4	a_1
1	2

Because the maximum deadline is 2.

So loss = 15 + 10. = 25.
(a_3 a_2)

Ques: Let the no. of task = 7.

task	a_1	a_2	a_3	a_4	a_5	a_6	a_7
di	1	3	3	4	1	2	1
Pi/wi	3	5	18	20	6	1	38

// sort the task on the basis of penalties.

tasks	a_1	a_4	a_3	a_5	a_2	a_1	a_6.
di	1	4	3	1	3	1	2
Pi/wi	38	20	18	6	5	3	1

Maximum deadline limit = 4

a_7	a_2	a_3	a_4
1	2	3	4

So loss = a_5 + a_1 + a_6 = 6 + 3 + 1 = 10.

Knapsack Problem:

The knapsack Problem is an optimization problem used to illustrate both problem and solution.

The problem is:

"Given a set of items with specific weights and values, the aim is to get as much values into the knapsack as possible given the weight constraint of the knapsack."

This problem is an example of a "combinatorial optimization problem" - a topic in mathematics and computer science about finding the optimal object among a set of objects.

There are two versions of the problem:

1. 0-1 Knapsack Problem. (Dynamic Approach)
2. Fractional Knapsack Problem. (Greedy Approach)

Fractional Knapsack Problem:

"Given n objects and a knapsack with a capacity M. (weight)

→ Each object i has weight w_i and Profit p_i,

→ For each object i, suppose a fraction x_i, $0 \leq x_i \leq 1$, can be placed in the

Knapsack, then the profit earned is $p_i x_i$

The objective is to maximize profit subject to capacity constraints.

i.e Maximize

p = profit
w = weight.

$$\sum_{i=1}^{n} p_i x_i$$

Subject to

$$\sum_{i=1}^{n} w_i x_i \leq M$$

where $0 \leq x_i \leq 1$,

$p_i > 0$;

$w_i > 0$;

A feasible solution is any sub-set $\{x_1, \ldots, x_n\}$ satisfying (2) and (3)

An optimal solution is a feasible solution that maximize $\sum_{i=1}^{n} p_i x_i$.

This problems appear in real world decision making processes in a wide variety of fields, such as finding the least wasteful way to cut raw materials, selection of investments and portfolios, resource allocation etc.

Example:

Find an optimal solutions for a given instance of knapsack problem. where the n (objects) = 3, M (weight of knapsack) = 20.

$\langle P_1, P_2, P_3 \rangle = \langle 25, 24, 15 \rangle$

$(w_1, w_2, w_3) = \langle 18, 15, 10 \rangle$

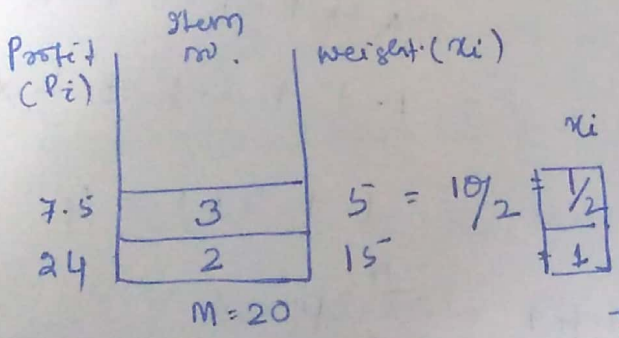
Find the ratio between Profit & weight.

$\langle P_1/w_1, P_2/w_2, P_3/w_3 \rangle = \langle 25/18, 24/15, 15/10 \rangle$

$= \langle 1.38, 1.6, 1.5 \rangle$

Sort the item by descending order of (P_i/w_i)

Item/Object.	2	3	1
P_i/w_i	1.6	1.5	1.38



Total Profit = $P_2 x_2 + P_3 x_3$

$= 24 \cdot 1 + 15 \cdot \frac{1}{2}$

$= 31.5$

Total weight = $w_2 x_2 + w_3 x_3$.

$= 15 \cdot 1 + 10 \cdot \frac{1}{2}$

$= 15 + 5 = 20$

Q. Find an optimal solution for a given instance of Fractional knapsack problem with 4 nos. of objects and the size of knapsack = 4.

Object (N)	A	B	C	D
Profit (P)	5	9	4	8
weight (w)	1	3	2	2

First find the ratio between Profit and weight.

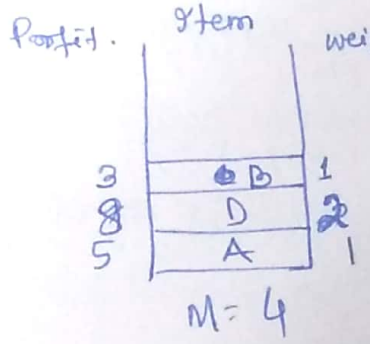
H)

i.e P_i/w_i =

Objects.	A	B	C	D
P_i/w_i	$5/1$	$9/3$	$4/2$	$8/2$
	$=5$	$=3$	$=2$	$=4$

Then sort the objects in decending order for P_i/w_i

Objects.	A	D	B	C
P_i/w_i	5	4	3	2



for 3 = 9

1 = 9/3 = 3

Hence the Total weight = $\sum_{i=1}^{n} w_i x_i$

$= w_A x_A + w_D x_D + w_B x_B.$

$= 1 \cdot 1 + 2 \cdot 1 + \frac{1}{3} \times 3$

$= 1 + 2 + 1 = 4$

Total Profit = $\sum_{i=1}^{n} P_i x_i$

$= P_A x_A + P_D x_D + P_B x_B.$

$= 5 \times 1 + 8 \times 1 + 9\frac{1}{3} \times 9.$

$= 5 + 8 + 3 = 16.$

Graham's Scan methods for solving Convex-hull (CH) Problem:

Graham's Scan solves the convex-hull problem by maintaining a stack S of candidate points. It pushes each point of the input set Q onto the stack one time. And it eventually pops from the stack each point that is not a vertex of CH(Q). The algorithm terminates when, stack S contains exactly the vertices of CH(Q) in counter clockwise order of their appearance on the boundary.

The procedure Graham-Scan takes an input a set Q of points, where $|Q| \geq 3$.

Graham-Scan (Q)

1. Let P_0 be the point in Q with minimum y co-ordinate or else leftmost such point in case of a tie.
2. let $\langle P_1, P_2, P_3, \ldots P_m \rangle$ be the remaining point in Q, sorted by polar angle in counterclockwise order around P_0.
3. Let S be an empty stack
4. PUSH (P_0, S)
5. PUSH (P_1, S)
6. PUSH (P_2, S)

7. for $i = 3$ to m

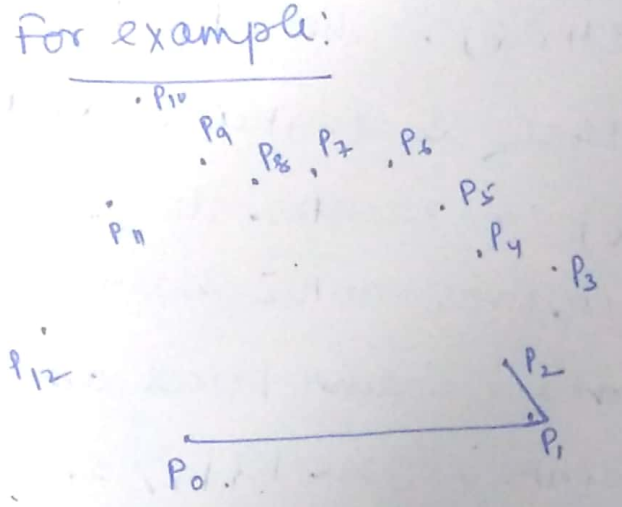
8. while the angle formed by points Next-To-Top(S), Top(S) and P_i makes a non left turn

9. pop(S)

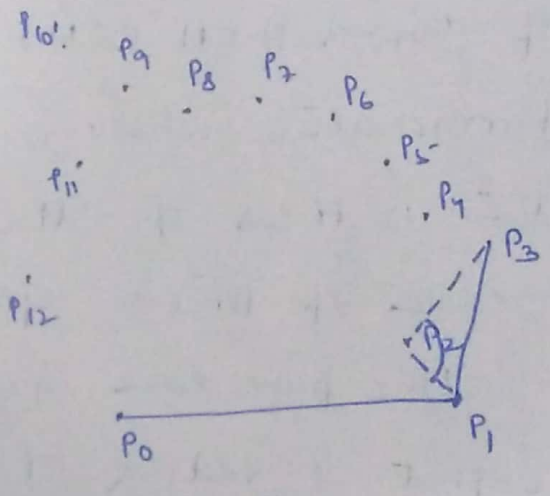
10. Push(P_i, S)

11. return S.

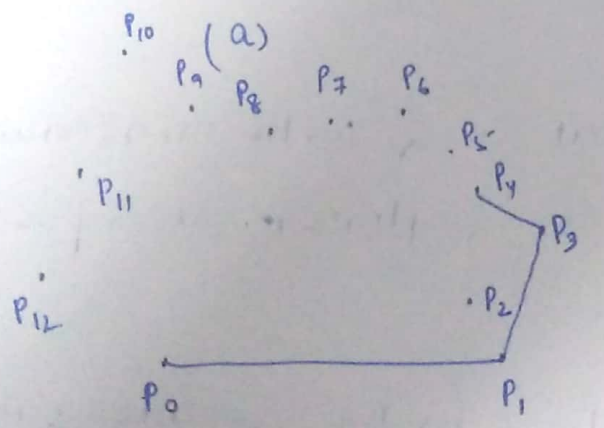
For example:



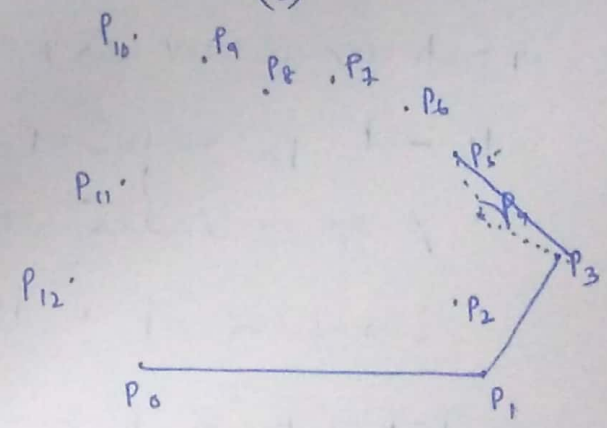
(a)



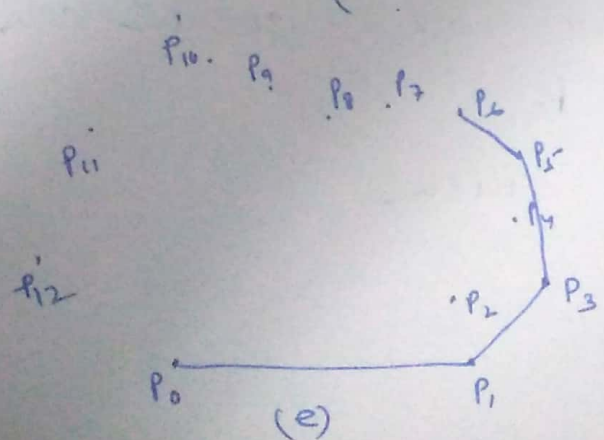
(b)



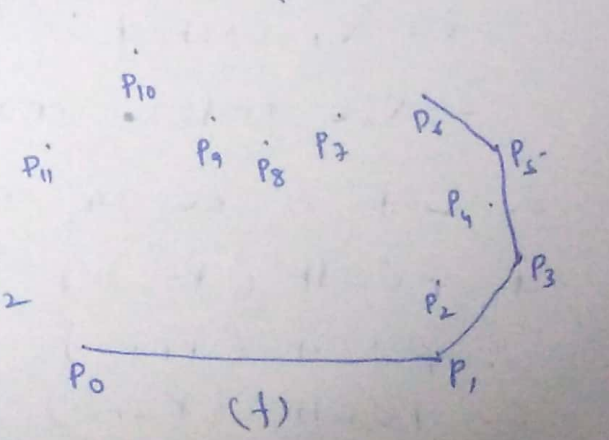
(c)



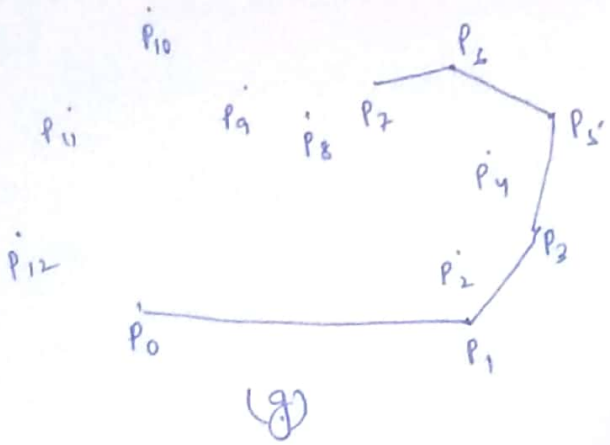
(d)



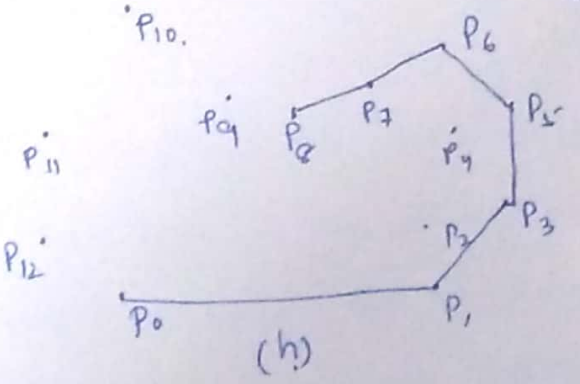
(e)



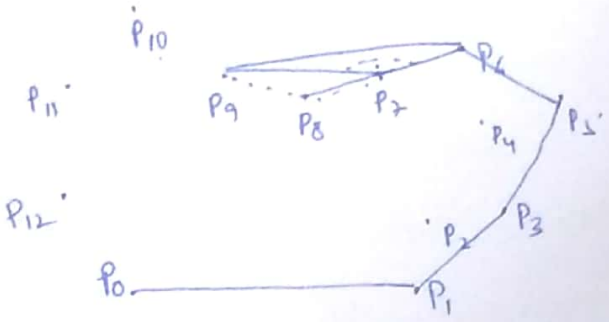
(f)



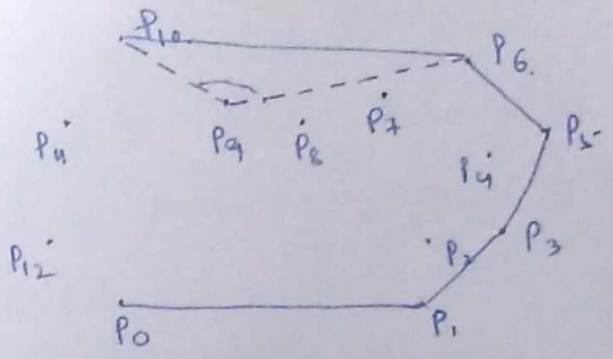
(g)



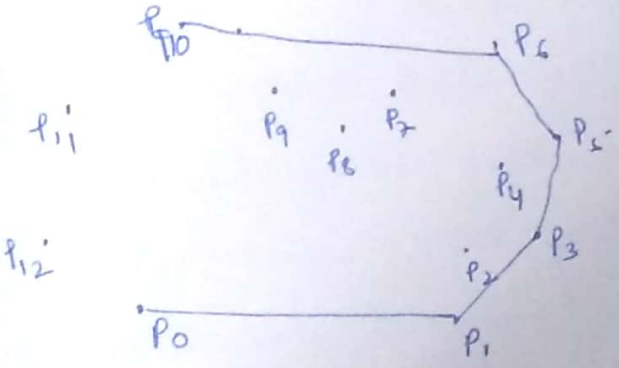
(h)



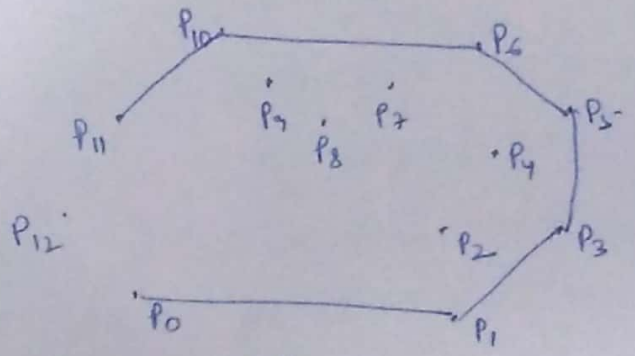
(i)



(j)



(k)



(l)

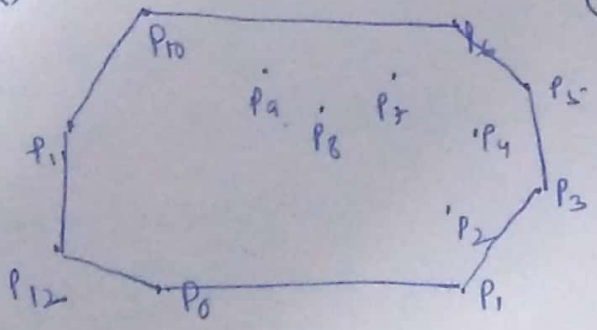


fig: The convex Hull returned by the Graham Scan Procedure.

The running time of Graham-Scan is $O(n \lg n)$, where $n = |Q|$. Line 1 takes — $Q(n)$

Line 2 takes — $O(n \lg n)$

Line 3-6 take — $O(1)$

Line 7-10 take — $O(n)$

Huffman Coding

Huffman coding is a compression coding Technique It is used for reducing the size of the data or message. Suppose we want to store the data in a file. so we want to store the data in compressed form to reduce the size of the file. when a data is sent over a network data can be compressed and send in a network to reduce the cost of transmission.

In this session we will discuss, if a normal message is sent over a network then what is the cost and how we reduce it by compressing the data by using fixed length and variable length encoding other wise known as Huffman coding. Now Let us solve a problem with the help of following message.

Message: BCCABBDDAECCBBAEDDCC

The above message we want to store in a machine or we want to send it over a network. The cost of this message is calculated in terms of bits.

The length of this message is 20. In general this message is sent by help of ASCII code which has 8 bits. In the above message there are 20 alphabets are available. If each alphabet is required 8 bits then 20 alphabets

required 20 x 8 bit = 160 bits. Hence the size of the message is 120 bits. This is the simple message sending with out encoding.

It was observed that the above string or message formed with the help of five english @ character i.e A to E. Let first apply the fixed size 3 bit for message transformation From the above message it was also observed that a single character is available more than one time. Let us record it in a table.

character	frequency	code
A	3	000
B	5	001
C	6	010
D	4	011
E	2	100.
	20.	

To store five character, there is a need of 3 bit character maximum. This three bit is provided by users. and it reduce the size of data in a file.

So for cost calculation by using fixed size bit

For 20 character = 20 × 3 = 60 bit.

(note: The important thing is: when I am sending the message to other, then she/he encoded this message or when I store the message in a file and letter on when I want to open then how can I remember the bits of each Letter. (i.e 000 means A, 001 for B and so on.). So I have to send the table also with the message :).

So the ASCII code of these character with the code (assigned for user [3 bit]) also is calculated.

So five alphabet take = 5 × 8 bits = 40 bits.

Codes for this five alphabet = 5 × 3 bits = 15 bits.

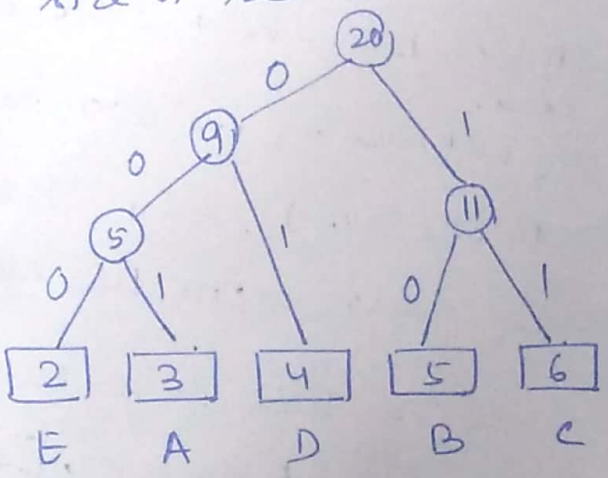
And the total message cost = 20 × 3 bits = 60 bits.

So Total no. of bits required = 115 bits.

now Let us try with variable size coding other wise known as Huffman coding. or prefix code.

The main aim is to reduce the size of the message with table.

Huffman says that we don't required fixed size coding. Because the frequency of characters are not same. Some are less and some are high. Let us see what is Huffman idea. So if we give small size code then the bit size is reduced.



char.	freq.	code.
A	3	001
B	5	10
C	6	11
D	4	01
E	2	000

Huffman code follows optimal merge pattern. Now we have to generate our code. He says that first arrange the alphabet in ascending order as per their frequency. Merge two smallest one and form a node. and we merge again two smallest as shown in above figure.

Now we got an optimal merge Patteren tree. which is a greedy approach. This tree will help us to define the code. where no code is prefix of another one. (i.e prefix code).

on the left hand side edges mark them 0 and on the right hand side edges mark them 1. Now find the code for each alphabet from sort to alphabet path.

Here we observe that some code is three bit, some code is two bit. i.e variable size code. Now if we use this code then what is the size of my message.

i.e

Five alphabet take with ASCII code $= 5 \times 8 = 40$ bit.

Total message cost. $= (3\times3 + 5\times2 + 6\times2 + 4\times2 + 2\times3.)$ $= 45$ bit

codes for this five alphabet $= (3+2+2+2+3) = 12$ bit.

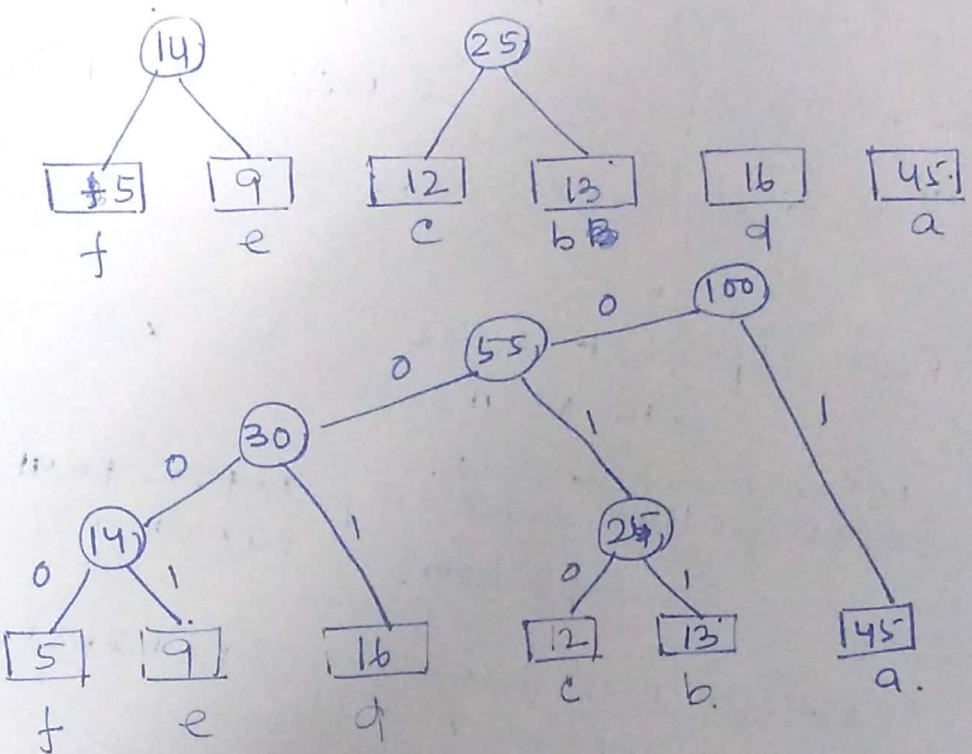
So total no. of bits required $= 97$ bit.

So it was observed that Huffman code takes very less bit to store or sending the message.

Ques: Give the bit to the following character by using Huffman code:

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

Arrange the characters in increasing order of their frequencies.



char.	frequency in thousand	code.
a	45	1
b	13	011
c	12	010
d	16	001
e	9	0001
f	5	0000

So the total bit required to store or transfer the message over a network is

$45 \times 1 + 13 \times 3 + 12 \times 3 + 16 \times 3 + 9 \times 4 + 5 \times 4 = 224$ bit in thousand.

So the formulation for total bit code is

$$B(T) = \sum_{c \in C} freq_c \cdot d_{T_c}$$

where $freq_c$ ← frequency of each character

d_{T_c} ← length of the code of each character.

Hence Huffman invented a greedy algorithm that constructs an optimal prefix called a Huffman code:

```
Huffman (C)
1. n = |C|
2. Q = C
3. for i = 1 to n-1
4.     allocate a new node z
5.     z.left = x = Extract_Min(Q)
6.     z.right = y = Extract_Min(Q)
7.     z.freq = x.freq + y.freq.
8.     Insert (Q, z)
9. return Extract_Min(Q).
```

The total running time of Huffman on a set of n characters is $O(n \lg n)$.

Pre-fix code:

A code is called prefix code or Prefix free code ; it no codeword is a Prefix of another one.

Example - { a = 0, b = 110, c = 10, d = 111 }

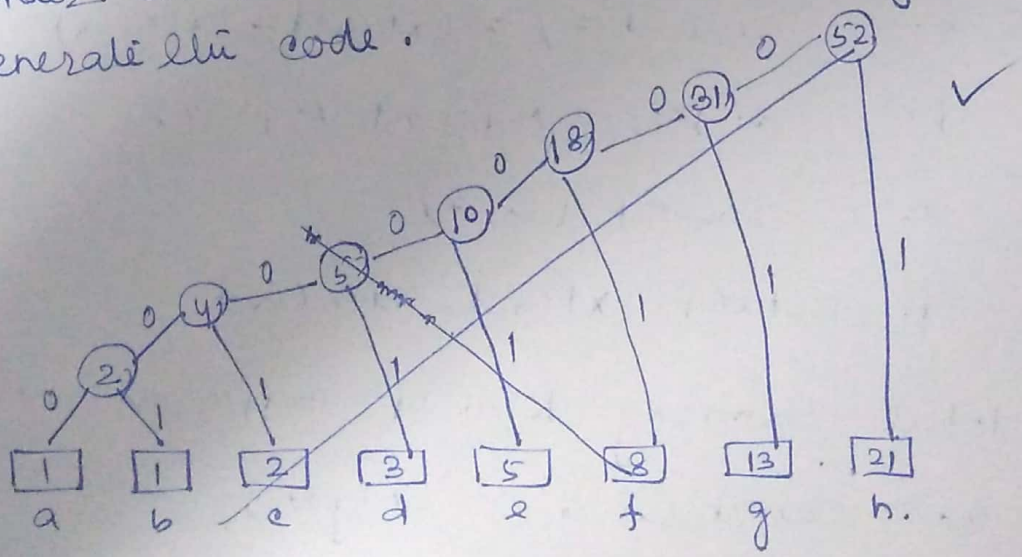
or { a = 1, b = 001, c = 01, d = 000 }.

is a prefix code.

Ques: find the optimal Huffman code for the set of frequency of character.

Character	a	b	c	d	e	f	g	h.
frequency in thousands.	1	1	2	3	5	8	13	21

first arrange these characters in increasing order. on the basis of their frequency. and then generate the code.



Hence the Huffman code of the characters are as follows.

char.	frequency	code	no. of Bits
a	1	0000000	7
b	1	0000001	7
c	2	000001	6
d	3	00001	5
e	5	0001	4
f	8	001	3.
g	13	01	2
h	21	1	1

Hence the total no. of bits required to store the data in a file or send message over internet.

$\Rightarrow (1\times7)+(1\times7)+(2\times6)+(3\times5)+(5\times4)+(8\times3)+(13\times2)+(21\times1)$

$\Rightarrow 7+7+12+15+20+24+26+21 = 132$ bits.

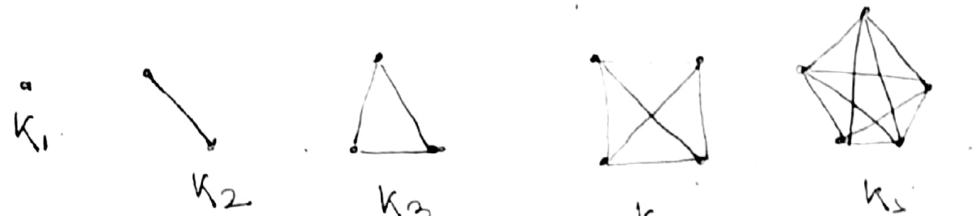
Complete Graph:

Defn: A complete graph is a graph with N vertices and an edge between every two vertices.

i.e → There are no loops.

→ Every two vertices share exactly one edge.

Example of complete Graph:



(edge = 0) (edge = 1) (edge = 3) (edge = 6) (edge = 10)

So the question is how many edges does K_n have?

→ K_n has N vertices.

→ Each vertex has degree N-1;

→ The sum of all degree $N(N-1)$

→ Now, by the help of Handshaking Theorem tells us that

The number of edges in K_N is $\frac{N(N-1)}{2}$.

So $N = 3$ then edge $= \frac{3(2)}{2} = 3$

$N = 4$ then edge $= \frac{4 \cdot 3}{2} = 6$

$N = 5$ then edge $= \frac{5 \cdot 4}{2} = 10$ and so on.

Spanning Tree:

A spanning tree is a subset of Graph G, which has all the vertices covered with minimum possible number of edges. So a spanning tree does not have cycles and it cannot be disconnected.

Properties of Spanning Tree:

→ A connected graph G can have more than one spanning tree.

→ All possible spanning trees of a graph G have the same number of edges and vertices.

→ The spanning tree does not have any cycle.

→ Removing one edge from the spanning tree will make the graph disconnected. i.e the spanning tree is minimally connected.

→ Adding one edge to the spanning tree will create a circuit or loop. i.e the spanning tree is maximally acyclic.

Application of Spanning Tree:

→ Civil network Planning

→ Computer Network Routing Protocol

→ Cluster Analysis.

Minimum Spanning Tree.

For a complete graph of n vertices. the edges = $n \times \frac{(n-1)}{2}$ [by Handshaking Theorem]

A tree is a minimal connected graph.

To span the graph with minimum number of edges is called spanning tree & if the graph total weight is also minimum then it is called Minimum spanning tree.

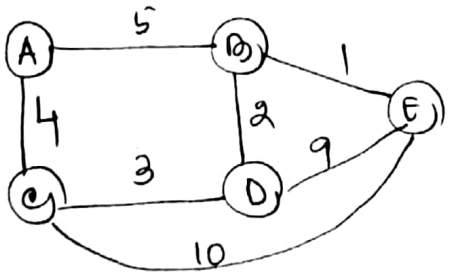
→ If there is n vertices, (n-1) edges then the graph is a tree.

→ A tree contain no cycle.

→ A tree is a minimal connected graph. i.e if we removed any one edge then it is disconnected.

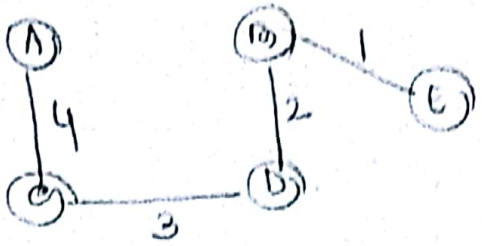
→ Complete graph → in which every vertex is connected directly to another vertex.

Kruskal Algorithm:



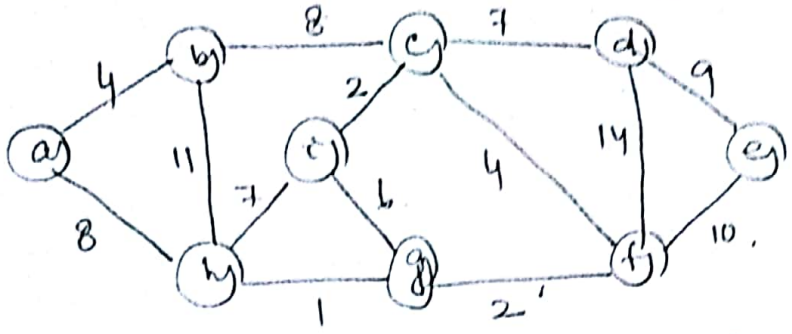
⟨B, E⟩ = 1 ✓
⟨B, D⟩ = 2 ✓
⟨C, D⟩ = 3 ✓
⟨A, C⟩ = 4 ✓
⟨A, B⟩ = 5 × cycle
⟨D, E⟩ = 9 ×
⟨E, C⟩ = 10 ×

The minimum spanning tree.



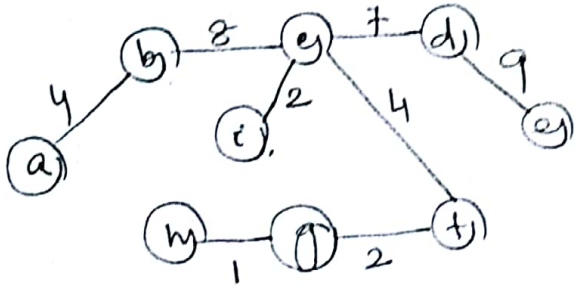
The total cost of spanning tree = 1+2+3+4 = 10.

Q. Find the MST of following graph.



first arrange the edges in ascending order.

$\langle h, g \rangle = 1$ ✓
$\langle e, i \rangle = 2$ ✓
$\langle g, f \rangle = 2$ ✓
$\langle a, b \rangle = 4$ ✓
$\langle e, f \rangle = 4$ ✓
$\langle i, g \rangle = 6$ × cycle.
$\langle c, d \rangle = 7$ ✓
$\langle h, i \rangle = 7$ × cycle.
$\langle b, c \rangle = 8$ ✓
$\langle a, h \rangle = 8$ × cycle.
$\langle d, e \rangle = 9$ ✓
$\langle f, e \rangle = 10$ × cycle
$\langle h, b \rangle = 11$ × cycle
$\langle d, f \rangle = 14$ × cycle.



Hence the total cost.

= 4+1+2+2+4+4+7+8+9.

= (37)

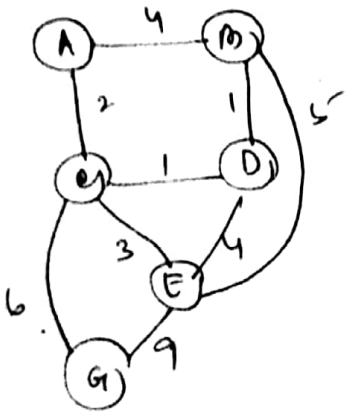
MST_Kruskal (G, w)

1. A = φ
2. for each vertex v ∈ V[G]
3. do Make_Set (v)
4. Sort the edges of E into nondecreasing order by weight w.
5. for each edge (u, v) ∈ E, taken in nondecreasing order by weight.
6. do if Find_Set (u) ≠ Find_Set (v)
7. then A = A ∪ {(u, v)}
   Union (u, v)
8.
9. return A.

The time complexity of Kruskal algorithm is O(E log E by observing the Dense graph that $|E| = |V|^2$ the $\lg|E| = O(\log V)$. Hence the complexity is

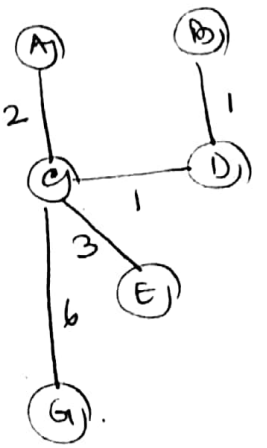
$O(E \lg V)$

Q1. Find MST of the following graph by Kruskal Algo.



⟨C, D⟩ = 1 ✓
⟨B, D⟩ = 1 ✓
⟨A, C⟩ = 2 ✓
⟨C, E⟩ = 3 ✓
⟨D, E⟩ = 4 × cycle.
⟨A, B⟩ = 4 × cycle.
⟨B, E⟩ = 5 × cycle.
⟨C, G⟩ = 6 ✓
⟨E, G⟩ = 9 × cycle.

Total cost = 16



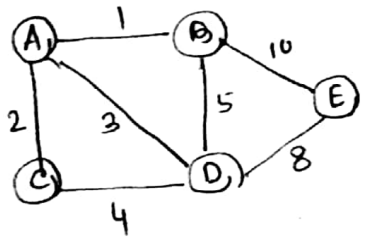
Prim's Algorithm

Prim's Algorithm is a greedy algorithm that finds a minimum spanning tree for a weighted undirected graph. i.e Finds a subset of the edges that forms a tree that include every vertex, where the total weight of all the edges in the tree is minimized.

```
MST-Prims (G, w, r)
1. for each u ← V[G]                    ]
2.       do key[u] = ∞                  ]— O(V)
3.          π[u] = NIL                  ]
4. key[r] — 0                           ]— O(1).
5. Q ← V[G]                             ]— O(1) by Fibonacci Heap. insert.
6. while Q ≠ Φ                          — O(lgV).
7.    do u ← Extract-Min[Q]             — O(lgV).
8.    for each v ∈ Adj[u]
9.        do if v ∈ Q and w(u,v) < key[v]
10.          then π[v] = u
11.               key[v] = w(u,v)      ← decrease key takes O(lgV) time. in Fibonacci heap.
```

O(E) × O(lgV) + O(VlgV) = O(ElgV)

eg.



Q:

0	∞	∞	∞	∞
A	B	C	D	E

step-1 u ← A

{Extract-Min[Q] = A}

Adj(u) = {B C D}

w(A,B) < key(B)

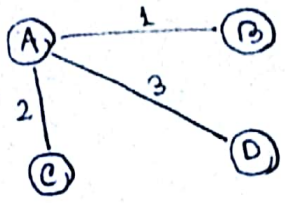
π(B) ← A

B = w(A,B).

w(A C) < key[C]

w(A D) < key[D]

0	1	2	3	∞
A	B	C	D	E



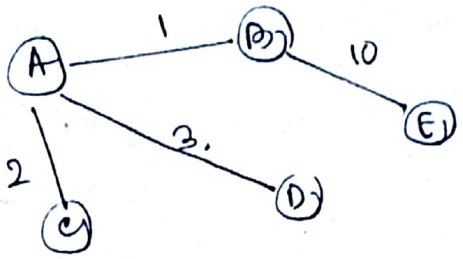
Step-2

Q:

0	1	2	3	∞
A	B	C	D	E

⟹

0	1	2	3	10
A	B	C	D	E



u ← B.

Adj [B] = { D, E }.

w(B, D) < key (D).

5 < 3.

w(B, E) < key (E)

10 < ∞.

E = 10.

Step-3

0	1	2	3	10
A	B	C	D	E

u ← C

Adj (C) = { D }.

w(C, D) < key (D).

4 < 3.

false.

Step-4

0	1	2	3	10
A	B	C	D	E

u ← D.

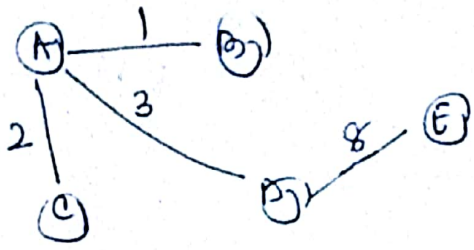
Adj (D) = { E }.

w(D, E) < key (E)

8 < 10.

π(E) = D. & F = 8

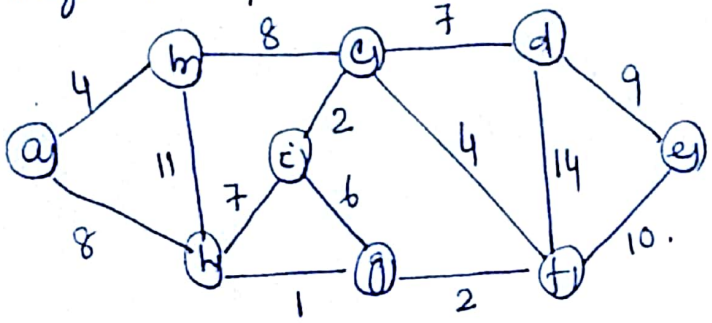
final Q : | 0 | 1 | 2 | 3 | 8 |
A B C D E



The complexity of prims algorithm is

$O(E + V \log V)$.

Q. Find the MST of following graph by using Prim's Algorithm



key

a	b	c	d	e	f	g	h	i

Parent (π)

a	b	c	d	e	f	g	h	i

after step 1 to 3.

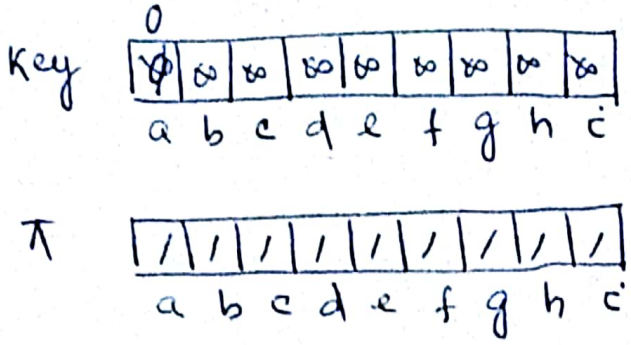
key

∞	∞	∞	∞	∞	∞	∞	∞	∞
a	b	c	d	e	f	g	h	i

π

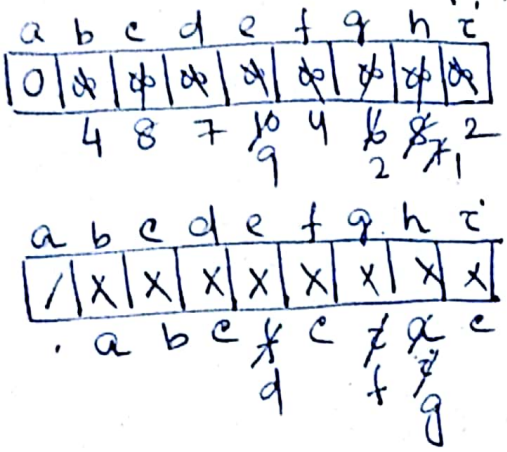
/	/	/	/	/	/	/	/	/
a	b	c	d	e	f	g	h	i

after step-4.



Step-5: $Q = \{a, b, c, d, e, f, g, h, i\}$.

Step-6 to 11 (changes done in Key array, π araray and Q.)



$Q = \{a, b, c, d, e, f, g, h, i\}$.

Iteration-1.

u = Extract-Min (Q) i.e 'a'.

adj [a] = {b, h}.

$w(a,b) < key[b]$

$4 < \infty$ True.

$w(a,h) < key[h]$

$8 < \infty$ True.

Iteration - 2.

u = Extract min (Q) = i.e 'b'.

adj [b] = {. c, h}.

w (b, c) < key [c]

8 < ∞ True.

w (b, h) < key [h]

11 < 8 False.

Iteration - 3

$u \leftarrow$ Extract_Min (Q) = i.e 'c'

adj [c] = {d, f, i}.

w (c, d) < key [d]

7 < ∞ True.

w (c, f) < key [f]

4 < ∞ True.

w (c, i) < key [i]

2 < ∞ True.

Iteration - 4

$u \leftarrow$ Extract_Min (Q) = i.e 'i'

adj [i] = {. h, g}.

w (i, h) < key [h]

7 < 8 True.

w (i, g) < key [g]

6 < ∞ True.

Iteration-6

u = Extract-Min(Q) i.e 'f'

adj[f] = {d, e, g}.

w(f, d) < key[d]

14 < 7 False.

w(f, e) < key[e]

10 < ∞ True.

w(f, g) < key[g]

2 < 6 True.

Iteration-6

u = Extract-Min(Q) i.e 'g'

adj[g] = {h}.

w(g, h) < key[h]

1 < 7 True.

Iteration-7

u ← Extract-Min(Q) i.e 'h'

adj[h] = {ϕ}.

Iteration-7

u ← Extract-Min(Q) i.e 'd'

adj[d] = {e}.

w(d, e) < key[e]

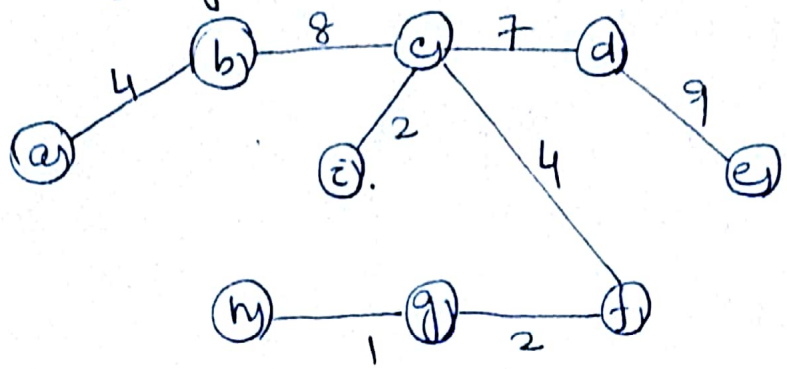
9 < 10 True

Iteration-8 u ← Extract-Min(Q) i.e 'e'

adj[e] = {ϕ}.

now Q = {ϕ}.

So MST of graph is



So total cost = 4+8+7+9+2+4+2+1 = (37)

~~Algorithm of Kruskal's~~

~~MST-Kruskal (G, w)~~

~~1. A = ϕ~~

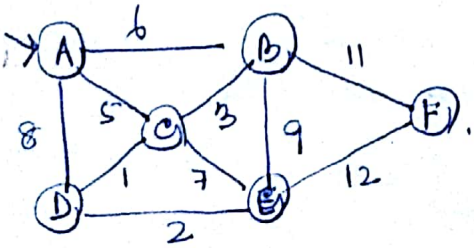
~~2. for each vertex v ∈ G, V~~

~~3. Make set (v)~~

~~4. Sort the edges of G.E into nondecreasing order by weight w.~~

~~5. for each edge (u, v) ∈ G.E, taken in nondecreasing order.~~

Q. Find the MST of following graph using Prim's Algorithm.



Single Source Shortest Path.

The Single Source shortest path in which, we have to find shortest path from a source vertex v to all other vertices in the graph.

Dijkstra's Algorithm solves the single-source shortest paths problem on a weighted directed graph $G = \langle V, E \rangle$ for each case in which all edges weights are nonnegative.

Algorithm of Dijkstra.

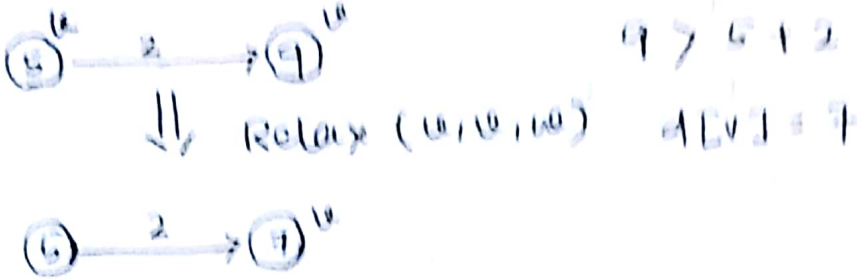
```
Dijkstra (G, w, s)
1. Initialize-Single-Source (G, s)
2.  S = ϕ
3.  Q = G.V
4.  while Q ≠ ϕ
5.      u = Extract-Min (Q)
6.      S = S ∪ {u}
7.      for each veretex v ∈ G.Adj[u]
8.          Relax (u, v, w)

Initialize-Single-Source (G, s)
1. for each vertex v ∈ V.[G]
2.      d[v] = ∞
3.      π[v] = NULL
4  d.[s] = 0.
```

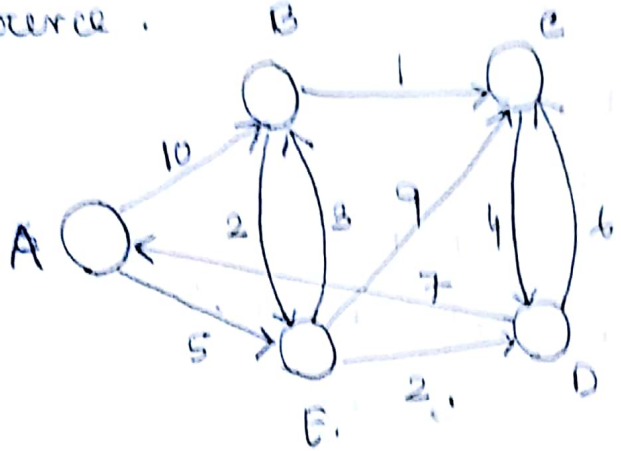
Relax (u, v, w)

1. if d[v] > d[u] + w(u,v)
2. d[v] = d[u] + w(u,v)
3. π[v] = u.

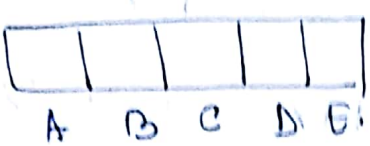
for example.



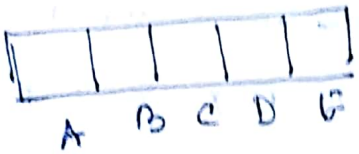
Q. Find the single source shortest path of the following graph. G (V, E) , where s is source. (nonnegative weighted directed)



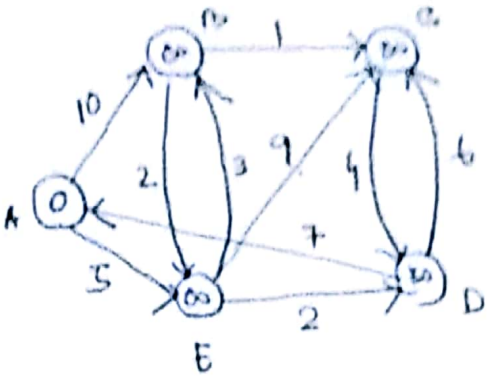
d / key



π



After Step 1



d/key

0	∞	∞	∞	∞
A	B	C	D	E

π

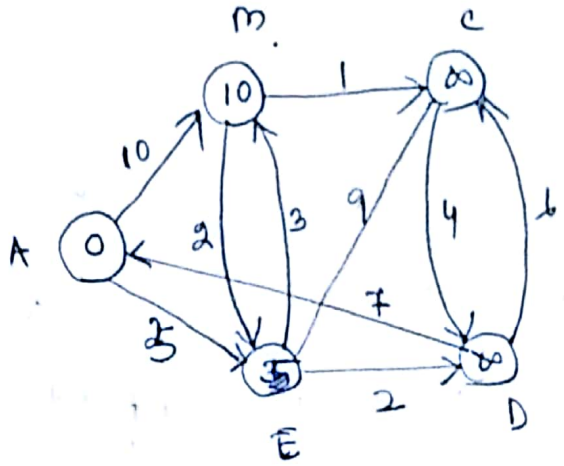
/	/	/	/	/
A	B	C	D	E

After Step-2 & 3

$S = \phi$

$Q = V[G]$ {A, B, C, D, E}

for step 4 to 8.



Step-1

d/key

0	10	∞	∞	05
A	B	C	D	E

π

/	A	/	/	A
A	B	C	D	E

while $Q \neq \phi$

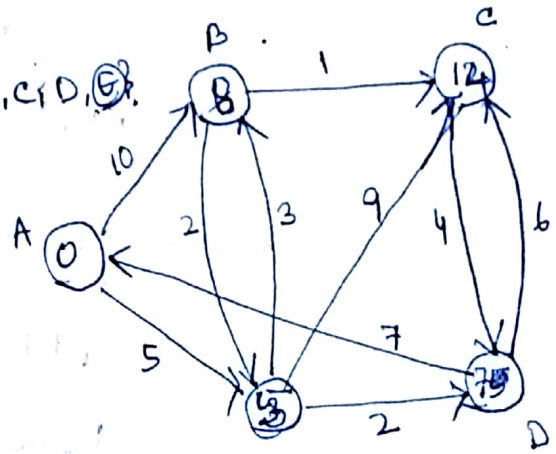
$u = A$. by Extract Min (Q).

adj [A] ← {B, E}

$S = \{A\}$

Relax (A).

$Q = \{B, C, D, E\}$



Step-2

d/key

0	10	∞	∞	05
A	B	C	D	E

				12	07
				E	E

π

/	A	X	X	A
A	B	C	D	E

$u = E$, by Extract Min (Q). E.

$S = \{A, E\}$ adj [E] ← {B, C, D}

Relax (E, B, w)

1. If 10 > ∞ + w(E, B)

   50 + 3 True.

   d[B] = 8

   π[B] = E.

2. Relax (E, C, w).

   If ∞ > 5 + 9 True.

   d[C] = 14

   π[C] = E.

Relax (E, D, w).

If ∞ > 5 + 2 True.

d[D] = 7

π[D] = E.

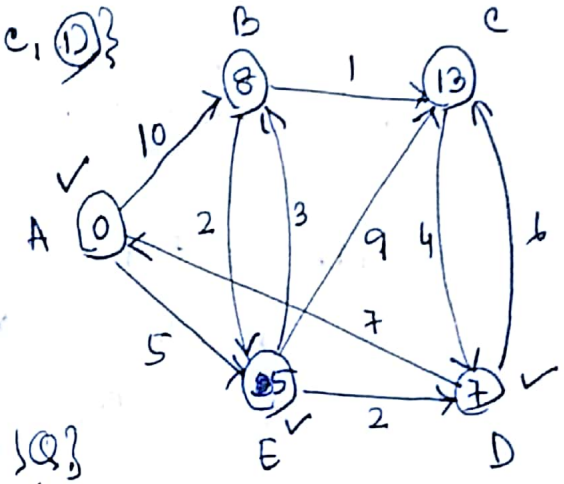
Iteration 3. Q = {B C, D}

d/key

13.				
0	8	14	7	5
A	B	C	D	E

π

D				
/	E	E	E	A
A	B	C	D	E



u = D, for Extract Min {Q}

S = {A, D, E}. adj[D] = {C}

Relax (D, C, w).

If 14 > 7 + 6 True.

d[C] = 13.

π[C] = D.

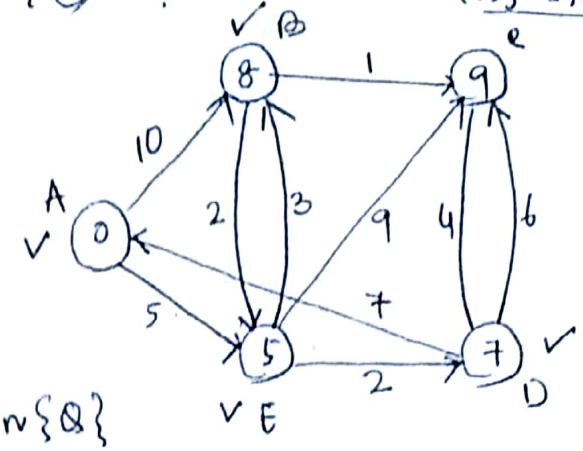
Iteration 4 $Q = \{B, C\}$ SSP (Dijkstra) 3

d/key

0	8	13	7	9
A	B	C	D	E

π

/	E	B	E	A
A	B	C	D	E



$u = B$ by Extract Min $\{Q\}$

$S = \{A, D, E, B\}$ $adj[B] = \{C\}$

Relax (B, C, w).

if $13 > 8+1$ true.

$d[C] = 9$

$\pi[C] = B$.

Iteration 5.

d/key

0	8	9	7	85
A	B	C	D	E

$Q = \{C\}$.

π

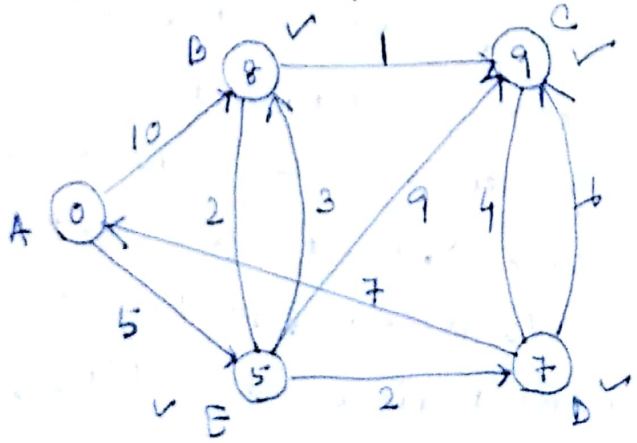
/	E	B	E	A
A	B	C	D	E

$u = C$ $adj[C] = \{D\}$.

$S = \{A, D, E, B, C\}$

So no Relax operation execute.

Finally.



Analysis of Dijkstra's Algorithm.

→ It maintains the min. priority queue Q by calling three priority-queue operations:
  → Insert (implicit in line 3)
  → Extract-Min (implicit in line 5)
  → Decrease-key (implicit in line 8). by using Relax function.

→ The algorithm calls both Insert and Extract-Min once per vertex.
  ⇒ Because each vertex $u \in V$ is added to set S exactly once.
  [i.e. each edge in the adjacency list Adj[u] is examined in the for loop (i.e. line 7-8)]

  ⇒ Since the ~~for loop~~ total number of edges in all the adjacency lists is $|E|$, this for loop iterates a total of $|E|$ times, thus the algorithm decreasekey used at most $|E|$ times overall.

→ The running time of Dijkstra's algorithm depends on, how we implement the priority queue.

Case-1 If Q is unsorted linear array.
  → Each Insert and Decrease key operations take $O(1)$ times. (for $|E|$ such operations)
  → Each Extract-Min operation takes $O(V)$ time. (for V such operations)

For a total time of $O(V^2 + E) = O(V^2)$

Case-2 If Q is as a Binary heap

Then Extract Min operation takes $O(\lg V)$ time for V such operations

Decrease key (in Relax) takes $O(\lg V)$ time for $|E|$ such operations.

Hence the running time of the Dijkstra Algorithm with Binary Heap is $O((V+E)\lg V)$

$\simeq O(E \lg V)$ if all the vertex are reachable from the source.

Case-3 If Q as a binary heap.

Then Extract Min takes $O(\lg V)$ time for V such operations.

Decrease key (in Relax) takes $O(1)$ amortized time for $|E|$ such operation.

Hence the running time of the Dijkstra Algorithm with fibonacci Heap is $O(V \lg V) + O(E)$.

$\simeq O(E + V \lg V)$

The Bellman-Ford Algorithm.

→ The Bellman-Ford algorithm solves the single-source shortest-paths problem in the general case in which edge weights may be negative.

→ Given a weighted, directed graph $G(V, E)$ with source 's' and weight w, the Bellman-Ford algorithm returns a boolean value indicating whether or not there is a negative-weight cycle that is reachable from the source.

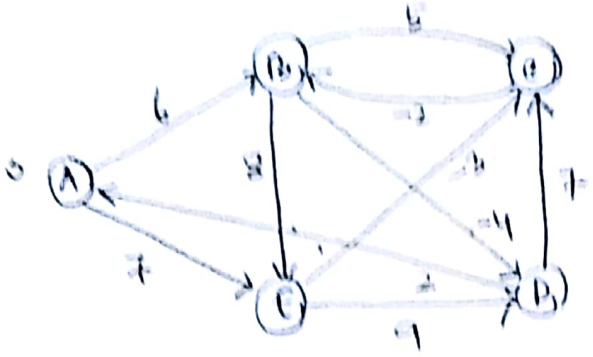
→ If there is such a cycle, the algorithm indicates that no solution exists.

→ If there is no such cycle, the algorithm produces the shortest paths and their weights.

Algorithm of Bellman-Ford

```
Bellman-Ford (G, w, s)
1. Initialize-Single-Source (G, s)
2. for i = 1 to |V[G]| - 1
3.     for each edge (u,v) ∈ E[G]
4.         Relax (u, v, w)
5. for each edge (u,v) ∈ E[G]
6.     if d[v] > d[u] + w(u,v)
7.         return False
8. return true.
```

The execution of Bellman Ford Algorithm on a graph with 5 vertices is given below.



depth. | 0 | ∞ | ∞ | ∞ | ∞ |
A B C D E.

π | N | N | N | N | N |
A B C D E

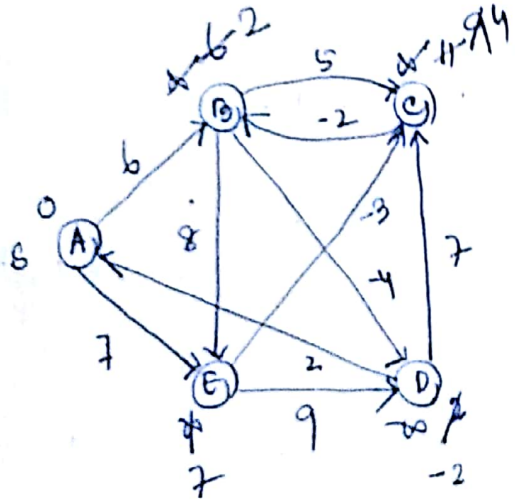
After the execution of line 1.

now execute line 2 to 4

The i^th loop execute for 4 times, i.e V[G] - 1

depth. | 0 | ∞ | ∞ | ∞ | ∞ |
A B C D E

π | N | N | N | N | N |
A B C D E.

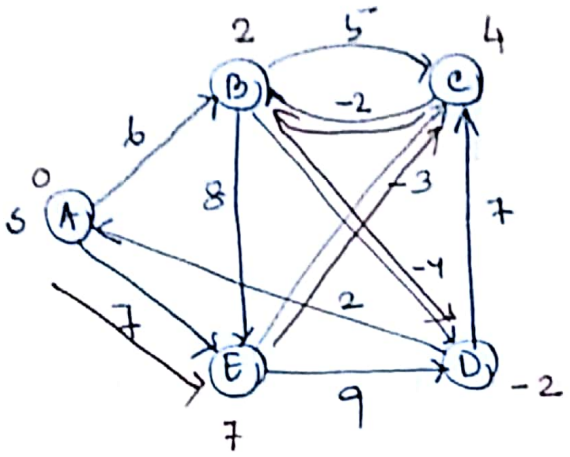


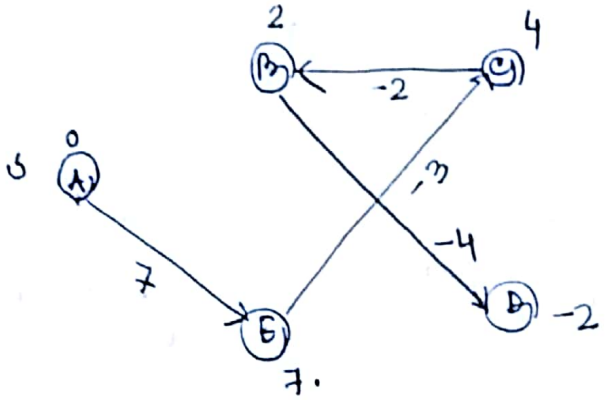
edge.	IT-1	IT-2	IT-3	IT-4
(A, B)	C	NC	NC	NC
(A, E)	C	NC	NC	NC
(B, C)	C	NC	NC	NC
(B, D)	C	NC	C	NC
(B, E)	NC	NC	NC	NC
(C, B)	NC	C	NC	NC
(D, C)	C	NC	NC	NC
(D, A)	NA	NA	NA	NA.
(E, C)	C	NC	NC	NC
(E, D)	NC.	NC	NC	NC.

Hence

depth. | 0 | 2 | 4 | -2 | 7 |
A B C D E

π | N | C | E | B | A |
A B C D E





After executing line 5 to 7 the algorithm return True. for the above example.

The Bellman-ford algorithm runs in time $O(VE)$. Since the initialization (line 1) takes $\Theta(V)$ time. each of the $|V|-1$ passes over the edges in lines 2-4 take $\Theta(E)$ time, and the for loop line 5-7 takes $O(E)$ time.

$\therefore$ $O(V) + \Theta(VE) + O(E) = O(VE)$ times.