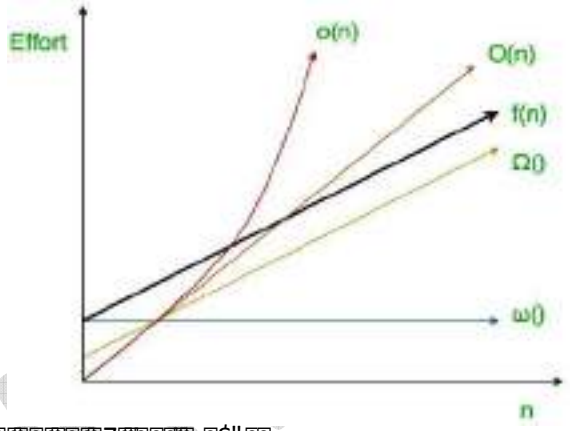
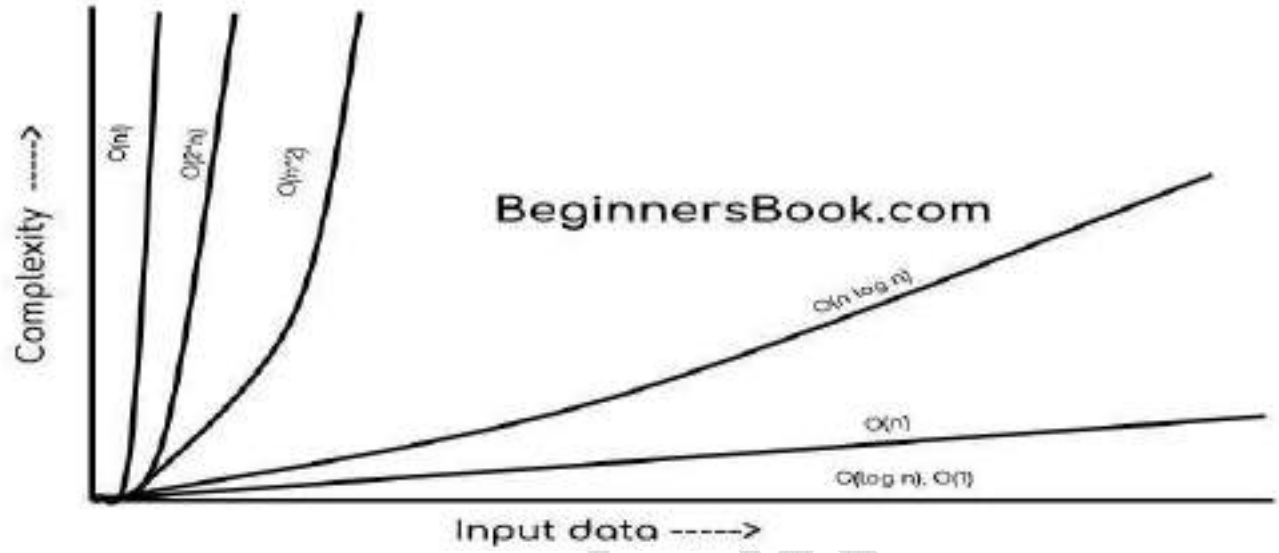
- [illegible]
- [illegible]
- [illegible]
- [illegible]

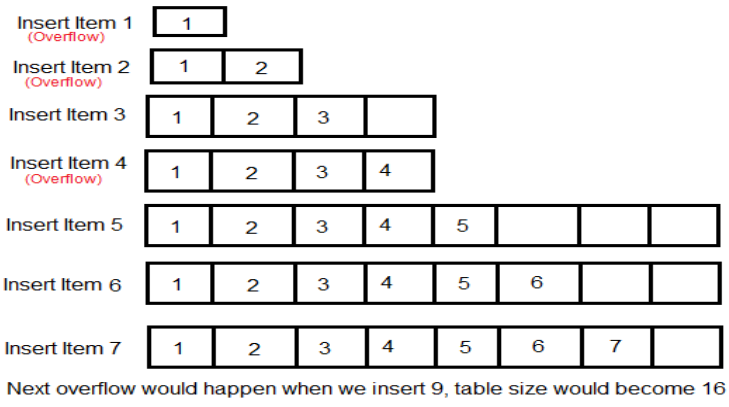
$$\frac{\sum_{i=1}^{n} p(i)}{(n+1)}$$

$$\frac{P((n+1)*(n+2)/2)}{(n+1)}$$


Effort
o(n)
O(n)
f(n)
Ω()
ω()
n


Complexity ----->
O(n)
O(n^2)
O(n^3)
BeginnersBook.com
O(n log n)
O(n)
O(log n), O(1)
Input data ----->
EARLY ACCESS

Initially table is empty and size is 0



Item No.	1	2	3	4	5	6	7	8	9	10	.....
Table Size	1	2	4	4	8	8	8	8	16	16	.....
Cost	1	2	3	1	5	1	1	1	9	1	.....

$$\text{Amortized Cost} = \frac{(1 + 2 + 3 + 5 + 1 + 1 + 9 + 1 \ldots)}{n}$$

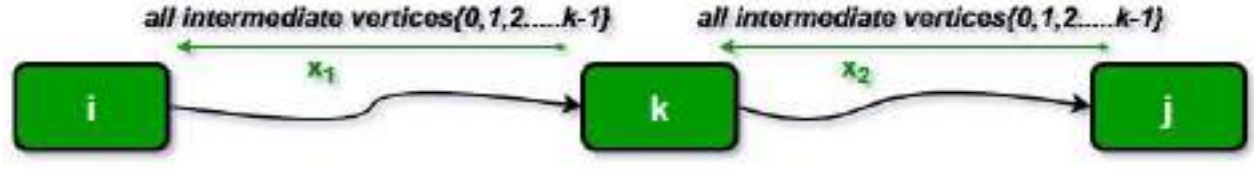
We can simplify the above series by breaking terms 2, 3, 5, 9.. into two as (1+1), (1+2), (1+4), (1+8)

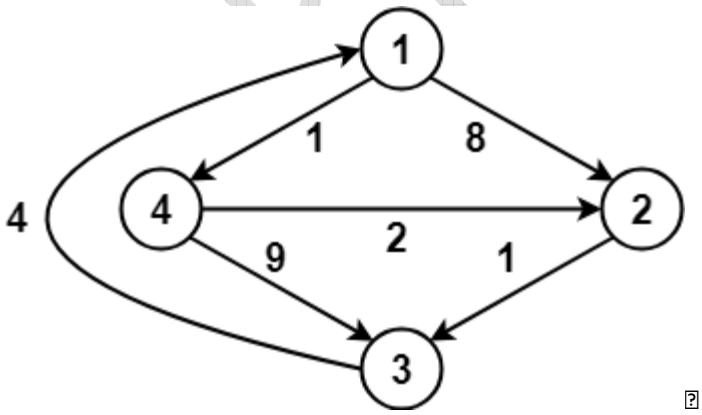
$$\text{Amortized Cost} = \frac{[\overbrace{(1 + 1 + 1 + 1 \ldots)}^{n \text{ terms}} + \overbrace{(1 + 2 + 4 + \ldots)}^{\lfloor \text{Log}_2(n-1) \rfloor + 1 \text{ terms}}]}{n}$$

$$<= \frac{[n + 2n]}{n}$$

$$<= 3$$

$$\text{Amortized Cost} = O(1)$$





[illegible]

$$D_0 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 8 & \infty & 1 \\ 2 & \infty & 0 & 1 & \infty \\ 3 & 4 & \infty & 0 & \infty \\ 4 & \infty & 2 & 9 & 0 \end{array}$$

[illegible]

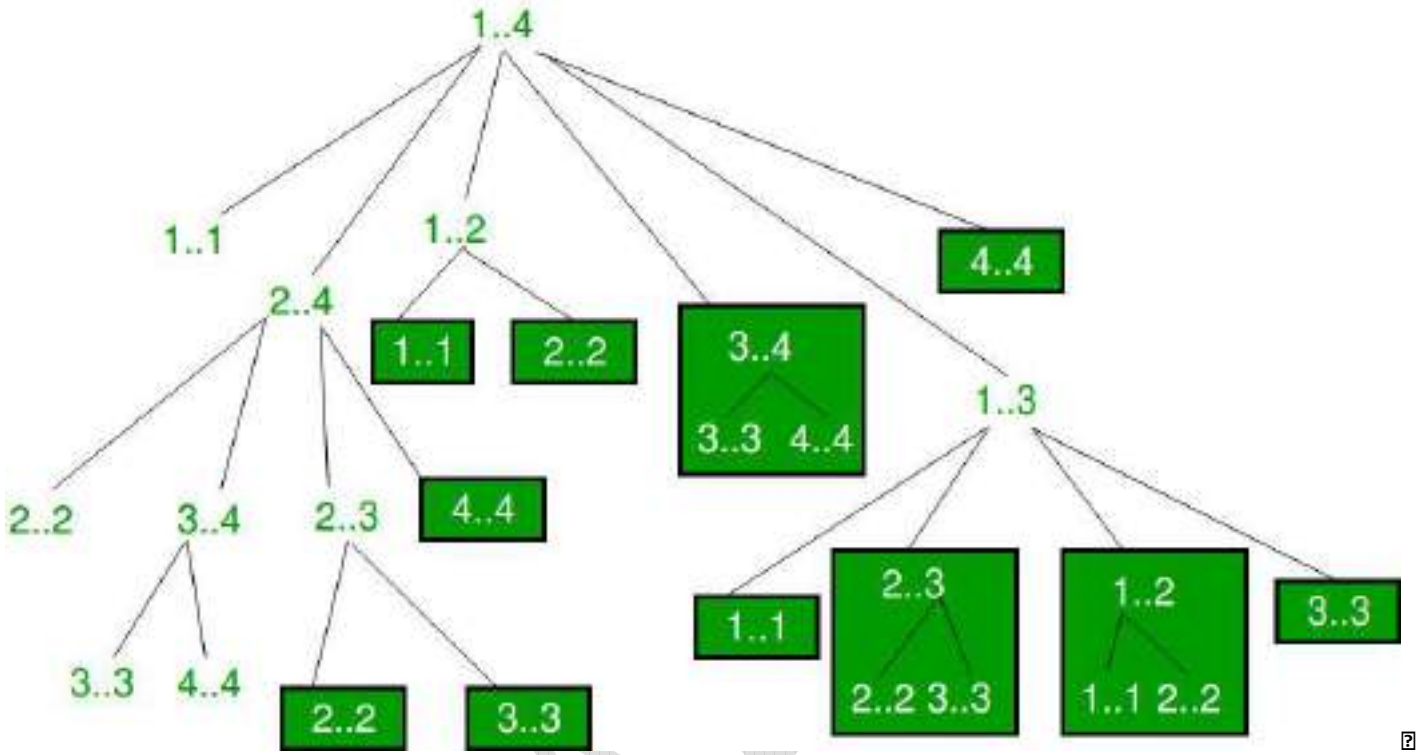
$$D_1 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 8 & \infty & 1 \\ 2 & \infty & 0 & 1 & \infty \\ 3 & 4 & 12 & 0 & 5 \\ 4 & \infty & 2 & 9 & 0 \end{array}$$

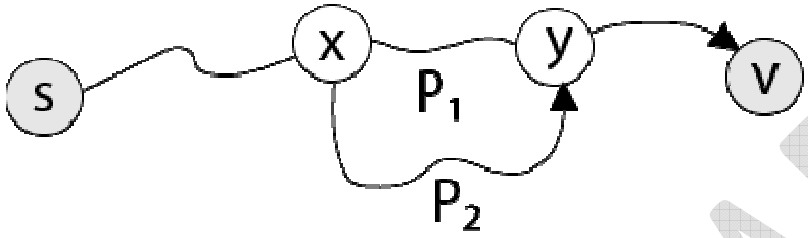
$$D_3 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 8 & 9 & 1 \\ 2 & 5 & 0 & 1 & 6 \\ 3 & 4 & 12 & 0 & 5 \\ 4 & 7 & 2 & 3 & 0 \end{array}$$

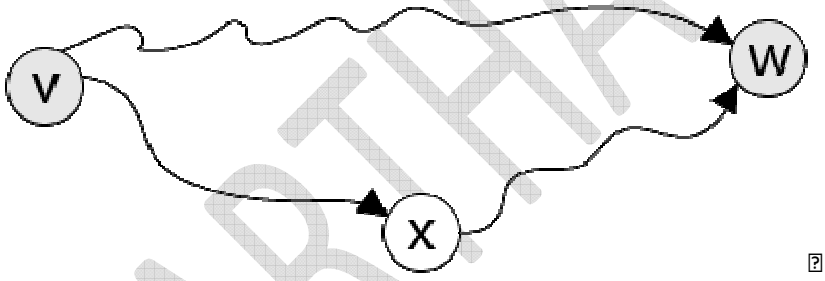
$$D_2 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 8 & 9 & 1 \\ 2 & \infty & 0 & 1 & \infty \\ 3 & 4 & 12 & 0 & 5 \\ 4 & \infty & 2 & 3 & 0 \end{array}$$

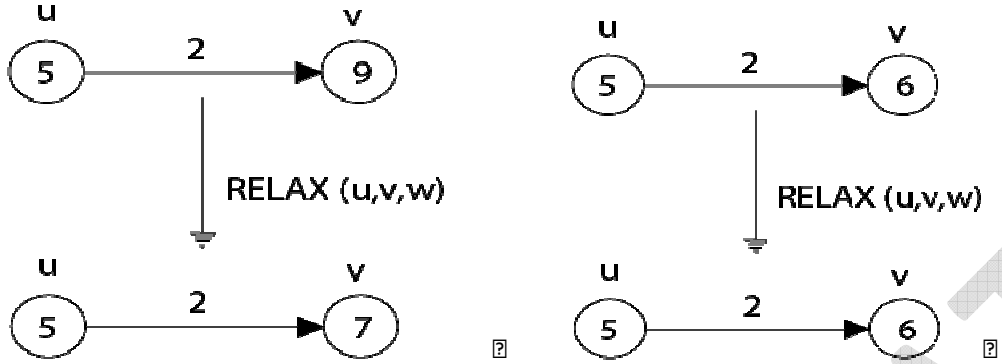
$$D_4 = \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 3 & 4 & 1 \\ 2 & 5 & 0 & 1 & 6 \\ 3 & 4 & 7 & 0 & 5 \\ 4 & 7 & 2 & 3 & 0 \end{array}$$

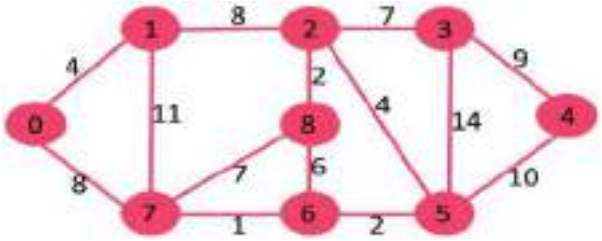
[illegible]

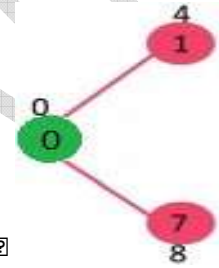

1..4
1..1
1..2
4..4
2..4
1..1
2..2
3..4
1..3
3..3 4..4
2..2
3..4
2..3
4..4
1..1
2..3
1..2
3..3
3..3
4..4
2..2
3..3
2..2 3..3
1..1 2..2

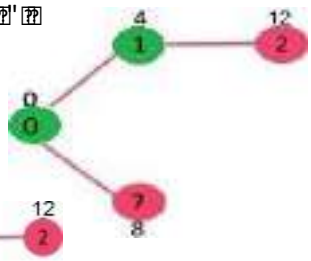


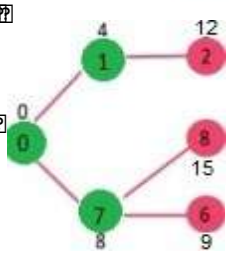


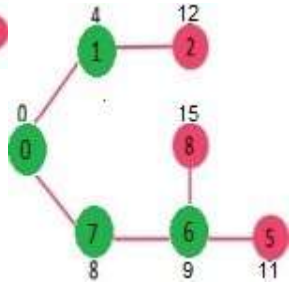


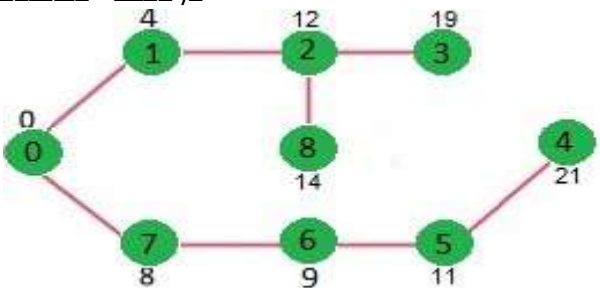


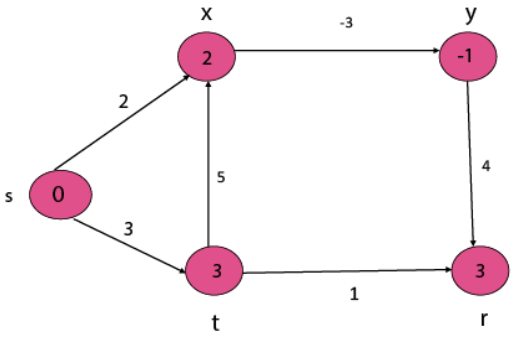


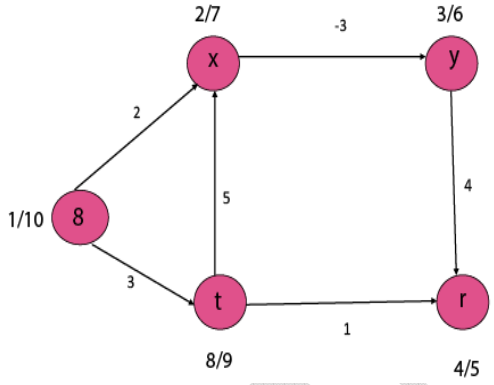


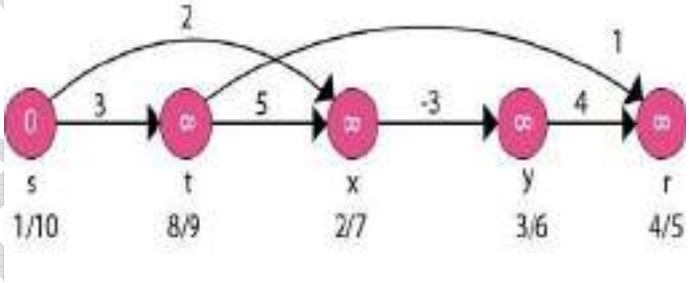




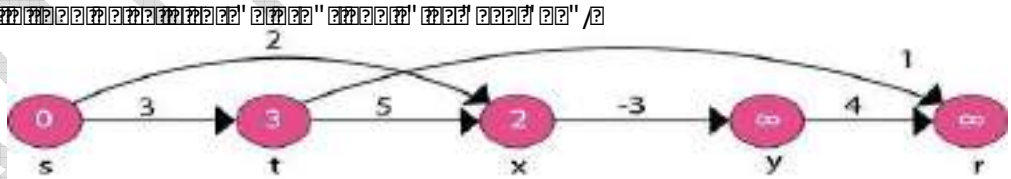


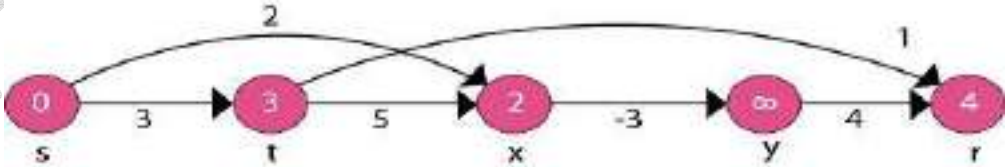


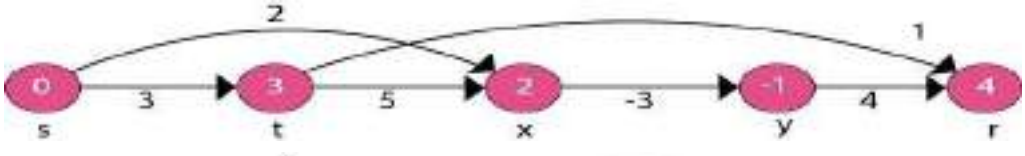


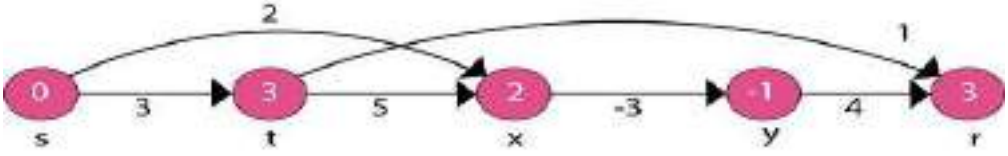


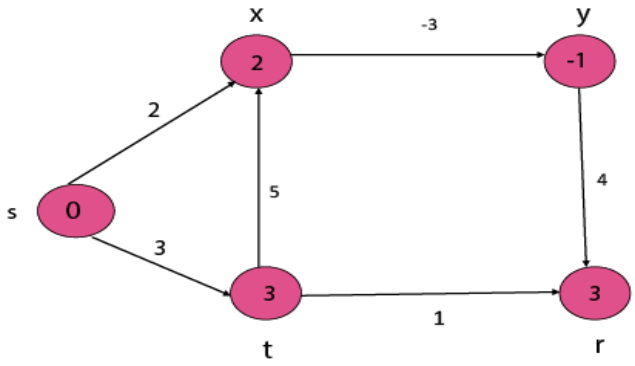
Initialize Single Source

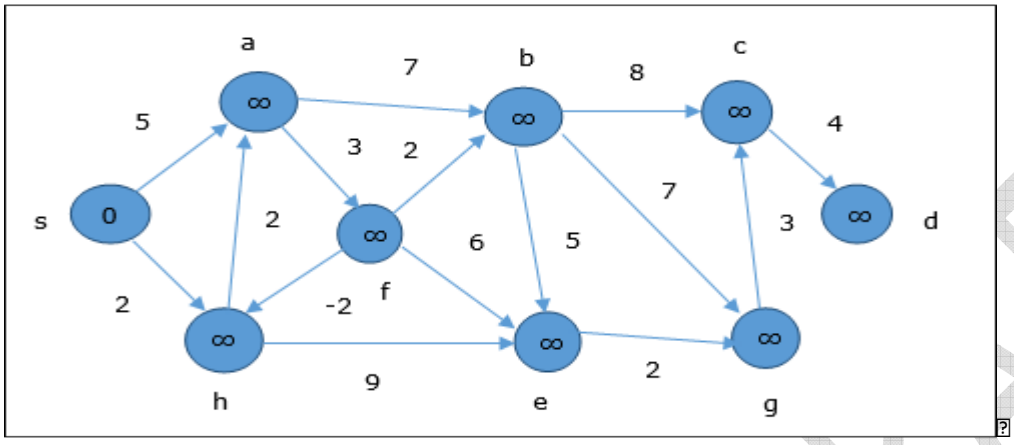


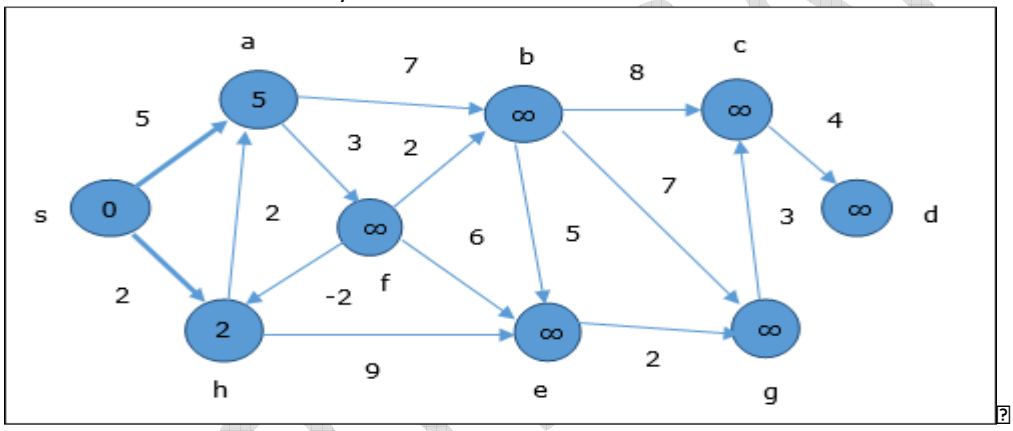


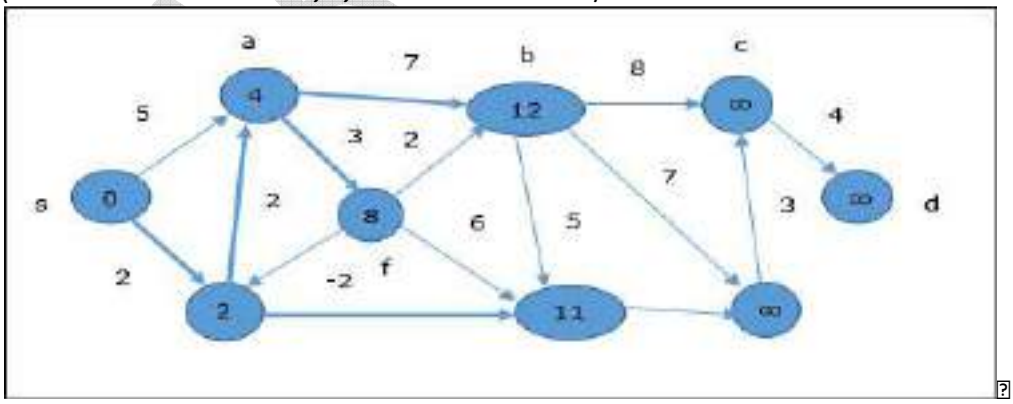




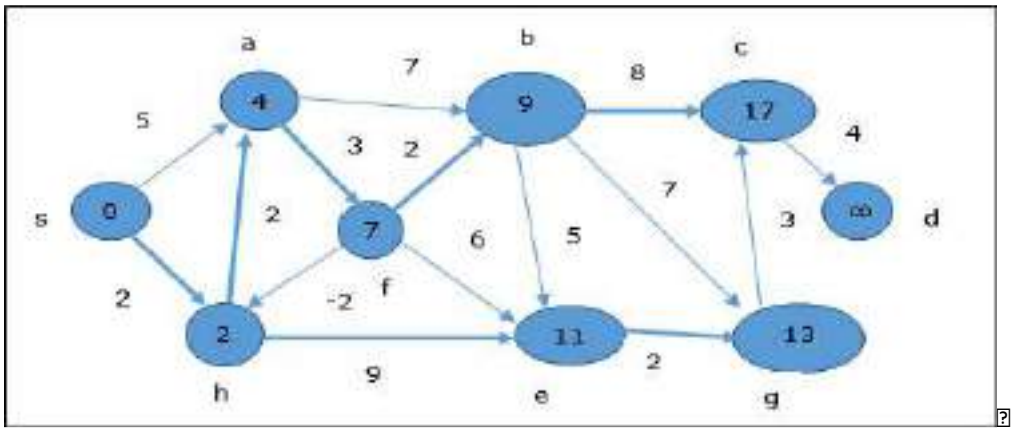


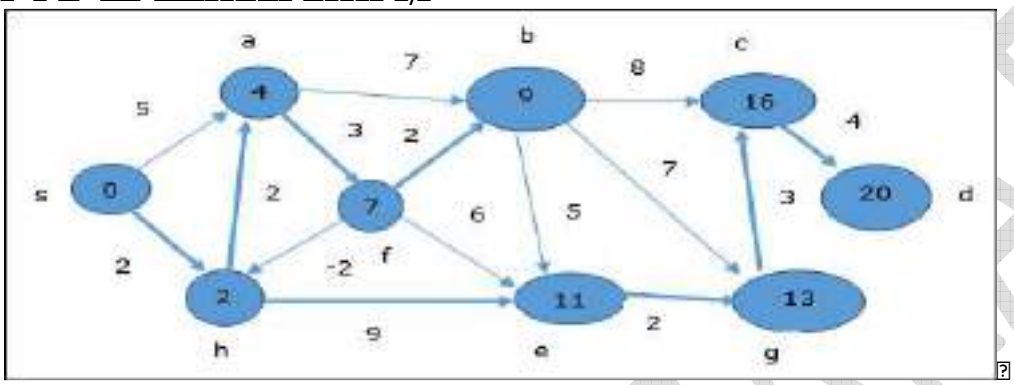






Following the same logic, in this step vertices ***b, f, c*** and ***g*** are updated.


a
b
c
4
9
17
5
7
8
4
3
2
s
0
2
7
6
5
3
∞
d
2
-2
f
2
11
13
2
9
h
e
g


a
b
c
4
9
16
5
7
8
4
3
2
s
0
2
7
6
5
3
20
d
2
-2
f
2
11
13
2
9
h
e
g

=

[illegible]

[illegible]

[illegible]

- [illegible]
- [illegible]
- [illegible]

[illegible]

- [illegible]
- [illegible]

[illegible]

[illegible] v_i/w_i [illegible]

6/ [illegible]
</ [illegible]
B/ [illegible]
_/ [illegible]
70 [illegible]

[illegible]

[illegible]

i.e., $5 \times \frac{5}{5} + 20 \times \frac{20}{20} + 40 \times \frac{35}{40}$

Weight = 5 + 20 + 35 = 60

Maximum Value:-

$30 \times \frac{5}{5} + 100 \times \frac{20}{20} + 160 \times \frac{35}{40}$

= 30 + 100 + 140 = 270 (Minimum Cost)

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. $p_i = \frac{v_i}{w_i}$

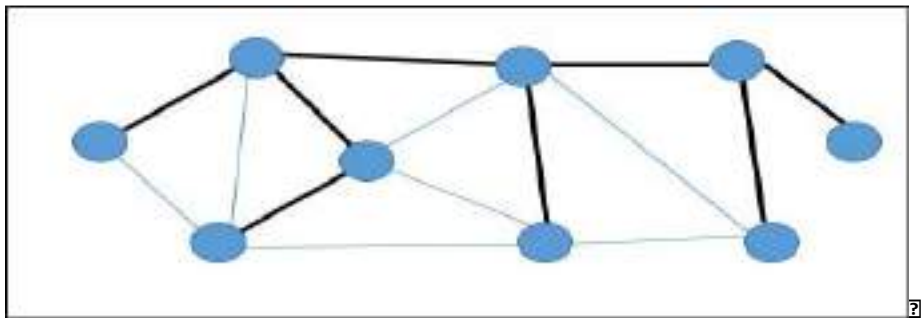
ITEM	w_i	v_i	$p_i = \frac{v_i}{w_i}$
I_1	5	30	6
I_2	10	20	2
I_3	20	100	5
I_4	30	90	3
I_5	40	160	4

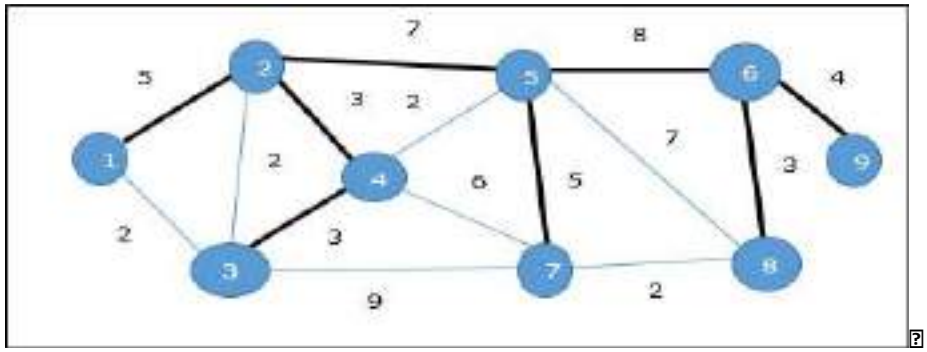
ITEM	w_i	v_i	$p_i = \frac{v_i}{w_i}$
I_1	5	30	6
I_3	20	100	5
I_5	40	160	4
I_4	30	90	3
I_2	10	20	2

Weight Limit (i):	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1\ v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2\ v_2 = 6$	0	1	6	7	7	7	7	7	7	7	7	7
$w_3 = 5\ v_3 = 18$	0	1	6	7	7	18	19	24	25	25	25	25
$w_4 = 6\ v_4 = 22$	0											
$w_5 = 7\ v_5 = 28$	0											

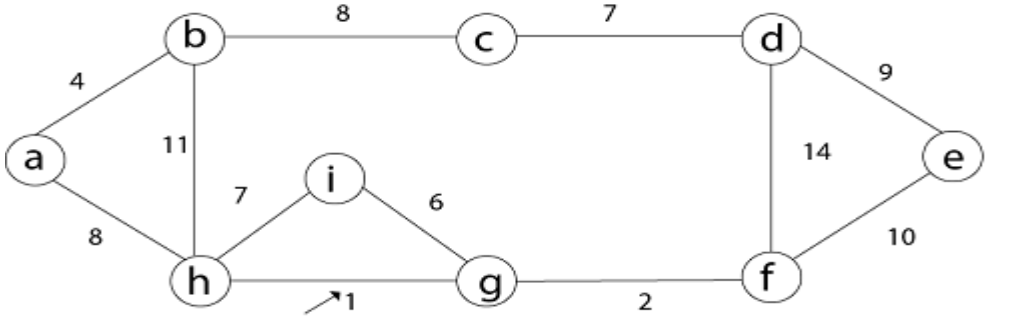
Weight Limit (i):	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1\ v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2\ v_2 = 6$	0	1	6	7	7	7	7	7	7	7	7	7
$w_3 = 5\ v_3 = 18$	0	1	6	7	7	18	19	24	25	25	25	25
$w_4 = 6\ v_4 = 22$	0	1	6	7	7	18	22	24	28	29	29	40
$w_5 = 7\ v_5 = 28$	0											

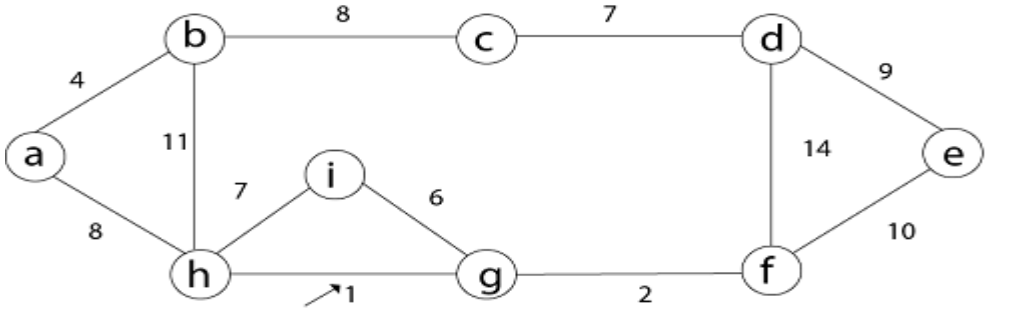
Weight Limit (i):	0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1\ v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2\ v_2 = 6$	0	1	6	7	7	7	7	7	7	7	7	7
$w_3 = 5\ v_3 = 18$	0	1	6	7	7	18	19	24	25	25	25	25
$w_4 = 6\ v_4 = 22$	0	1	6	7	7	18	22	24	28	29	29	40
$w_5 = 7\ v_5 = 28$	0	1	6	7	7	18	22	28	29	34	35	40

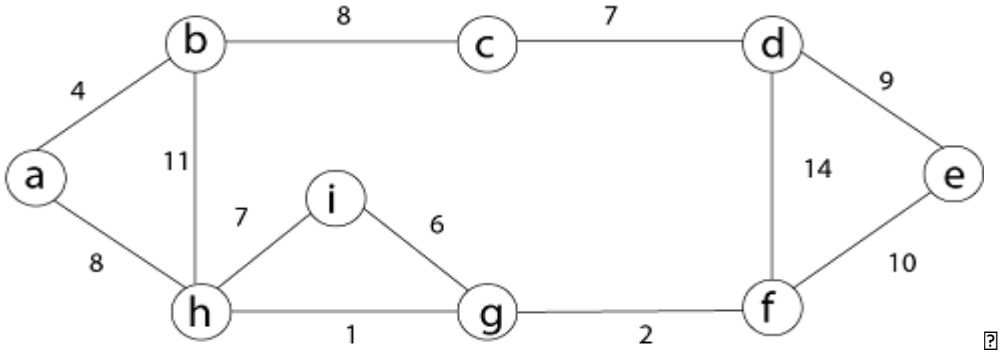


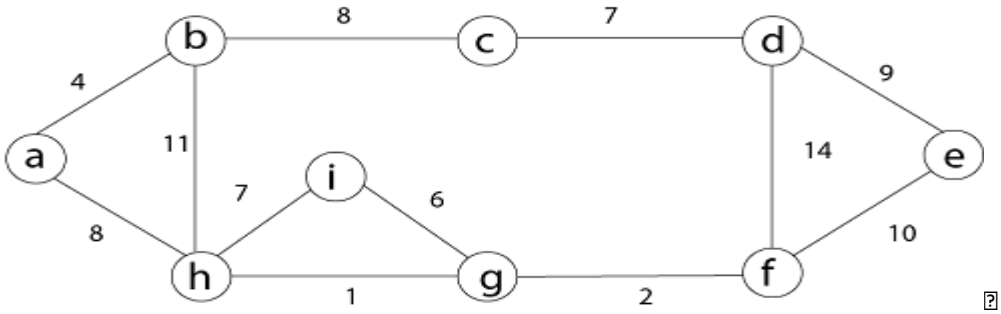

7
8
5
2
5
6
4
3
2
7
1
2
4
6
5
3
9
2
3
3
7
8
9
2

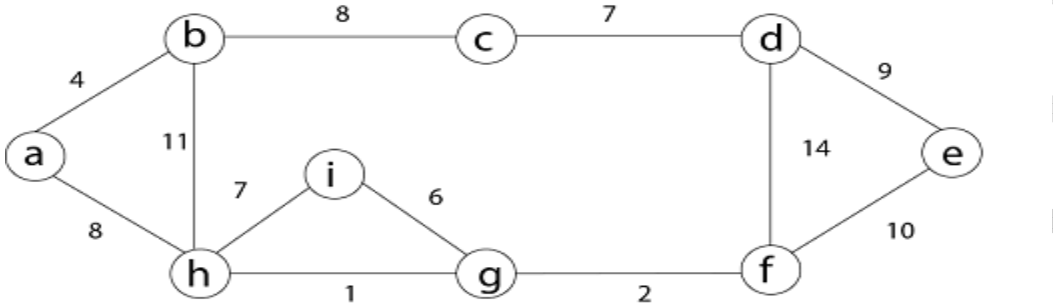
Weight	Source	Destination
1	h	g
2	g	f
4	a	b
6	i	g
7	h	i
7	c	d
8	b	c
8	a	h
9	d	e
10	e	f
11	b	h
14	d	f

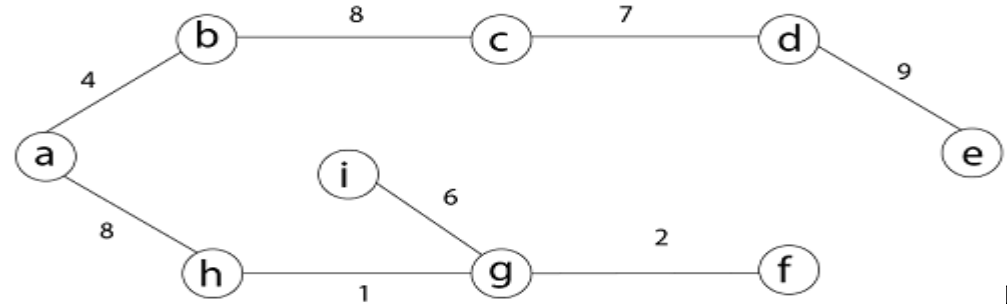


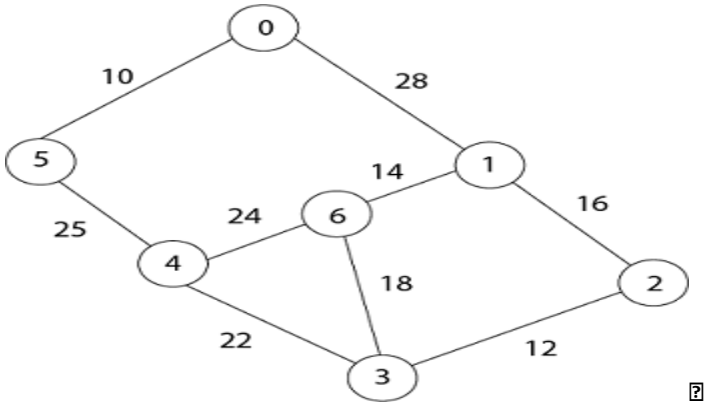





a
b
c
d
e
f
g
h
i
4
8
8
11
7
1
6
7
2
14
9
10

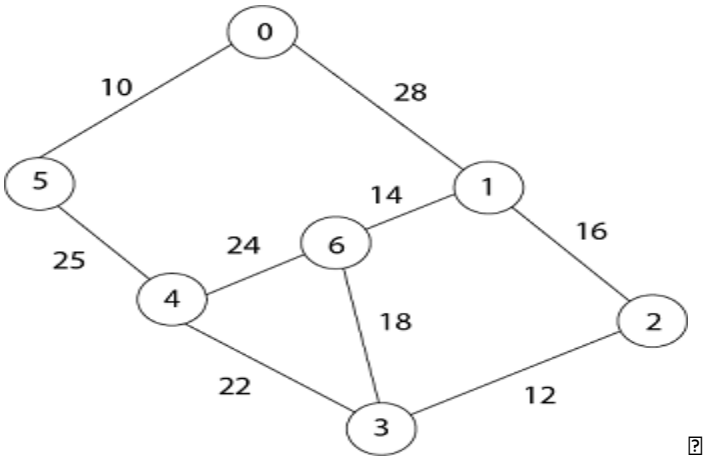

a
b
c
d
e
f
g
h
i
4
8
8
11
7
1
6
7
2
14
9
10


a
b
c
d
e
f
g
h
i
4
8
8
6
1
2
9



Vertex	0	1	2	3	4	5	6
Key Value	0	∞	∞	∞	∞	∞	∞
Parent	NIL	NIL	NIL	NIL	NIL	NIL	NIL

Vertex	0	1	2	3	4	5	6
Key Value	0	28	∞	∞	∞	10	∞
Parent	NIL	0	NIL	NIL	NIL	0	NIL



</ _
B/ _
_/ #"
d/ 0
Q/ 0

0 "

u

0 e9 h (G 9" ' "

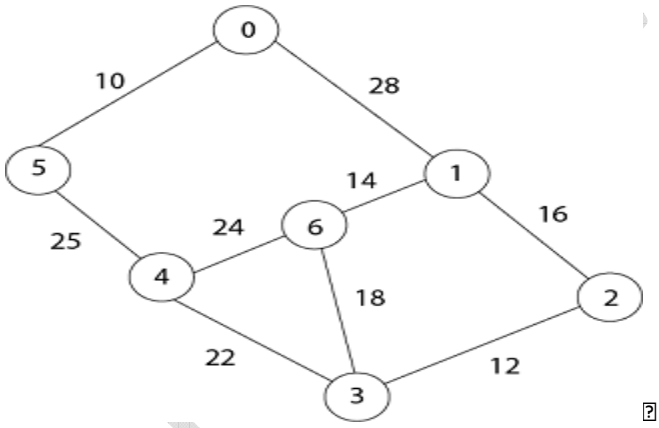
Vertex	0	1	2	3	4	5	6
Key Value	0	28	12	22	25	10	18
Parent	NIL	0	3	4	5	0	3

6/ e9 h (
</
B/ 6<
_/ 0 "
d/ >
Q/ B "
[/ $ 6"
\/ 0 6
: / 0 6

"

u

Vertex	0	1	2	3	4	5	6
Key Value	0	16	12	22	25	10	18
Parent	NIL	2	3	4	5	0	3



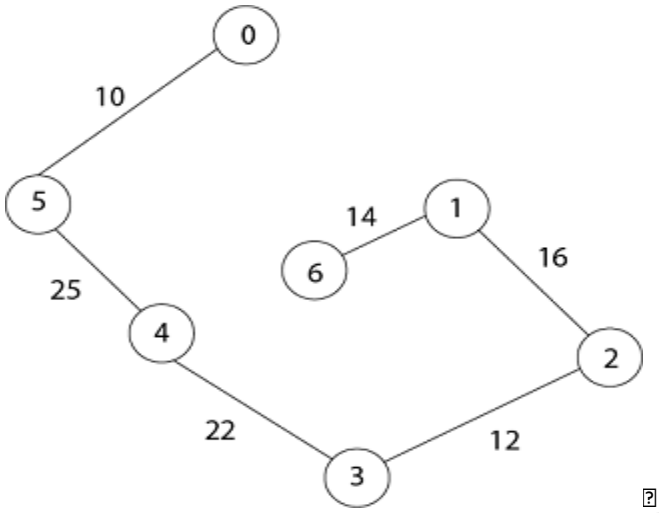
0 e9 h (G 9" ' "

6/ >
</ 7 '
B/ $"
_/ 0
d/ 0

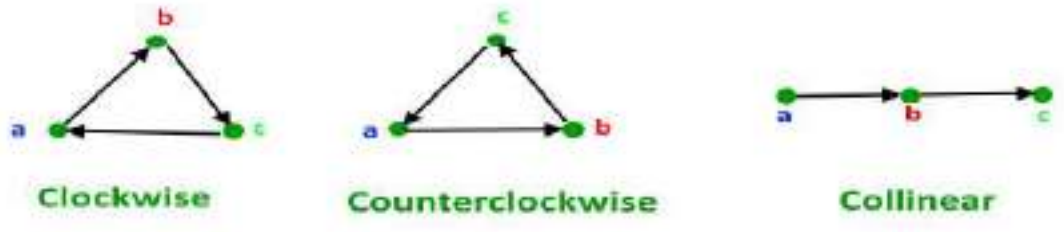
) "

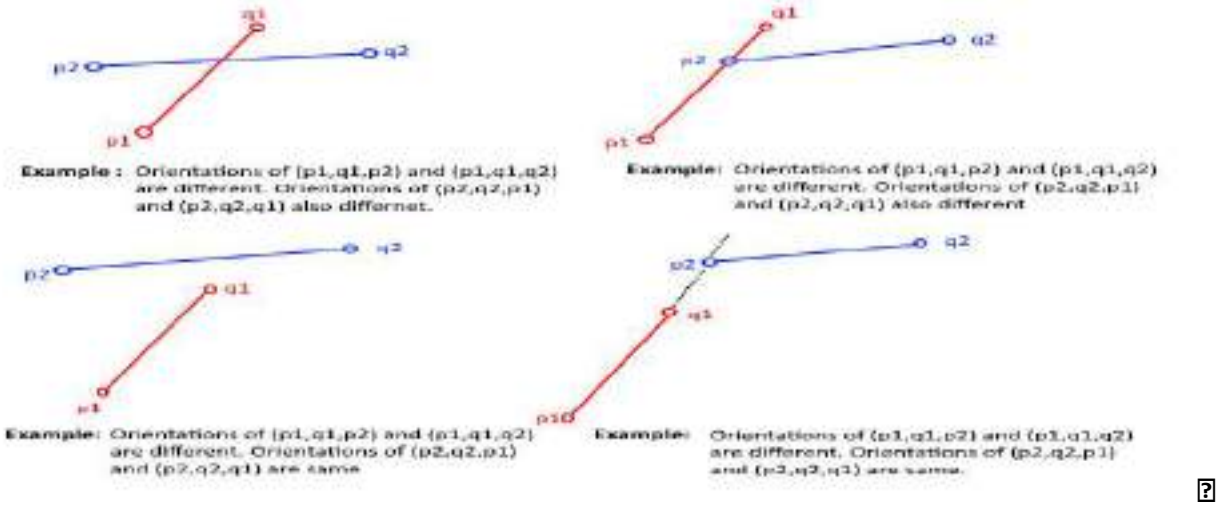
v

Vertex	0	1	2	3	4	5	6
Key Value	0	16	12	22	25	10	14
Parent	NIL	2	3	4	5	0	1



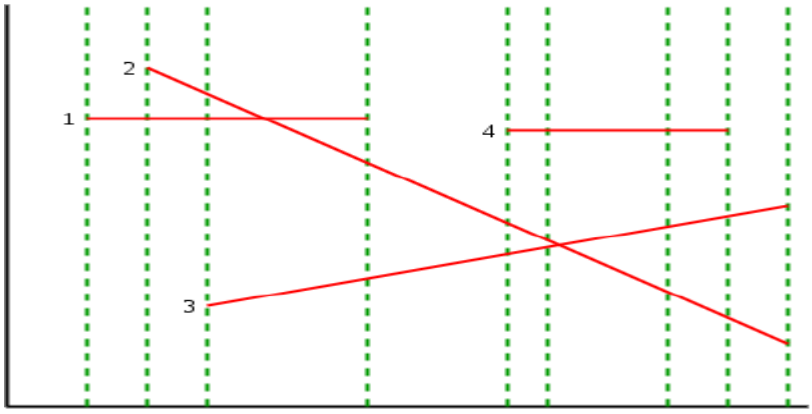
Dynamic Programming	Greedy Method
1. Dynamic Programming is used to obtain the optimal solution.	1. Greedy Method is also used to get the optimal solution.
2. In Dynamic Programming, we choose at each step, but the choice may depend on the solution to sub-problems.	2. In a greedy Algorithm, we make whatever choice seems best at the moment and then solve the sub-problems arising after the choice is made.
3. Less efficient as compared to a greedy approach	3. More efficient as compared to a greedy approach
4. Example: 0/1 Knapsack	4. Example: Fractional Knapsack
5. It is guaranteed that Dynamic Programming will generate an optimal solution using Principle of Optimality.	5. In Greedy Method, there is no such guarantee of getting Optimal Solution.



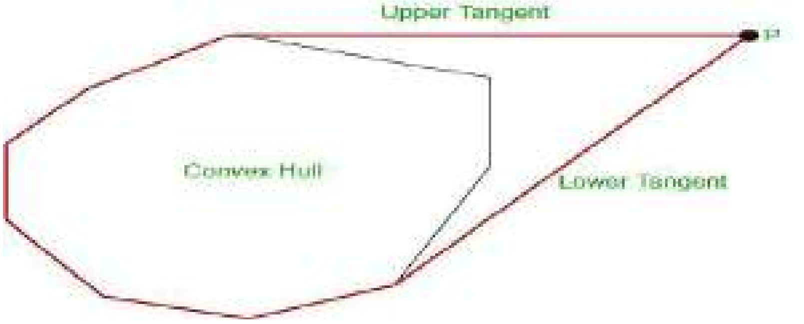


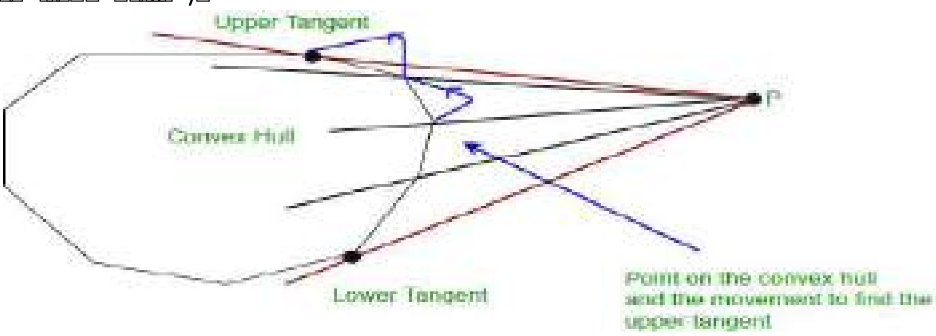
Example: All points are collinear. The x-projections of (p1,q1) and (p2,q2) intersect. The y-projection of (p1,q1) and (p2,q2) also intersect

Example: All points are collinear .The x-projections of (p1,q1) and (p2,q2) not intersect. The y-projection of (p1,q1) and (p2,q2) do not intersect

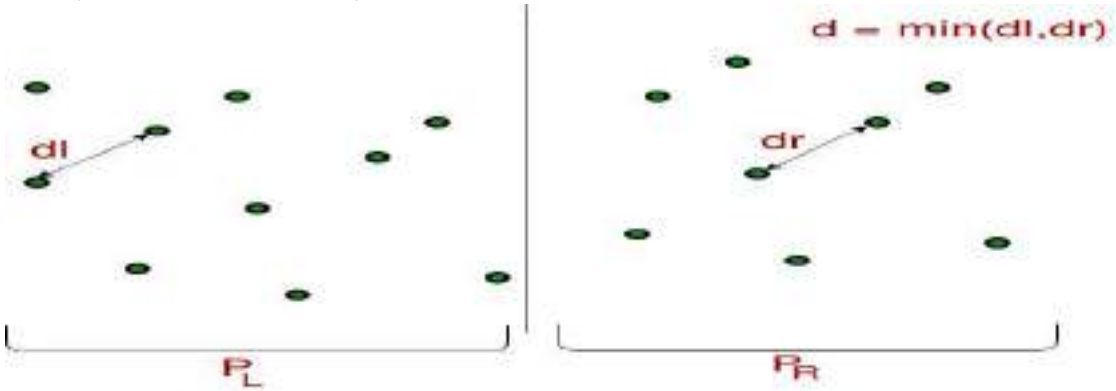


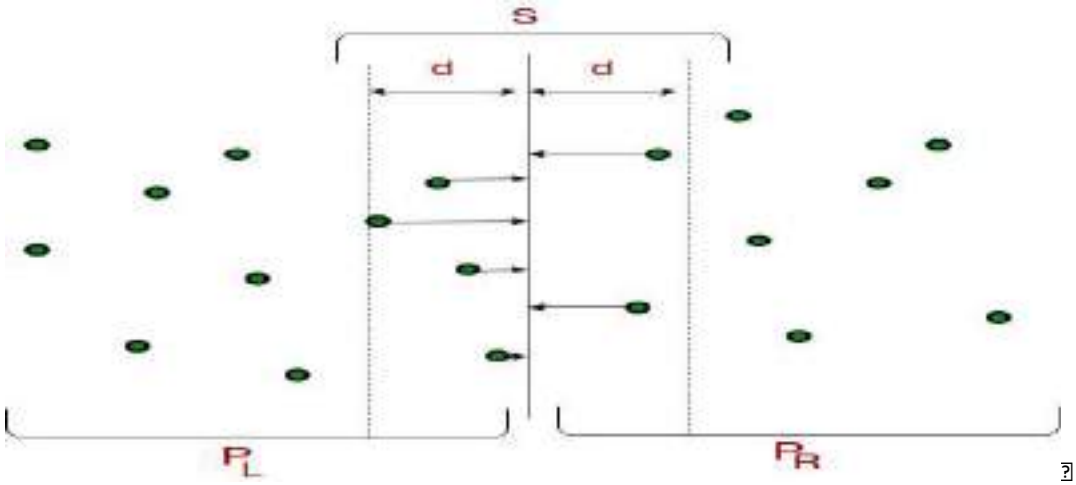
Sweep Lines

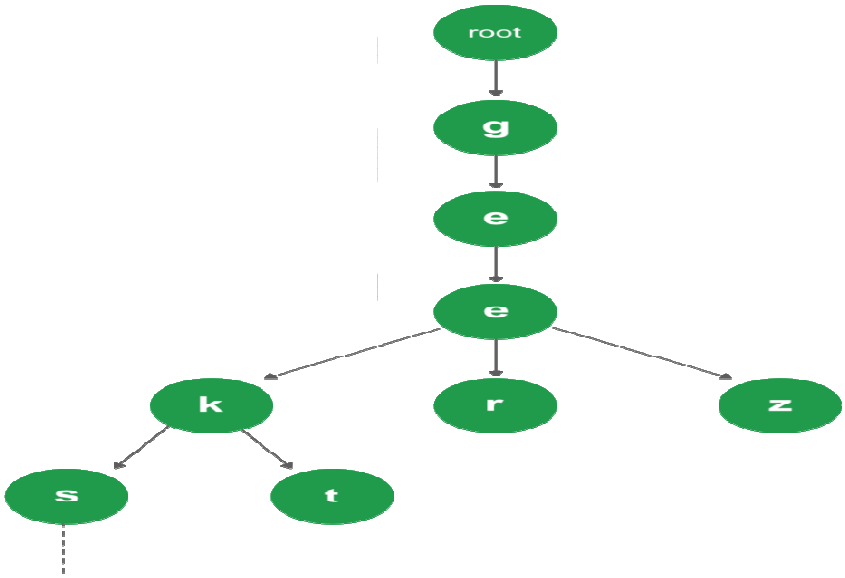

Upper Tangent
P
Convex Hull
Lower Tangent


Upper Tangent
P
Convex Hull
Lower Tangent
Point on the convex hull
and the movement to find the
upper tangent

$$\|pq\| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$






root
g
e
e
k
r
z
s
t

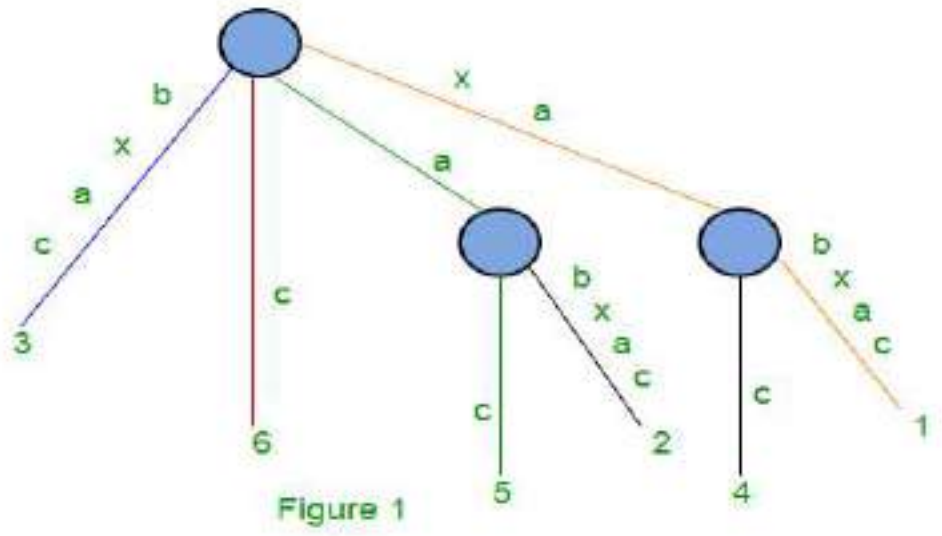


Figure 1

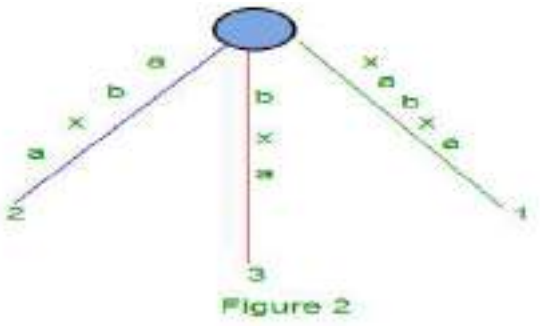


Figure 2

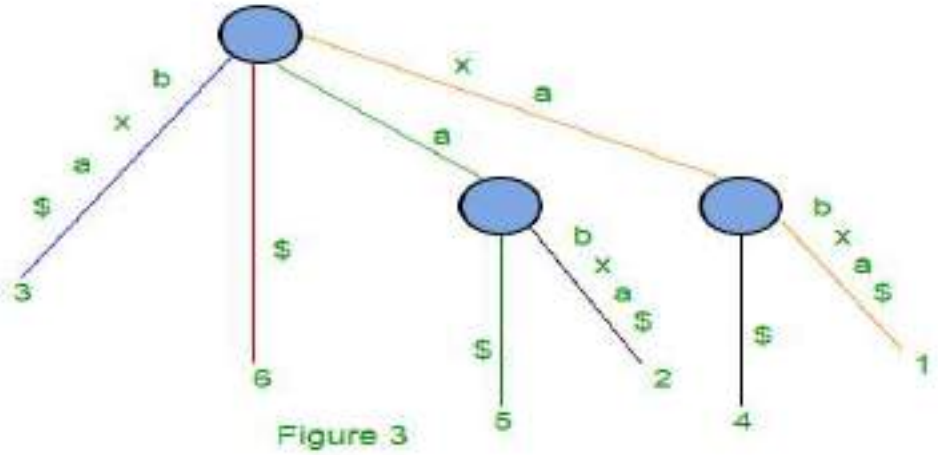
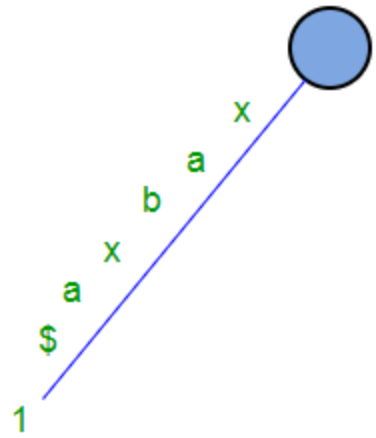
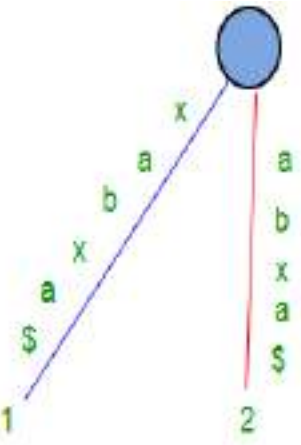


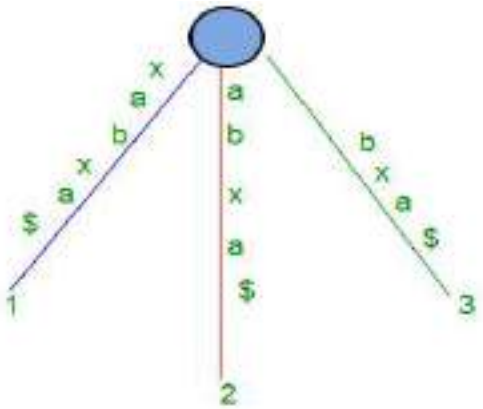
Figure 3



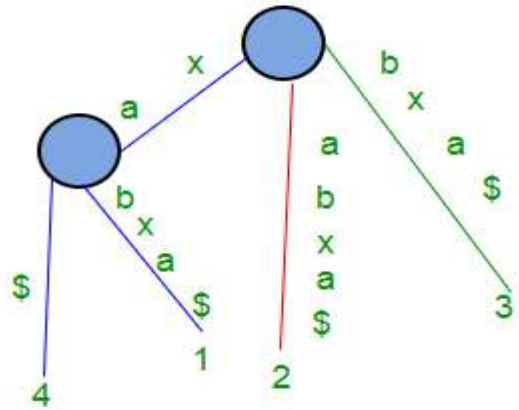
Tree with suffix N1, S[1....6]
Figure 4



Tree with suffix N1, S[1....6] and N2, S[2...6]
Figure 5



Tree with suffixes N1,N2 and N3
Figure 6



Tree with suffix N1, N2, N3 and N4
Figure 7

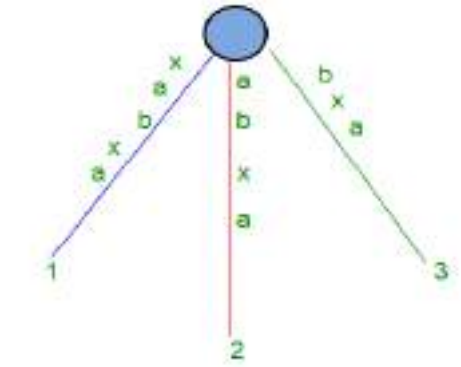


Figure 8 : Implicit suffix tree for storing xabxa
Suffix tree shown in Figure 3

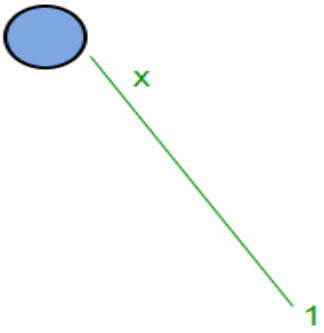


Figure 9 : T1 for S[1...1]
Adding suffixes of x(x)
Rule2-A new leaf node

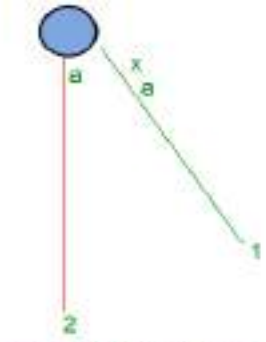


Figure 10 : T2 for S[1...2]
Adding suffixes of xa(xa and a)
Rule1-Extending path label in existing leaf edge
Rule2-A new leaf node

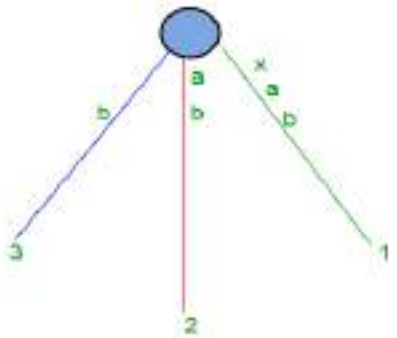


Figure 11 : T3 for S[1...3]
Adding suffixes of xab(xab ab and b)
Rule1-Extending path label in existing leaf edge
Rule2-A new leaf node

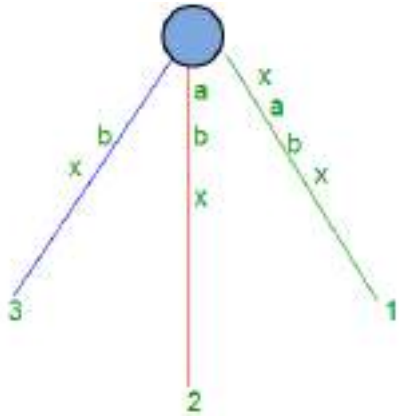


Figure 12 : T4 for S[1...4]
Adding suffixes of xabx(xabx,abx,bx and x)
Rule1-Extending path label in existing leaf edge
Rule3-Do nothing(path with label x already present)

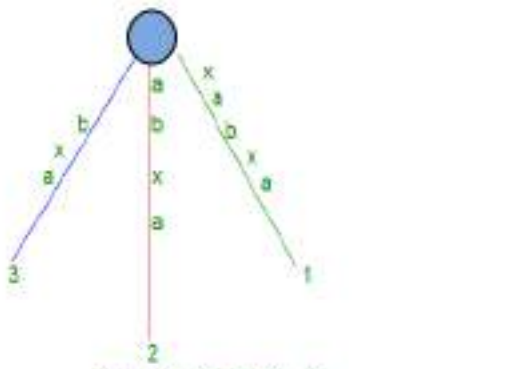


Figure 13 : T5 for S[1...6]
Adding suffixes of xabxa(xabxa,abxa,bxa,xa and x)
Rule1-Extending path label in existing leaf edge
Rule3-Do nothing(path with label xa and a already present)

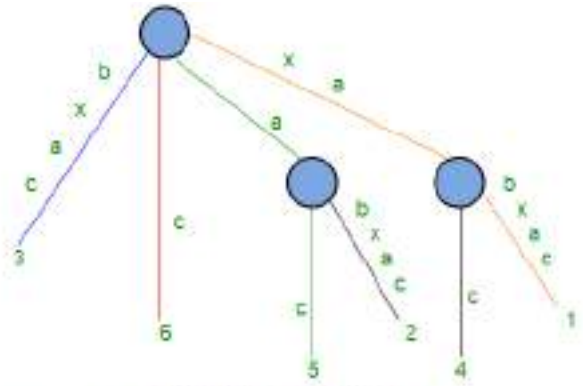


Figure 14 : T6 for S[1...6]
Adding suffixes of xabxac(xabxac, abxac, bxac, xac, ac, c)
Rule1-Extending path label in Existing leaf edge.
Rule2-Three new leaf edges and two new internal nodes

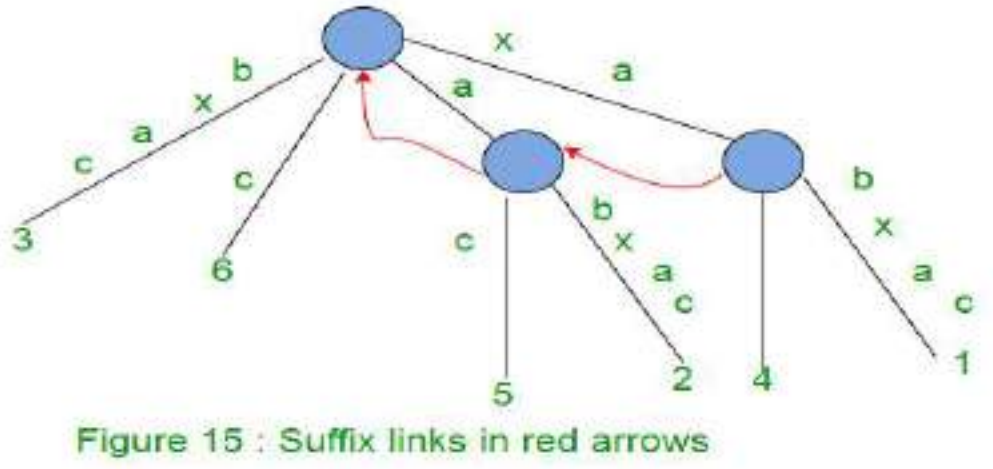


Figure 15 : Suffix links in red arrows

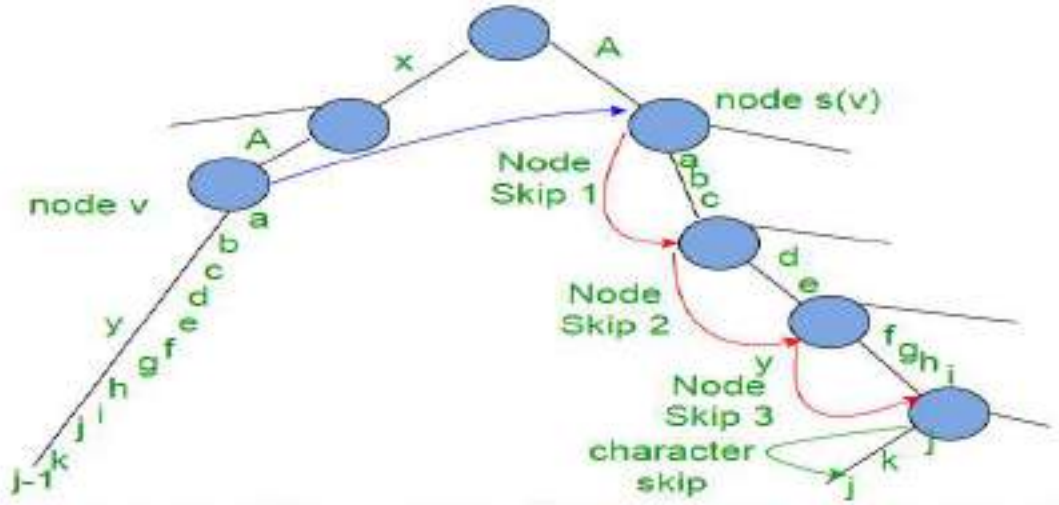


Figure 18 : skip/count trick : substring y from node v has length 11. Substring y from node s(v) is two characters down the last node, after 3 node skips

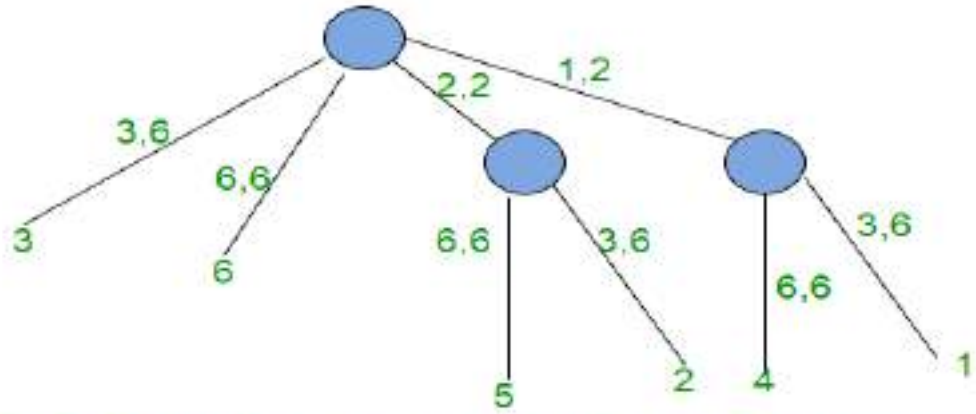


Figure 19 : Suffix tree for string xabxac with edge-label compression
Figure 14 shows same suffix tree without edge-label compression

+'6>' "" ' "6- 6< -' " 9' 6 /

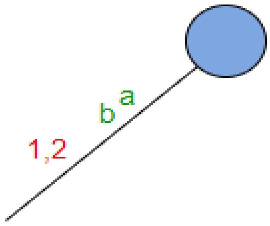


Figure 21 : Phase 2, Extension1-Rule1 applied extended the leaf edge from (1,1) to (1,2)

- +'<--j k' " /" ' - 0 " * ' ' -' " 9' < /

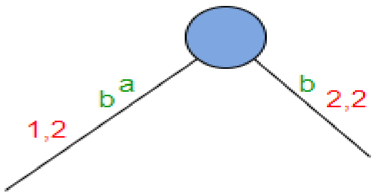


Figure 22 : Phase 2, Extension2-Rule2 applied
Created a leaf edge(2,2)
Phase 2 completes here

' < ' ' 0 " -' ' < /
' < 0 " 0 ' ' " ' /9' 6 ' 6+' 9' < ' <
+' /

- ' B0 ' ' 0 ' 6 B' ' /
(' ' ' B0 ' ' */-- " " ' j k j k j k/
+' 6--j k " /
" * ' -' " * " ' -' " /
+' 6>' "" ' "6- <B ' " 9' 6 /

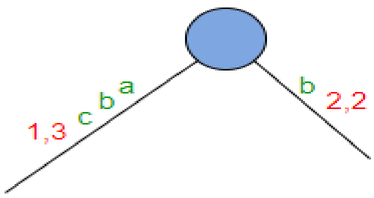


Figure 23 : Phase 3, Extension1-Rule1 applied
Extended the leaf edge from (1,2) to (1,3)

- +' <--j k' " /
" * ' -' " * " ' -' " /
+' <>' "" ' "6- <B ' " 9' 6 /

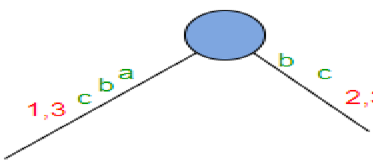


Figure 24 : Phase 3, Extension2-Rule1 applied
Extended the leaf edge from (2,2) to (2,3)

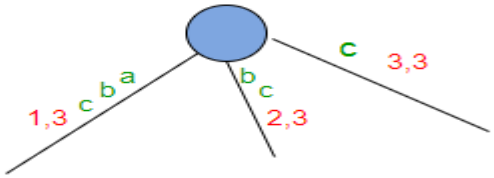


Figure 25 : Phase 3, Extension3-Rule2 applied
Created a leaf edge(3,3)
Phase 3 completes here

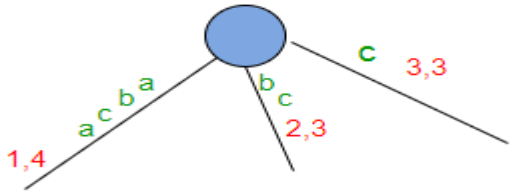
-


Figure 26 : Phase 4, Extension 1
Rule 1 applied

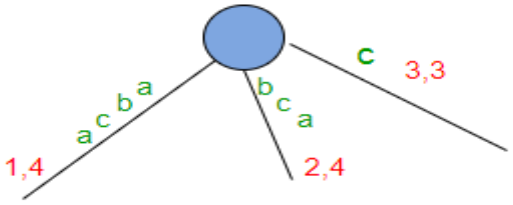


Figure 27 : Phase 4, Extension 2
Rule 1 applied

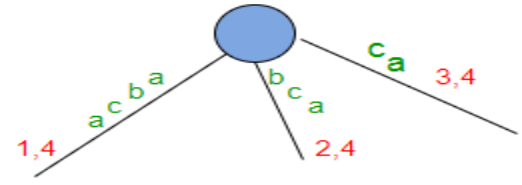
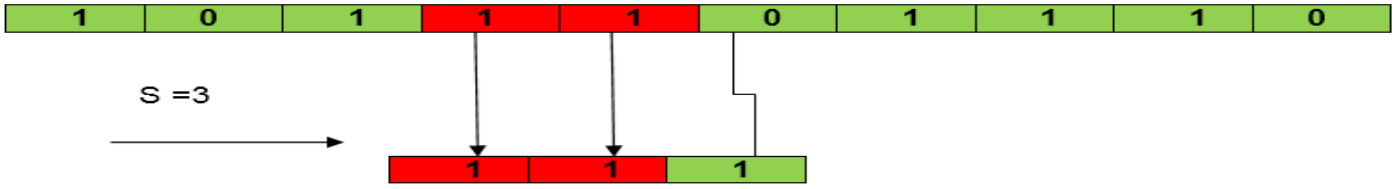
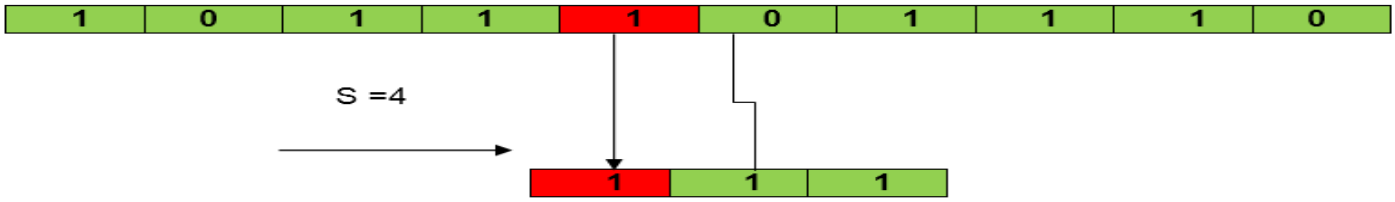
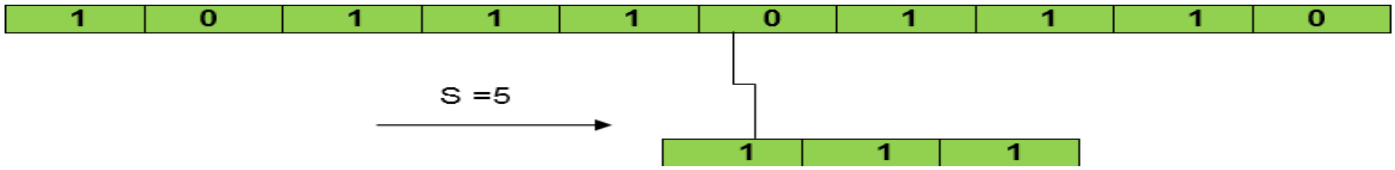
-


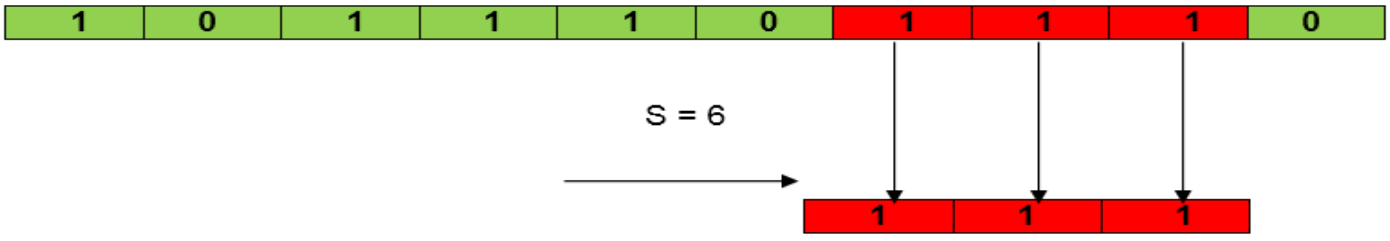
Figure 28 : Phase 4, Extension 3
Rule 1 applied

So, S=2 is a Valid Shift

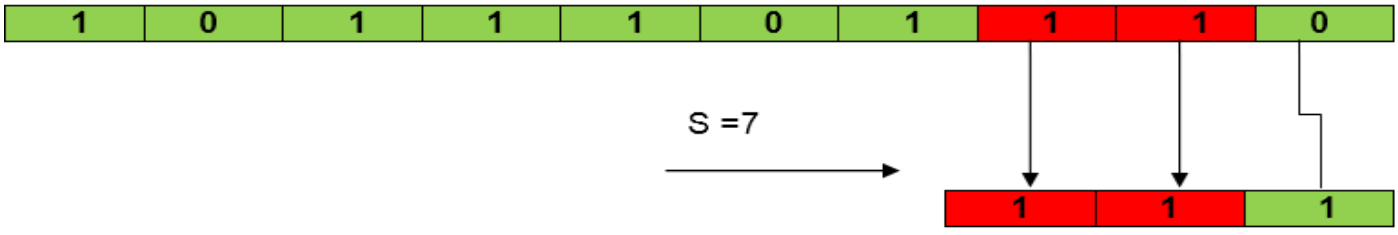








So, S=6 is a Valid Shift



31 mod 11 = 9 not equal to 4

14 mod 11 = 3 not equal to 4

41 mod 11 = 8 not equal to 4

15 mod 11 = 4 equal to 4 SPURIOUS HIT

59 mod 11 = 4 equal to 4 SPURIOUS HIT

92 mod 11 = 4 equal to 4 SPURIOUS HIT

26 mod 11 = 4 EXACT MATCH

65 mod 11 = 10 not equal to 4

53 mod 11 = 9 not equal to 4

35 mod 11 = 2 not equal to 4

The Pattern occurs with shift 6.

Step 1: q = 2, k = 0

Π [2] = 0

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0					

Step 2: q = 3, k = 0

Π [3] = 1

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0	1				

Step3: q =4, k =1

Π [4] = 2

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	A
π	0	0	1	2			

Step4: q = 5, k =2

Π [5] = 3

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0	1	2	3		

Step5: q = 6, k = 3

Π [6] = 0

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0	1	2	3	0	

Step6: q = 7, k = 1

Π [7] = 1

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0	1	2	3	0	1

q	1	2	3	4	5	6	7
p	a	b	A	b	a	c	a
π	0	0	1	2	3	0	1

T:

b	a	c	b	a	b	a	b	a	b	a	c	a	c	a

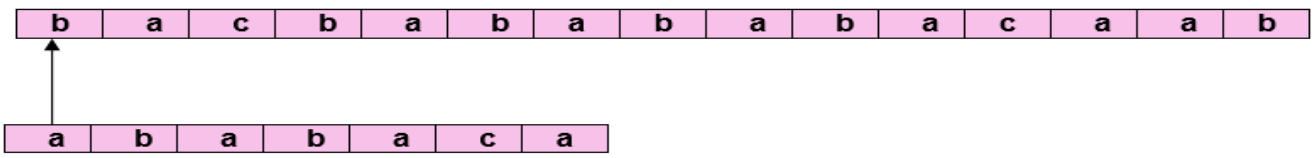
P:

a	b	a	b	a	c	a

q	1	2	3	4	5	6	7
p	a	b	A	b	a	c	a
π	0	0	1	2	3	0	1

Step1: i=1, q=0

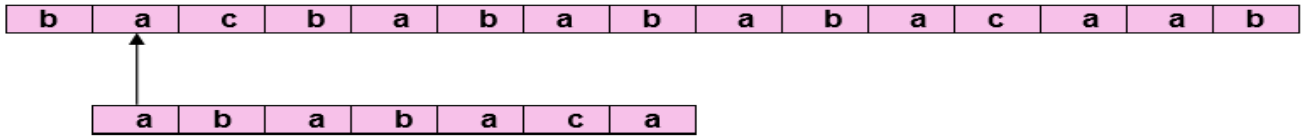
Comparing P [1] with T [1]



P [1] does not match with T [1]. 'p' will be shifted one position to the right.

Step2: i = 2, q = 0

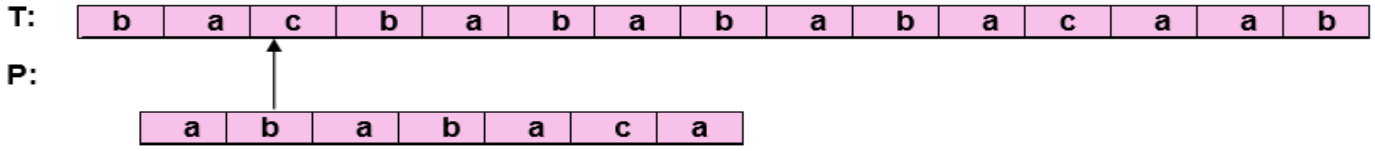
Comparing P [1] with T [2]



P [1] matches T [2]. Since there is a match, p is not shifted.

Step 3: i = 3, q = 1

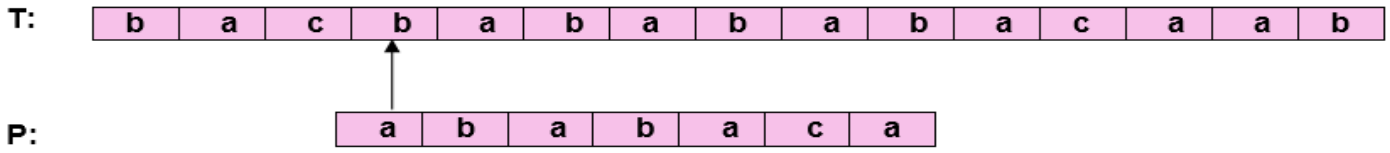
Comparing P [2] with T [3] P [2] doesn't match with T [3]



Backtracking on p, Comparing P [1] and T [3]

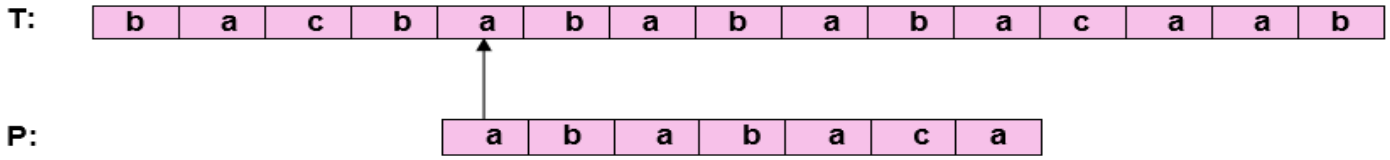
Step4: i = 4, q = 0

Comparing P [1] with T [4] P [1] doesn't match with T [4]



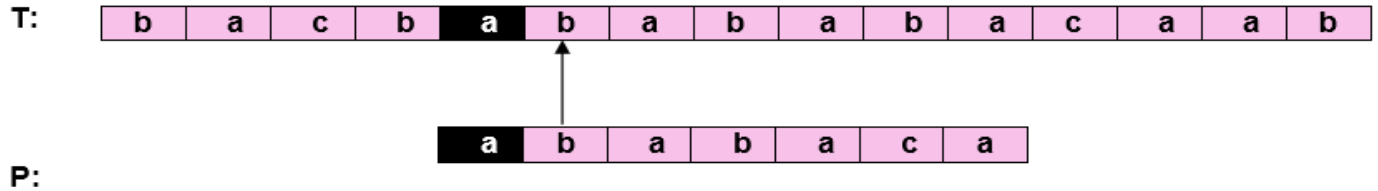
Step5: i = 5, q = 0

Comparing P [1] with T [5] P [1] match with T [5]



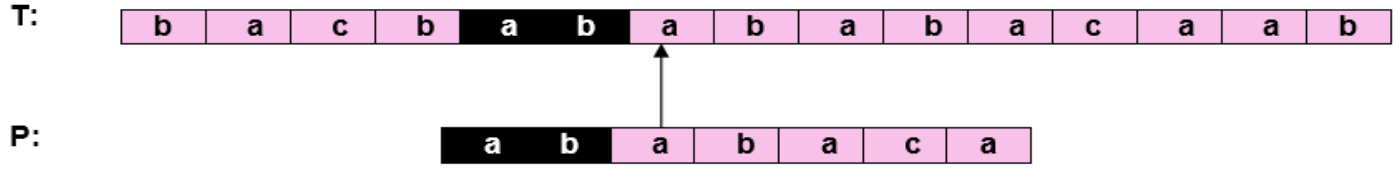
Step6: i = 6, q = 1

Comparing P [2] with T [6] P [2] matches with T [6]



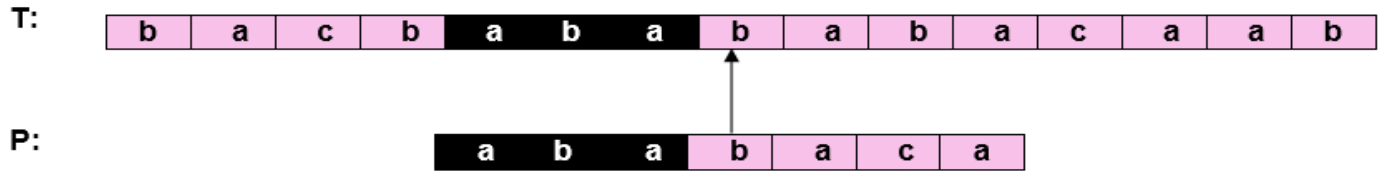
Step7: i = 7, q = 2

Comparing P [3] with T [7] P [3] matches with T [7]



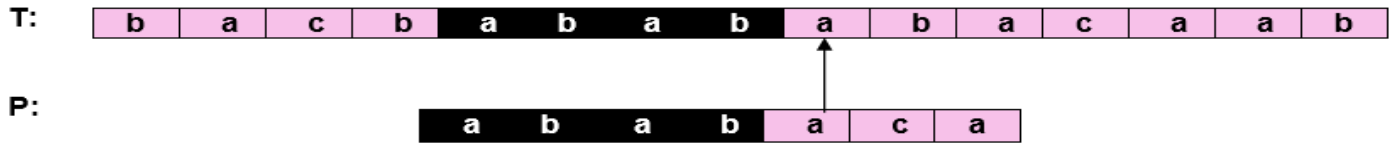
Step8: i = 8, q =3

Comparing P [4] with T [8] P [4] matches with T [8]



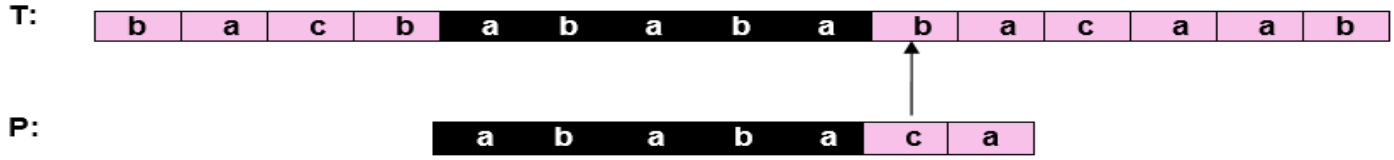
Step9: i = 9, q = 4

Comparing P [5] with T [9] P [5] matches with T [9]



Step10: i = 10, q = 5

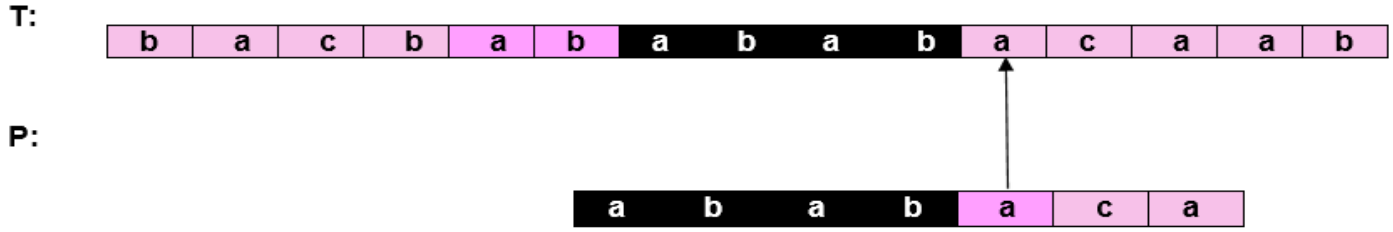
Comparing P [6] with T [10] P [6] doesn't match with T [10]



Backtracking on p, Comparing P [4] with T [10] because after mismatch q = π [5] = 3

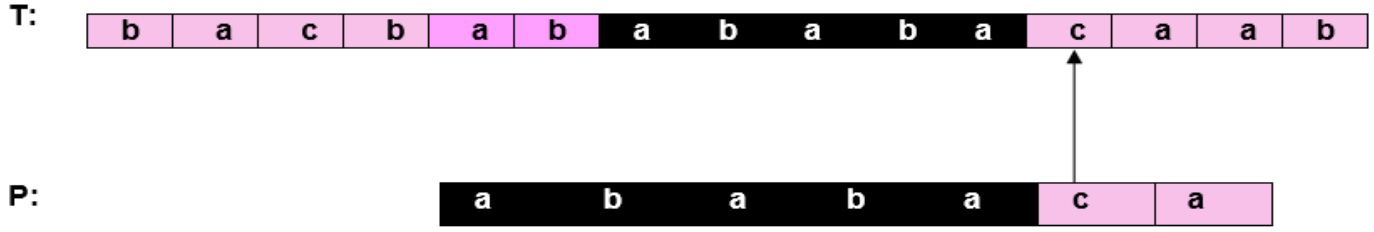
Step11: i = 11, q =4

Comparing P [5] with T [11] P [5] match with T [11]



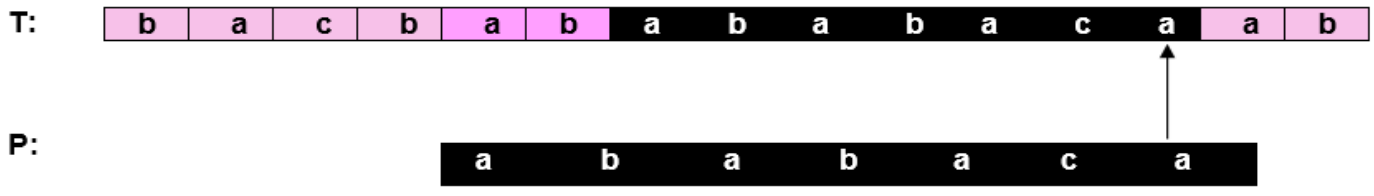
Step12: i = 12, q = 5

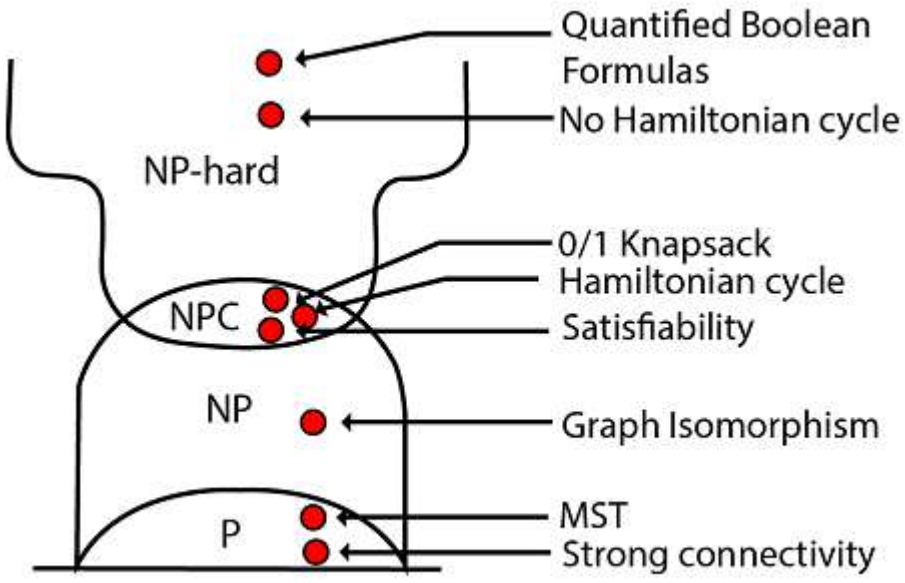
Comparing P [6] with T [12] P [6] matches with T [12]

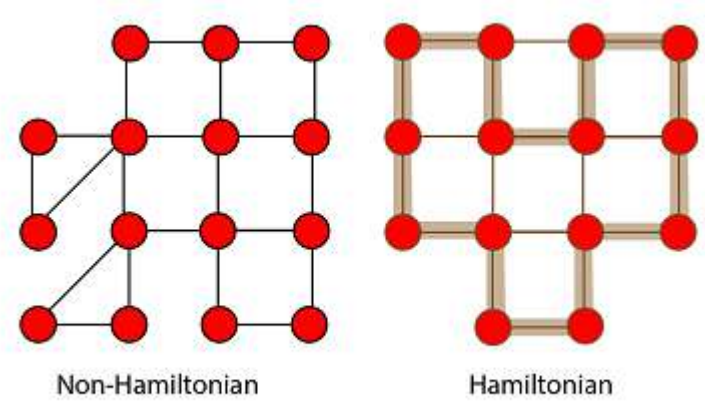


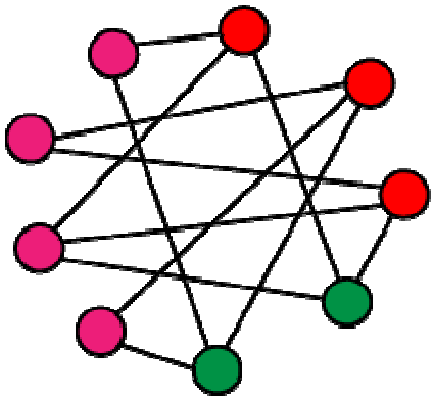
Step13: i = 3, q = 6

Comparing P [7] with T [13] P [7] matches with T [13]

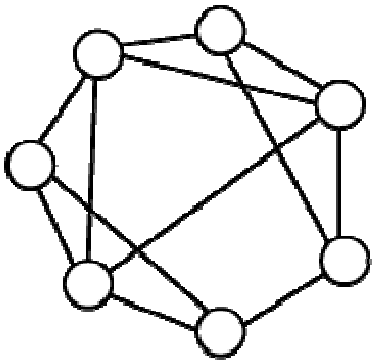



Quantified Boolean
Formulas
No Hamiltonian cycle
NP-hard
0/1 Knapsack
Hamiltonian cycle
Satisfiability
NPC
NP
Graph Isomorphism
MST
P
Strong connectivity


Non-Hamiltonian
Hamiltonian



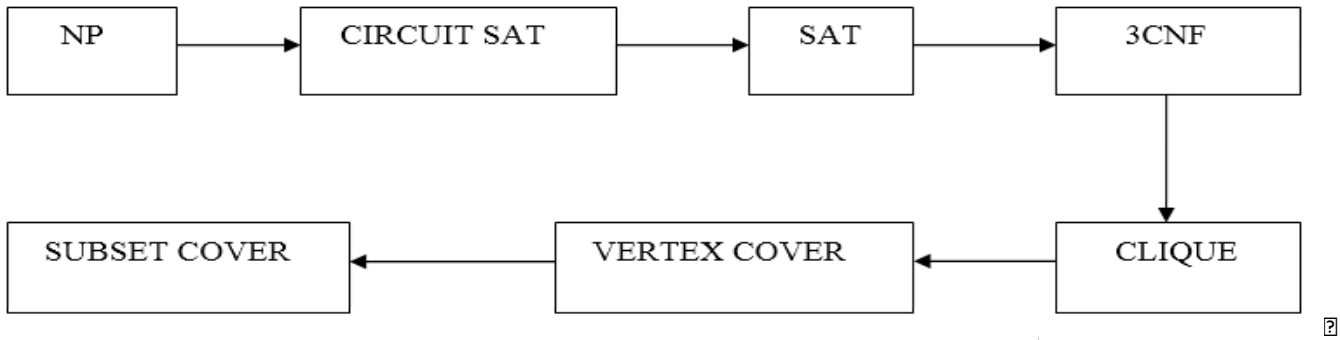
3-Colorable

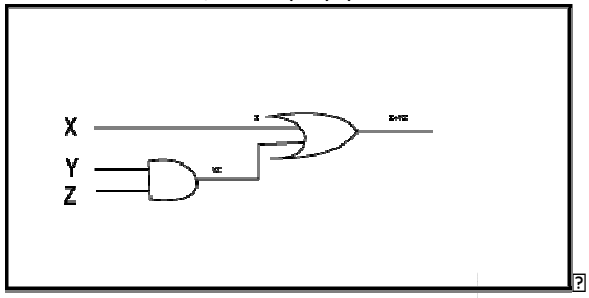


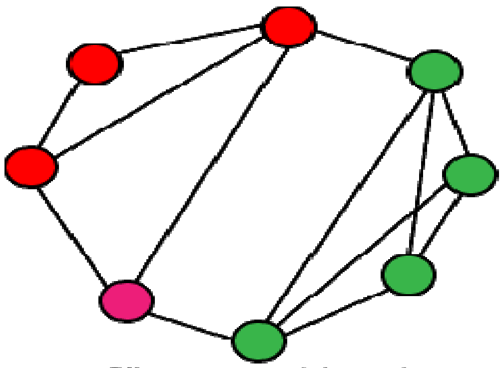
Not 3-Colorable

</ [illegible]

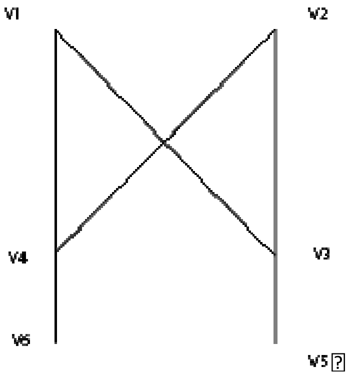
So according to the given decision-based NP problem, you can decide in the form of yes or no. If, yes then you have to do verify and convert into another problem via reduction concept. If you are being performed, both then decision-based NP problems are in NP compete.



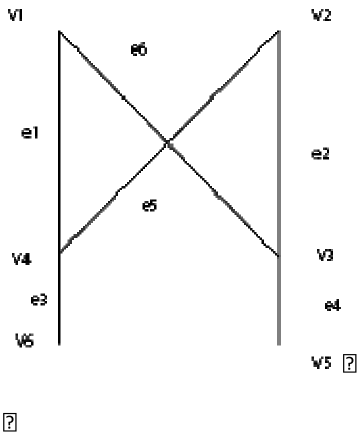




Clique cover (size=3)


V1
V2
V4
V3
V6
V5

[illegible]


V1
V2
e6
e1
e2
e5
V4
V3
e3
e4
V6
V5