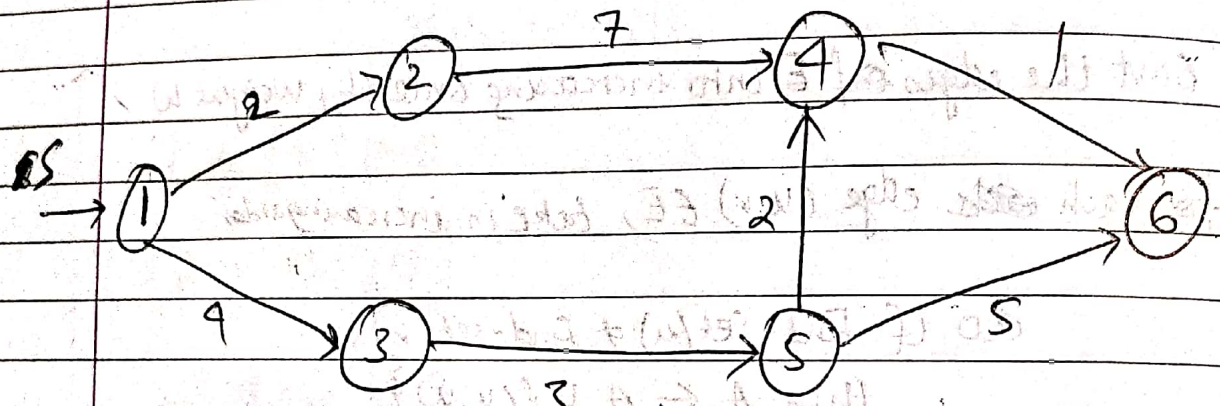
Single Source shortest path problem,

1) Dijkstra Algorithm - Works on directed and non-directed graphs together



Relation

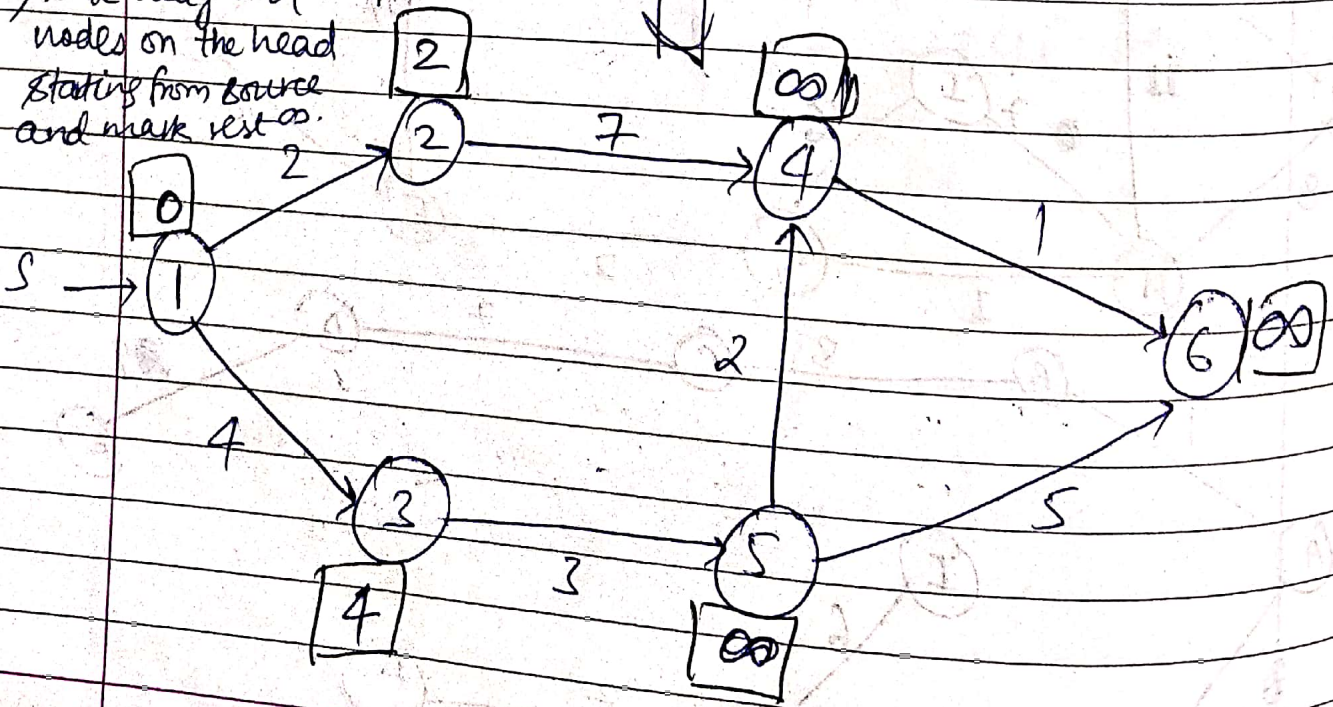
weight

if (dist[u] + C[u,v] < d[v])

d[v] = d[u] + C[u,v]

u, v are adjacent nodes or concurrent nodes

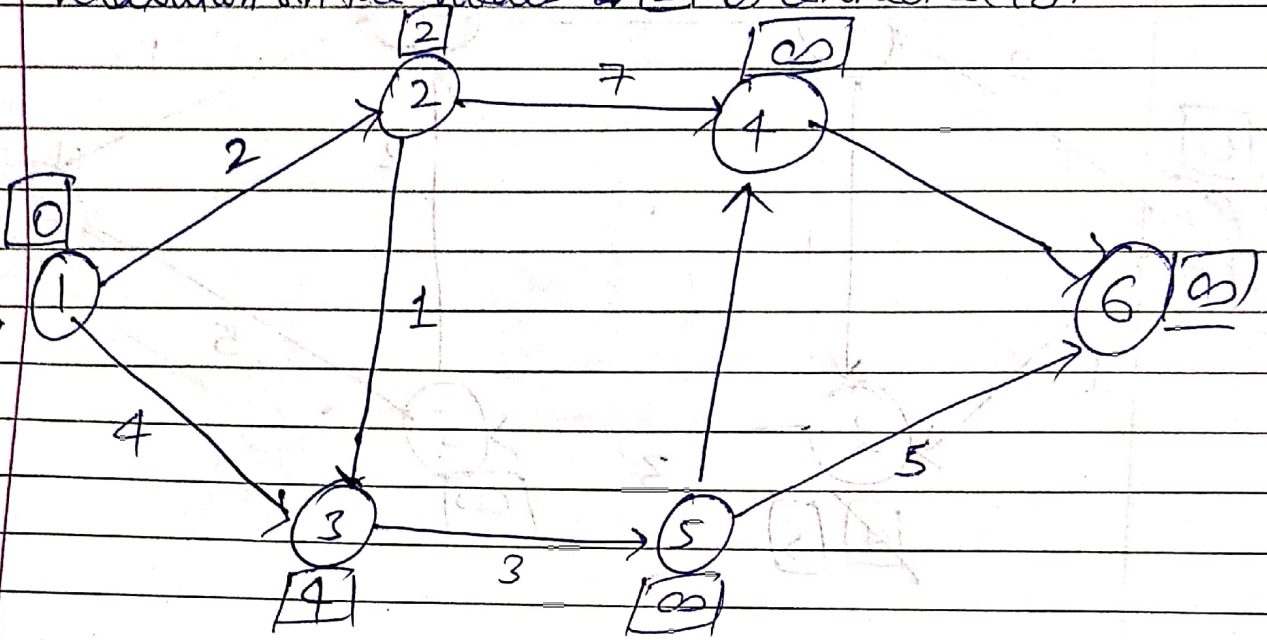
a) Write weight of respective nodes on the head starting from source and mark rest ∞.



DELTA | Page No. | Date / /

b) Select shortest path out of [2], [4], [∞], [∞], [∞]

After selecting the shortest path, i.e. [2] here, perform relaxation on the nodes [2] is connected to.



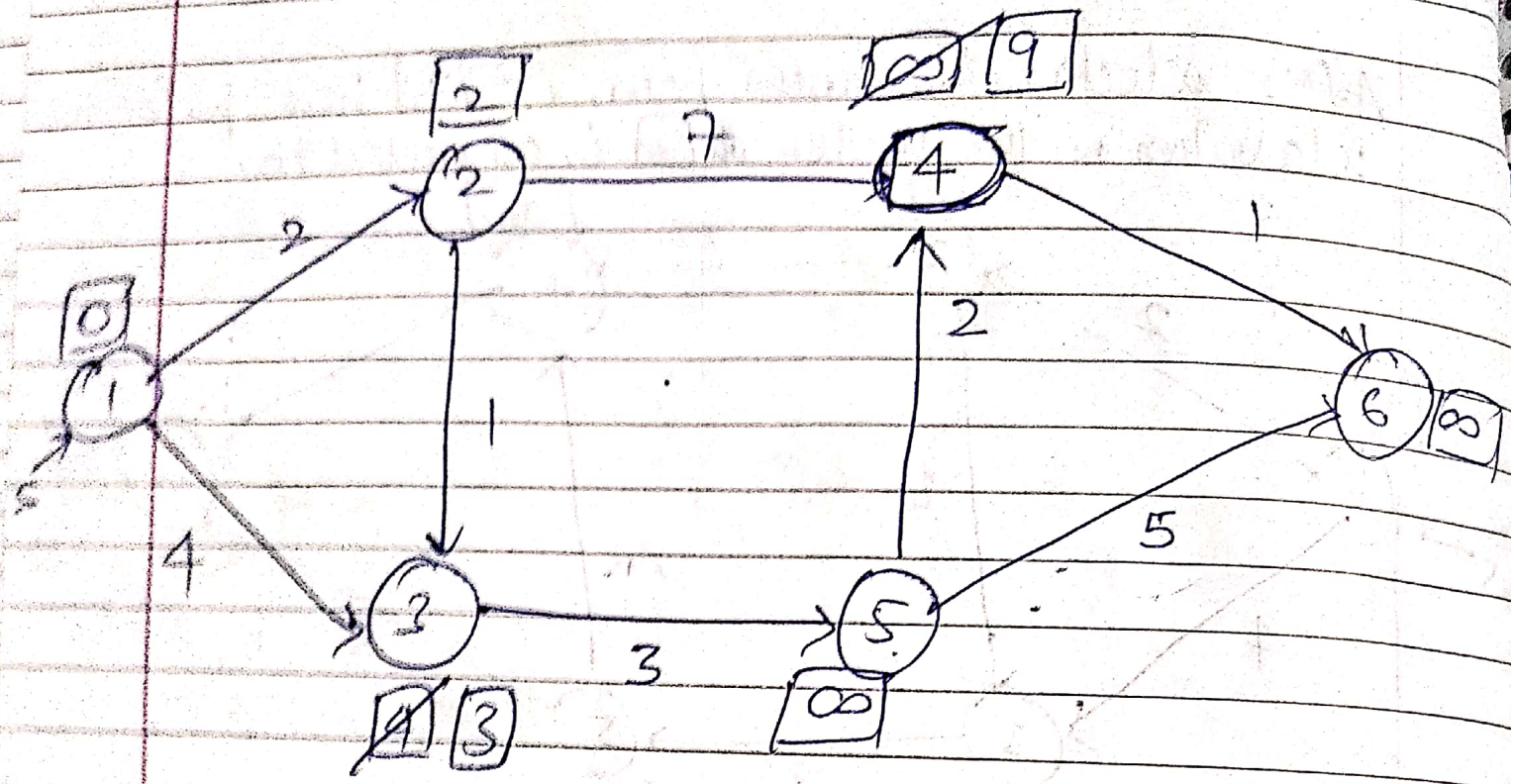
Perform relaxation on the nodes (2) is connected to and change the head weight in boxes [ ] accordingly.

Eg path from (2) to (3)

hence [2] + 1 ⇒ 3 so (3) will have [3] as head weight but not [4] ^ 3 < 4 and

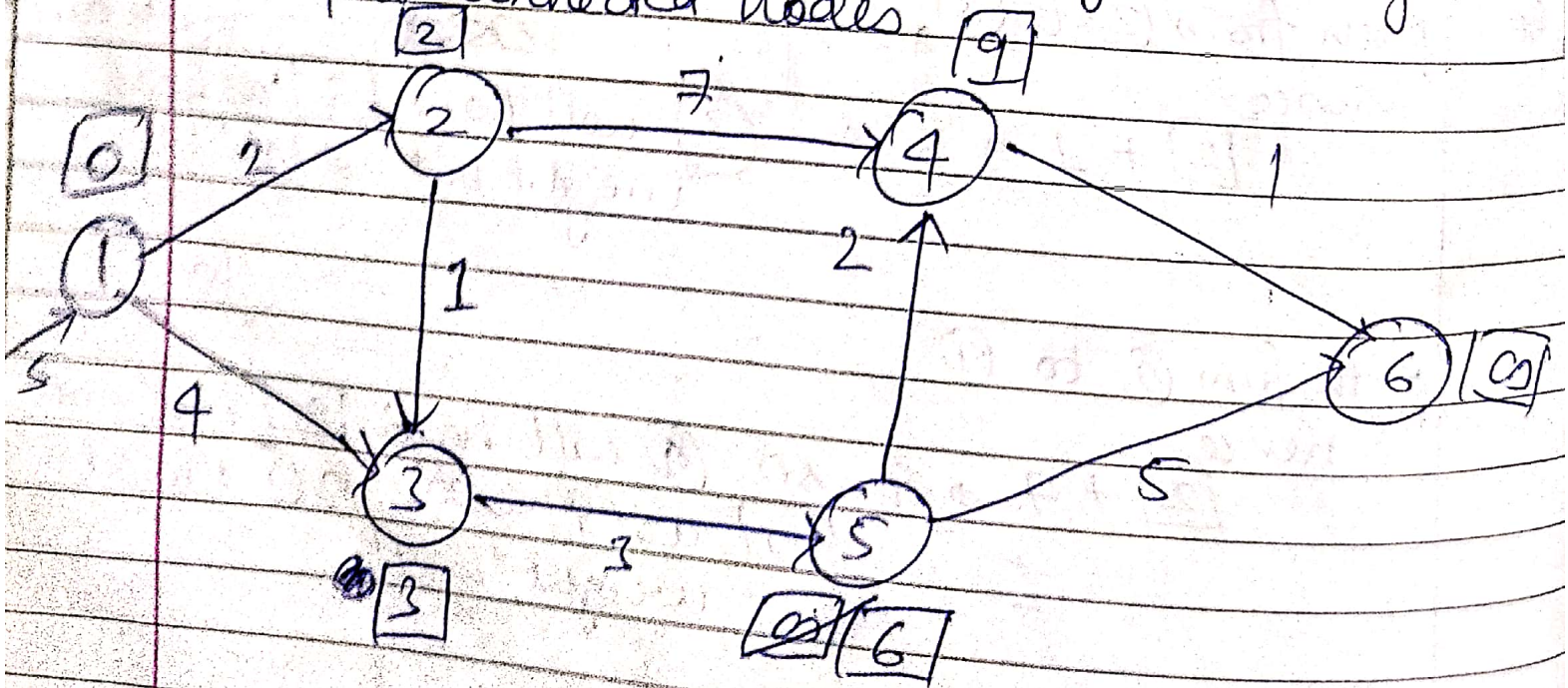
path from (2) to (4)

hence [2] + 7 ⇒ 9 so (4) will have [9] but not [∞] as [9] < [∞] so (4)'s head weight is [9].

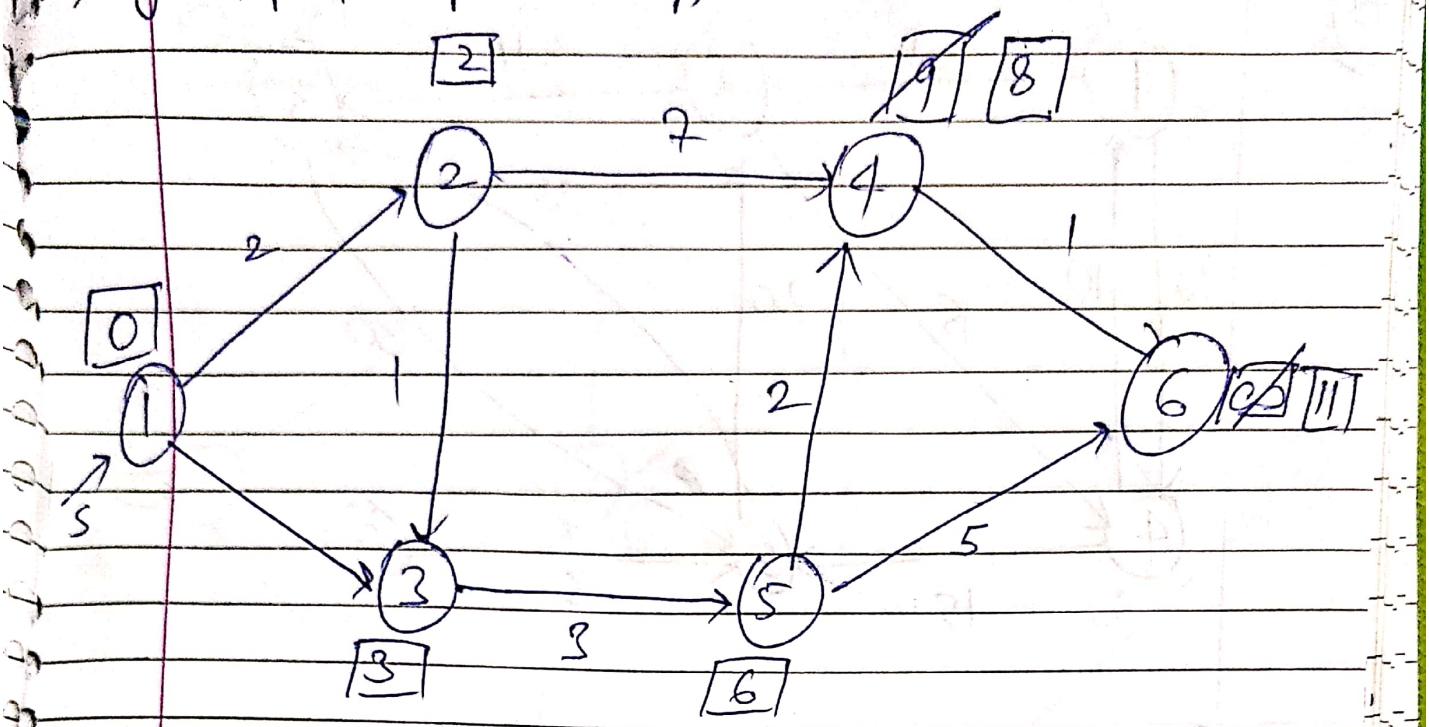


c) Perform the same step as and check the smallest head weight of the nodes i.e. [3] [9] [∞] [∞]

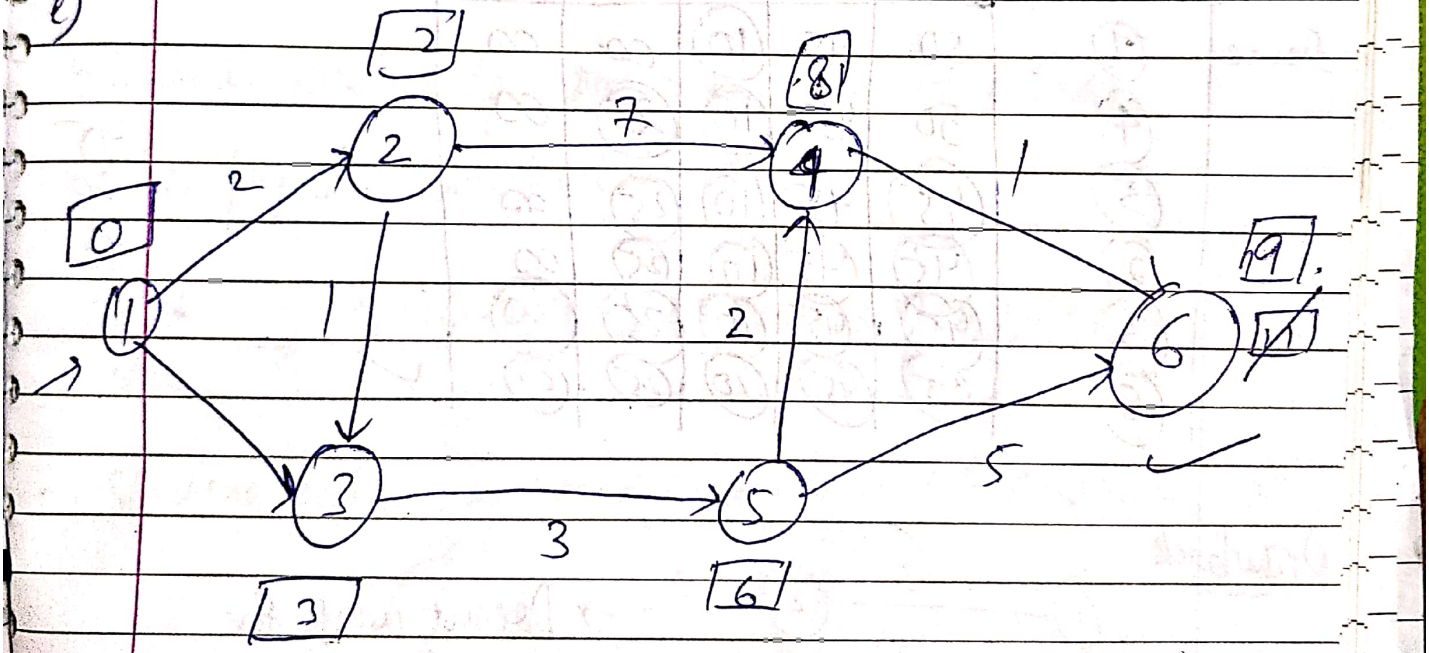
Here its [3] so check and change headweights of its connected nodes



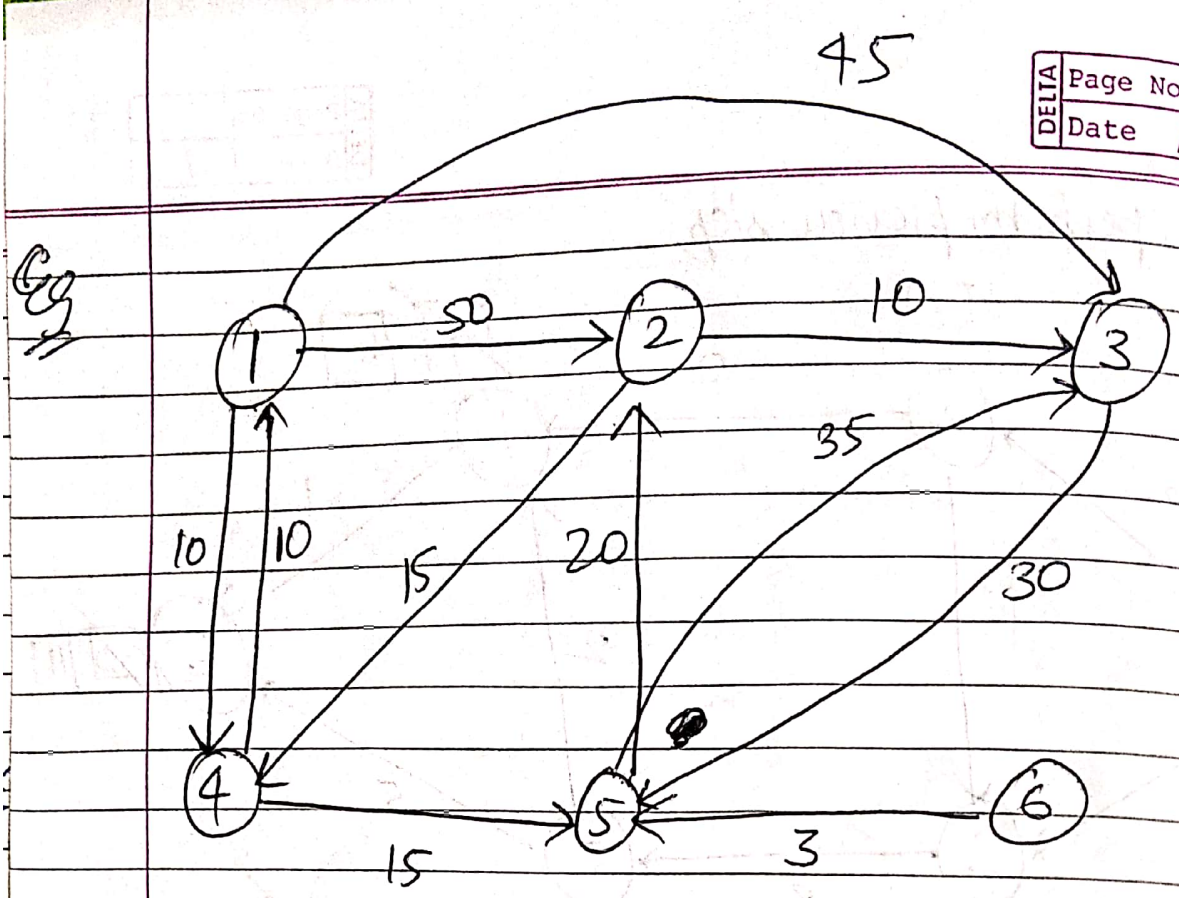
d) Again, perform previous step,



e)

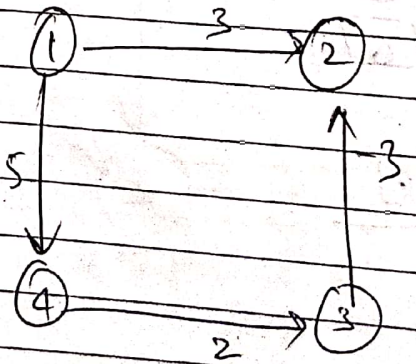


V	distance[v]
1	0
2	2
3	3
4	8
5	6
6	9



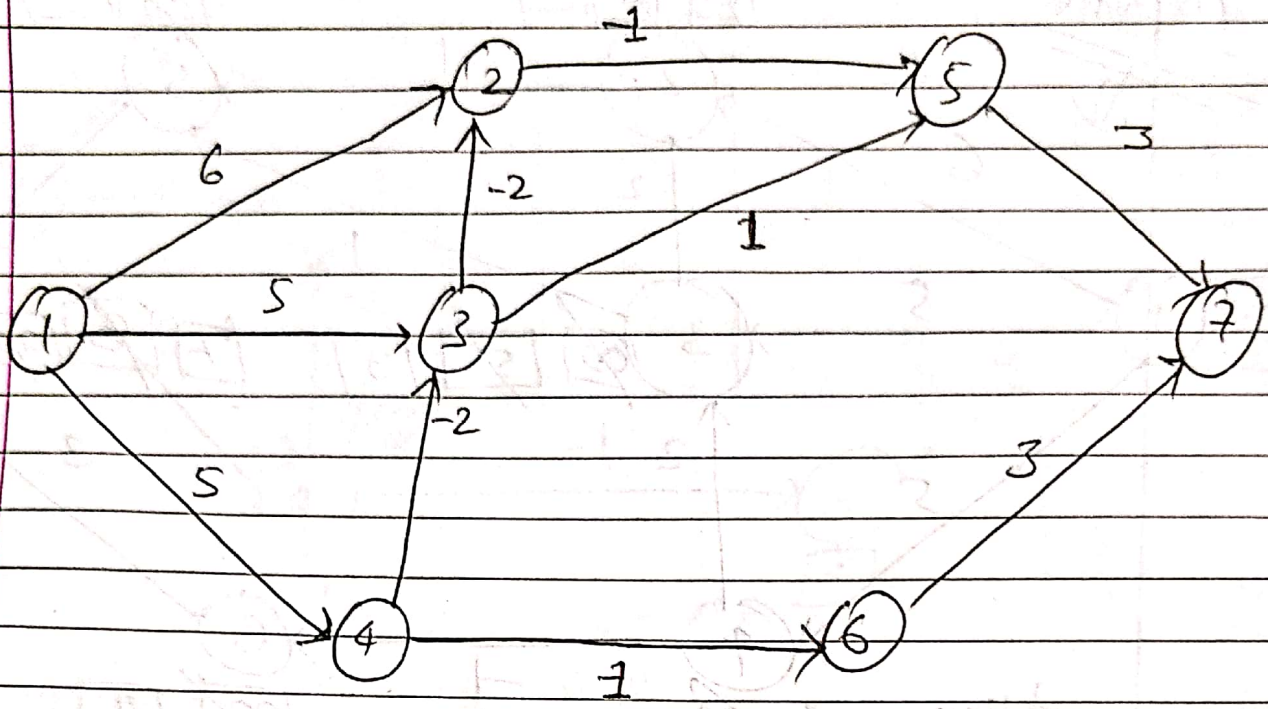
Selected Vertex	2	3	4	5	6
Source (1)	50	45	(10)	∞	∞
(4)	50	45	(10)	(25)	∞
(5)	(45)	45	(10)	(25)	∞
(2)	(45)	(45)	(10)	(25)	∞
(3)	(45)	(45)	(10)	(25)	(∞)
(6)	(45)	(45)	(10)	(25)	(∞)

Drawback



→ Doesn't work for ve edges and fails.
→ May work or may not

BELLMAN FORD ALGORITHM

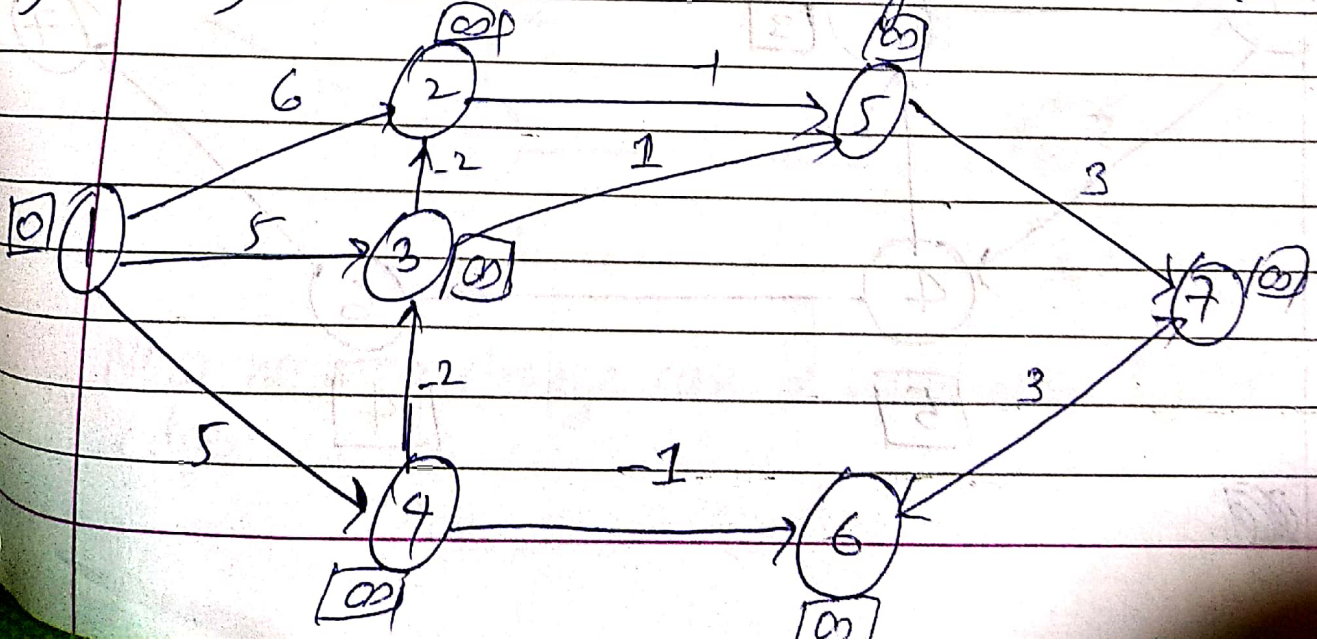


a) Repeatedly relax all edges n-1 times that is V-1 ⇒ 6 here.

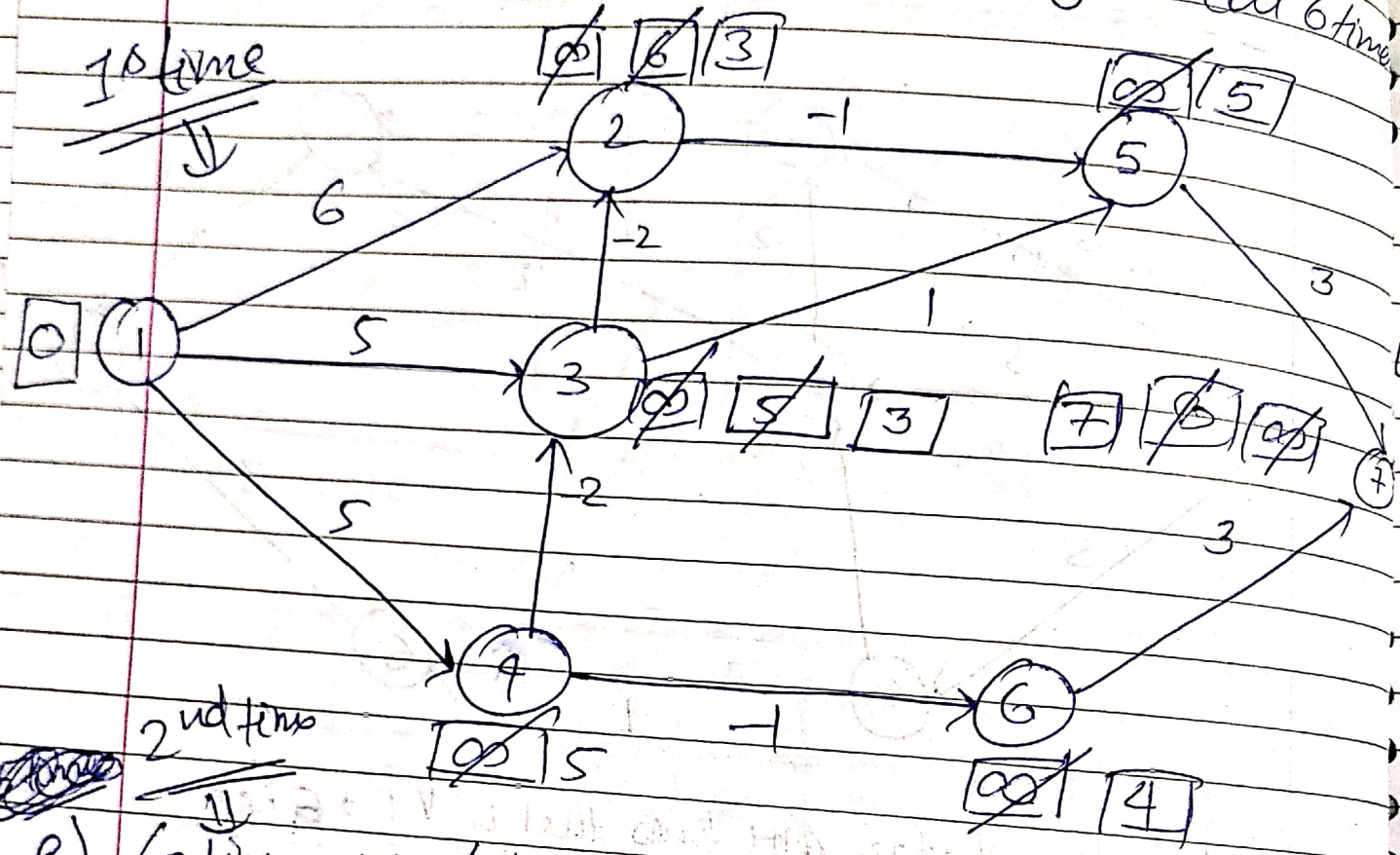
b) Then prepare a list of all the edges,

List of edges → (1,2)(1,3)(1,4)(2,5)(3,2)(3,5)(4,3)(4,6)(5,7)(6,7)

c) Starting vertex has naked head weigh ⇒ 0 and other ∞

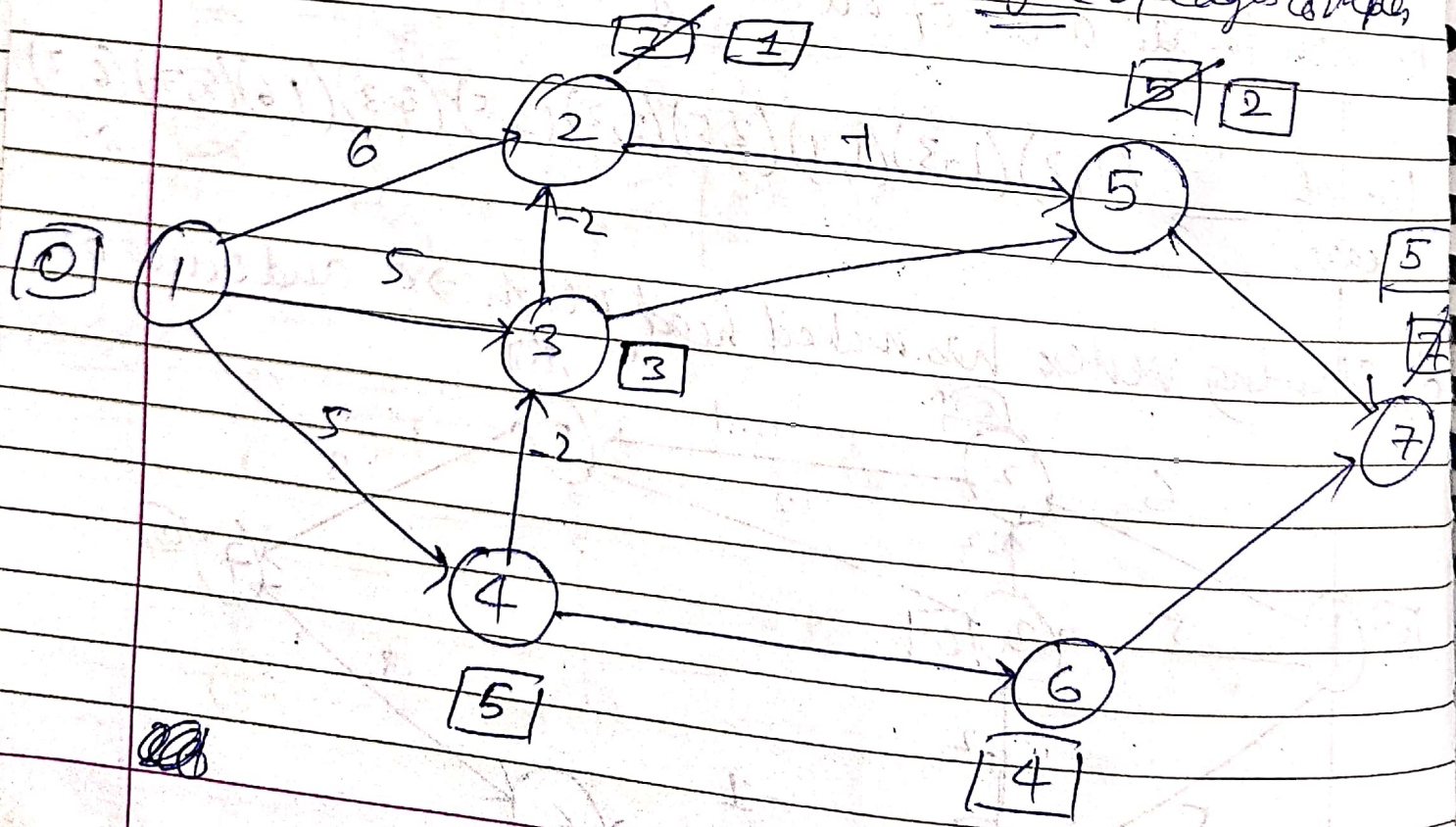


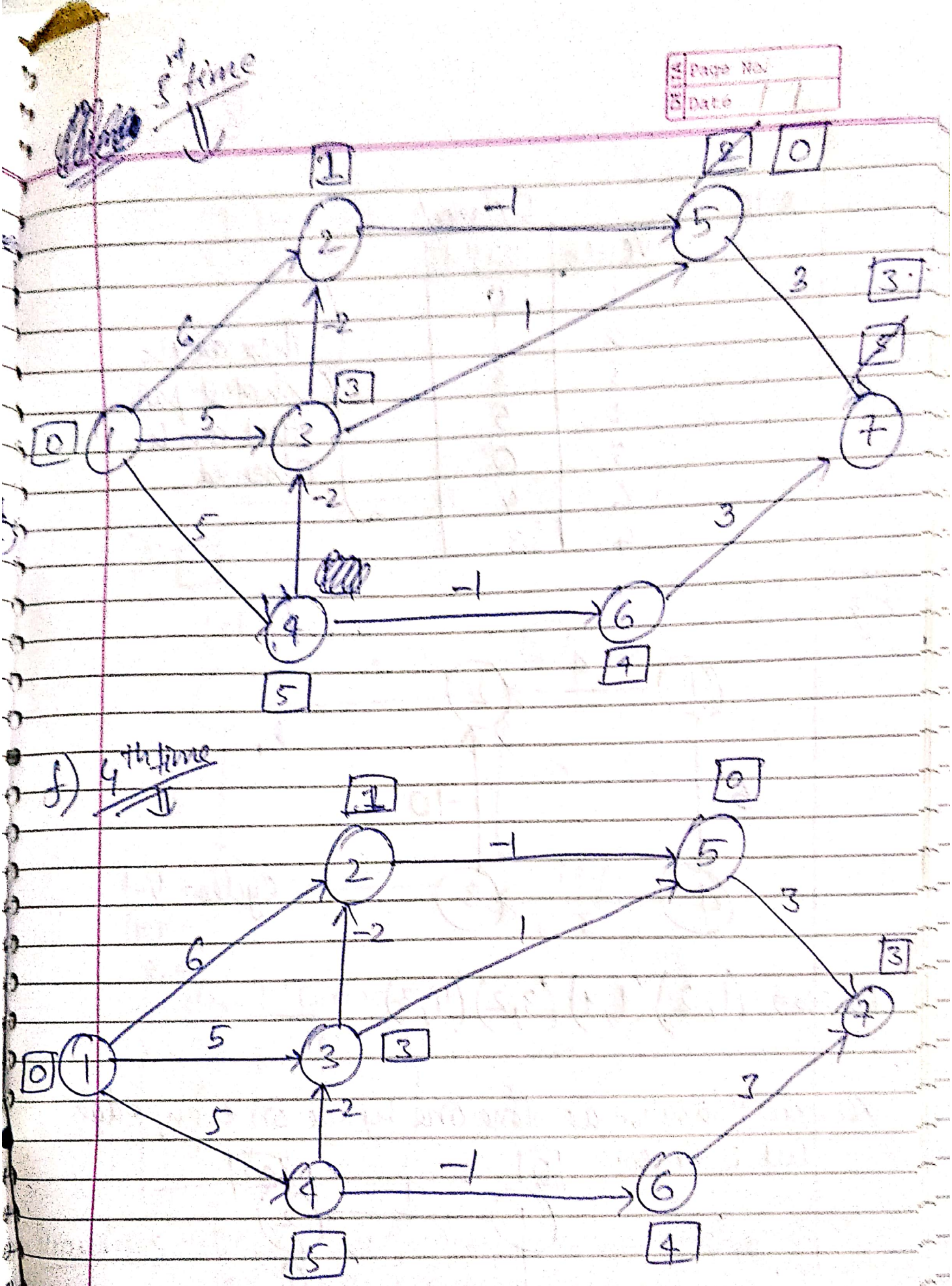
d) Now starting relaxing the edges one by one till 6 times



2nd time

e) Continue, second time again after one cycle of edges complete





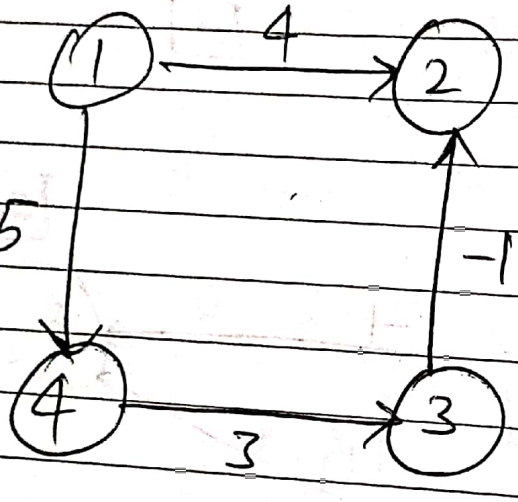
Now, no more changes can be incorporated by the edges.

hence

Vertex	Distance/ Weight
1	0
2	1
3	3
4	5
5	0
6	4
7	3

These are the shortest path that can be observed.

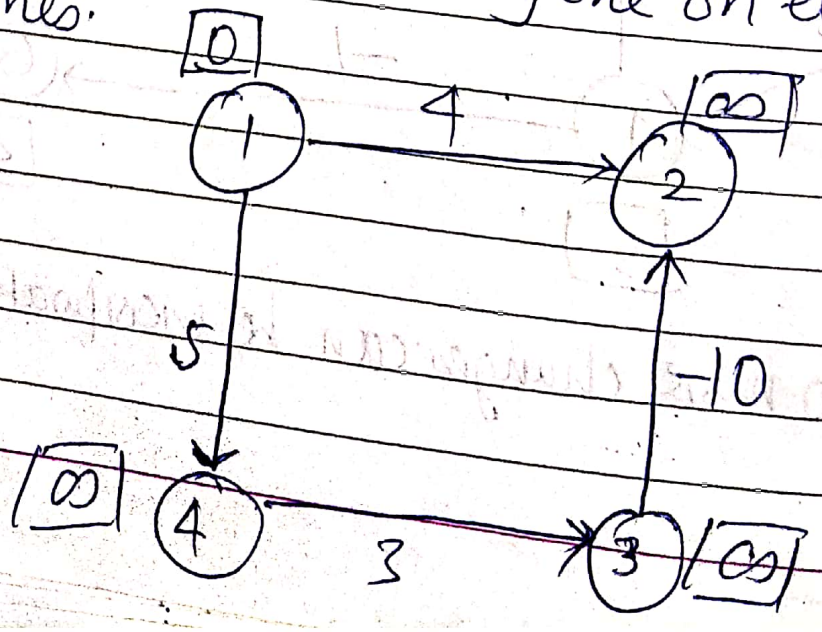
Eg



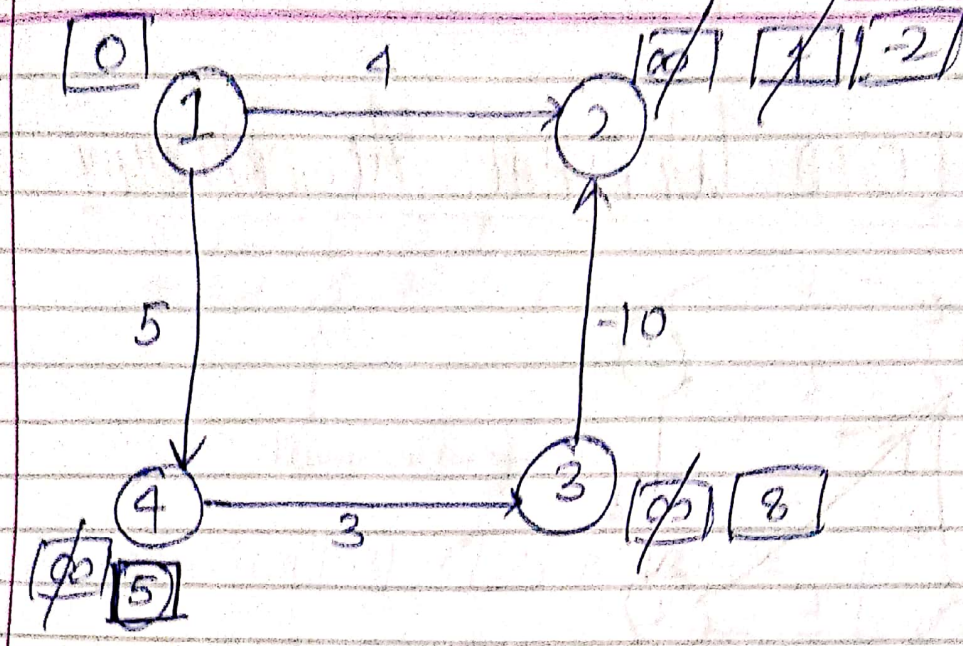
Cycles = V-1
= 4-1
= 3.

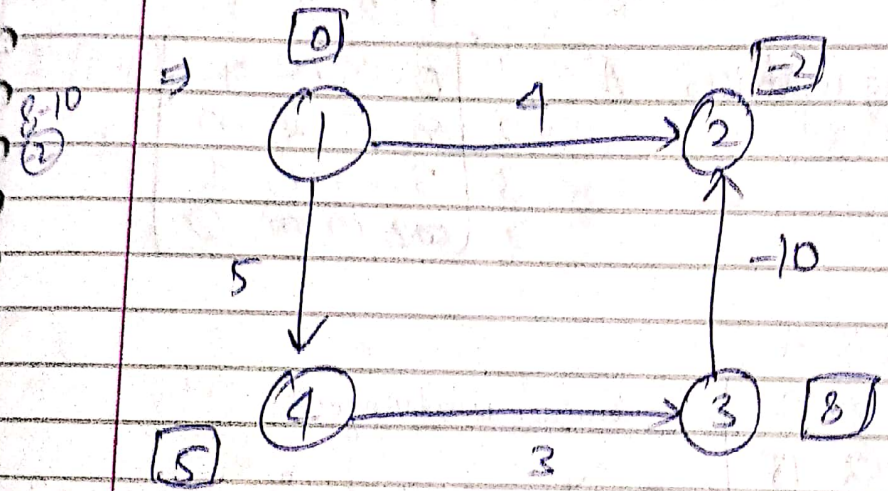
Edges ⇒ (1,2) (1,4) (3,2) (4,3)

So now relaxation are done one by one on every edge till 3 times.



(1,2)(1,4)(3,2)(4,3)





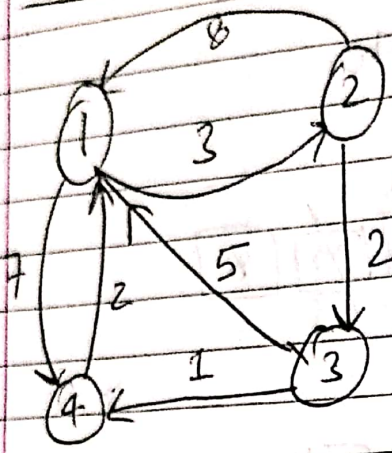
Vertex	Distance
1	0
2	-2
3	8
4	5

Drawback – Doesn't work when a cycle of edges with -ve weight are there as it keeps on relaxing even after (v-1) times and doesn't stop at (v-1) times.

⇒ Doesn't work with -ve weight cycle.

All pairs shortest path

FLOYD WARSHALL ALGORITHM



Done by preparing matrices, A^0 =

	1	2	3	4
1	0	3	∞	7
2	8	0	2	∞
3	5	∞	0	1
4	2	∞	∞	0

Via previous ①

A^1 =

	1	2	3	4
1	0	3	∞	7
2	8	0	2	15
3	5	8	0	1
4	2	5	∞	0

← Remain unchanged.

Remain unchanged ↑

$A^0[2,3]$ | $A^0[2,1] + A^0[1,3]$

2 ← 8 + ∞ (Remains same)

$A^0[2,4]$ | $A^0(2,1) + A^0(1,4)$

∞ | 8 + 7

∞ > 15 (change value)

$A^0[3,2]$ | $A^0[3,1] + A^0[1,2]$

∞ | 5 + 3

∞ > 8 (change to this value)

$A^0[3,4]$ | $A^0[3,1] + A^0[1,4]$

1 | 5 + 7

1 < 13. (Remains same as 54 in matrix)

$A^0[4,2]$ | $A^0[4,1] + A^0[1,2]$

∞ | 2 + 3

∞ > 5 (change value)

$A^0[4,3]$ | $A^0[4,1] + A^0[1,3]$

∞ | 2 + ∞

DELTA | Page No. | Date / /

A^2 =

	1	2	3	4
1	0	3	5	7
2	8	0	2	15
3	5	8	0	1
4	2	5	7	0

→ Remain unchanged.

↑ Remain unchanged

A'[1,3) | A'[1,2]+A'[2,3] | A'[3,1] | A'[3,2]+A'[2,1] | A'[4,1] | A'[4,2]+A'[3,1]
∞ > 5 | 5 | 8+8 | 2 < 2+8

A'[1,4] | A'[1,2]+A'[2,4] | 5 < 16 | A'[4,3] | A'[4,2]+A'[2,3]
7 | 3 + 15 | A'[3,4] | A'[3,2]+A'[2,4] | ∞ | 5+2
7 < 18 | 1 | 8 + 15 = 23 | > 7

A^3 =

	1	2	3	4
1	0	3	5	6
2	7	0	2	3
3	5	8	0	1
4	2	5	7	0

→ Remain unchanged

↑ Remain unchanged

A³[2,1] | A³[2,3]+A³[3,1] | A³[1,2] | A'[1,3]+A'[3,2] | A³[4,1] | A³[4,3]+A³[3,1]
8 | 2+5 | 3 | 5+8 | (2) | 7+5
8 > 7 | (3) < 13

A³[2,4] | A³[2,3]+A³[3,4] | A³[1,4] | A'[1,3]+A'[3,4] | A³[4,2] | A³[4,3]+A³[3,2]
15 | 2+1 | 7 | 5+1 | (8) | 7+8
15 > (3) | 7 > (6)

$$A^1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 5 & 6 \\ 5 & 0 & 2 & 3 \\ 3 & 6 & 0 & 1 \\ 2 & 8 & 7 & 0 \end{bmatrix} \end{array}$$

$A^4[1,2]$ $A^4[1,4]+A^4[4,2]$ $A^4[2,1]$ $A^4[2,4]+A^4[4,1]$ $A^4[3,1]$ $A^4[3,4]+A^4[4,$

5 5 3

$A^4[1,3]$ $A^4[1,4]+A^4[4,3]$ $A^4[2,3]$ $A^4[2,4]+A^4[4,3]$ $A^4[3,2]$ $A^4[3,4]+A^4[4,$

5 2 6

$$A^4 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 3 & 5 & 6 \\ 5 & 0 & 2 & 3 \\ 3 & 6 & 0 & 1 \\ 2 & 8 & 7 & 0 \end{bmatrix} \end{array}$$

Formula $\Rightarrow$ $A^k[i,j] = \min\{A^{k-1}[i,j], A^{k-1}[i,k]+A^{k-1}[k,j]\}$

```
for(k=1; k<=n; k++)
{
  for(i=1; i<=n; i++)
  {
    for(j=1; j<=n; j++)
    {
      A(i,j) = min[A[i,j], A[i,k]+A[k,j]);
    }
  }
}
```

$O(n^3)$

NAÏVE STRING MATCHING

Pattern : $P[m] \rightarrow P[0 \ldots m]$

Text : $T[n] \rightarrow T[0 \ldots n]$

P occurs with shift (S) in T if $0 \leq S \leq n-m$

and

$T[s+0, \ldots s+m-1] = P[0 \ldots m]$

Conditions ⇒ if $0 \leq S \leq n-m$

and $T[s+0 \ldots s+m-1] = P[0 \ldots m]$.

then S is valid shift

other S is invalid shift

	0	1	2	3	4	5	6	7	8	9	10	11	12
T	H	E	L	L	O	_	H	I	B	U	D	D	Y

S →

	0	1
P	H	I

Algo

```
n = T.length
m = P.length
for s=0 to n-m
    if P[0..m-1] == T[s+0 .... s+m-1]
        print "Pattern occured with shift" s
```

RABIN KARP ALGORITHM

	1	2	3	4	5	6
Text ⇒	a	a	a	a	a	b

	1	2	3
Pattern ⇒	a	a	b

The characters in Text are given values and then according found using pattern

a = 1
b = 2
c = 3
d = 4
e = 5
f = 6
g = 7
|
|
z = 26.

Now, in pattern

a	a	b

1+1+2 ⇒ 4. → Hash Code

(hash function)

As pattern is of 3 characters, so start taking three characters from text and adding them.

so

1	2	3
a	a	a

1+1+1 = 3 ≠ 4 (Not found)

so

2	3	4
a	a	a

1+1+1 = 3 ≠ 4 (Not found)

similarly, its continued and slided to next value

(Rolling hash function)

O(n−m+1)

1	2	3	4	5	6	7	8
a	b	c	d	a	b	c	e

Text = abcdabce

n = 8

1	2	3
b	c	e

Pattern = bce

m = 3

hash code

hash value ⇒ 2+3+5 = 10

a = 1
b = 2
c = 3
d = 4
e = 5
f = 6
g = 7
h = 8
i = 9
j = 10

By using Rolling hash function, pattern matches at

6	7	8
b	c	e

and hence hash code matches there.

Drawback

When pattern's hash code matches texts codes but pattern doesn't match, and many alphabets are checked everytime.

1	2	3	4	5	6	7	8	9	10	11
c	c	a	c	c	a	a	e	d	b	a

Text ⇒ ccacccaaedba

1	2	3
d	b	a

Pattern ⇒ dba

4+2+1 ⇒ 7

a = 1
b = 2
c = 3
d = 4
e = 5

but

1	2	3										
c	c	a	c	c	a	a	e	d	b	a		

3+3+1+3+3 = 7

1+1+5 = 7

7 7 7

All these match hash code but are not a pattern.

hence its time consuming in this case

So, if we take such an easy hash function, there is possibility of collision and they are not the pattern.

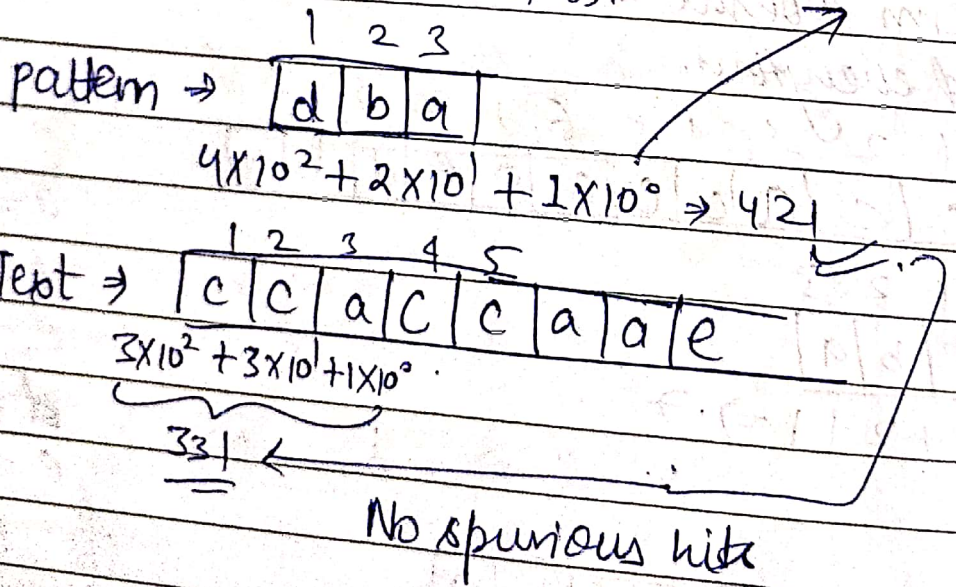
The patterns that don't match but have same hash codes are called spurious hits.

$O(mn) \Rightarrow$ WC

To avoid spurious hits, we need to take a good hash function that doesn't cause collisions easily.

Idea suggested by Rabin-Karp.

Now, our hash function is:- Rabin Fingerprint func.



Algo

```
n = T.length
m = P.length

h^p = hash(P)
h^t = hash(T[0 --- m-1])

for s = 0 to n-m
    if (h^p = h^T)
        if (P[0 --- m-1] = T[s to, --- s+m-1]
            print "Pattern found with shift" s.

    if (s < n-m)
        h^T = hash(T(s+1, ... s+m-1).
```

Knuth- Morris Pratt Algorithm (KMP)

	1	2	3	4	5	6	7	8
String :	a	b	c	d	e	f	g	h

	1	2	3
pattern :	d	e	f

So, in KMP we study the pattern before directly start comparing it with text and with this analysis we can do faster comparison.

Suppose,

	1	2	3	4	5	6	7
pattern ⇒	a	b	c	d	a	b	c

prefix ⇒ a, ab, abc, abcd, ——

Suffix ⇒ c, bc, abc, dabc, ——

Now, is there any prefix same as suffix

prefix ⇒ a, ab, abc, ...

suffix ⇒ c, bc, abc, ...

pattern ⇒ abcdabc

Now, some patterns are discussed :-

P_1 ⇒ abcdabeabf

a) Make zero where the patten which don't repeat and which repeat mark the indexes of repeated values.

so P_1 = abcdabeabf

1 2 3 4 5 6 7 8 9 10

0 0 0 0 1 2 0 1 2 0

P_2 ⇒ abcdeabfabc

1 2 3 4 5 6 7 8 9 10 11

0 0 0 0 0 1 2 0 1 2 3

P_3 ⇒ aabcadaabe

1 2 3 4 5 6 7 8 9 10

0 1 0 0 1 0 1 2 3 0

P_4 ⇒ aaaabaacd

1 2 3 4 5 6 7 8 9

0 1 2 3 0 1 2 0 0

Now, we prepare a table or [π or lps]

↓

longest prefix / suffix

String : | a | b | a | b | c | a | b | c | a | b | a | b | a | b | d | n

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Pattern : | a | b | a | b | d | m

1 2 3 4 5

$O(m+n)$

j

0 1 2 3 4 5

NO Pattern ⇒ | a | b | a | b | d |

0 0 1 2 0

i

SO

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

String ⇒ | a | b | a | b | c | a | b | c | a | b | a | b | a | b | d |

if (S[i] == P[j+1])

i++;

j++;

else

j = Π[value] → if (j == 0) i++;

Algo

n = T.length

m = P.length

Π = compute-prefix function (P)

q ← 0

for i ← 1 to n

do while q > 0 and P[q+1] ≠ T[i]

do q ← Π[q]

if (P[q+1] = T[i]

then q ← q+1

if q = m

print ("Pattern occurs" i-m)

q ← Π[q].