8+9

DELTA Page No. Date / /

Difference of two array

-1 -1 -1 -1 -1 0
10 10 10 11
1 1 1 1 1 1
2 6 9 2 8
8 4 1 8 3

Approach

-1 -1 -1
10 10 11 i
add zero 1 1 1 1 ← j — a
0 2 9 8 ← j — b
0 8 1 3 ← k — diff
new

12
add
29

~~first~~ we need to subtract 1,1,1,1 ~~from~~ to 2 9 8

size of 1,1,1,1 ⇒ 4

size of 2,9,8 ⇒ 3

we need to take the new array as size of biggest one.

~~as~~ ~~the~~ now we set the iterator i, j, k to the end of end of arrays and we move one by one till we get (k>=0) condition

1. Now we subtract if (b[i] (+carry) not satisfy ~~as~~ a[i] then we assign carry to (-1), and we increment the b[i] to 10 more the we ~~#~~subtract.

2.) If it satisfy then simply we subtract and assign c = 0;

3. But there is problem in case 1000 − 1, the b array will stop soon

there → 1000 ← j
no → ...1 → i
no ⇐ k

~~therefore~~ therefore we need to check that if ⇒ i >= 0 we take b[i] normal value if it is less then we take 0.

4. And at last we need to exceed all the preceding "0"

like 0813 ⇒ 813
x

DELTA Page No. Date / /

Rotate an array

eg:- 1 2 3 4 5

if K=0

1 2 3 4 5

K=1

5 1 2 3 4

K=2

4 5 1 2 3

K=3

3 4 5 1 2

K=4

2 3 4 5 1

K=5

1 2 3 4 5

K=6

5 1 2 3 4

K=-1

2 3 4 5 1

Same

Same

approach

Therefore we can conclude that if we get K > then the size of the array then we need to see how minimum many time the array rotate

like if K=101 then the array Same rotate only 1 times

K = 101 / size

101 / 5

remainder = 1

Similarly like

K=6

K = 6/5 = 1

i.e. why K=1 and K=6 are same

Now for the negative value we just subtract the K value from the total size

~~K = 5 + (-1)~~

K = K + count

K = 5 (-1) + 5 = 4

K = 4, K = -1

∴ for K=5-1 we need to rotate the only by 4

∴ K=-1, K=4 are same

Now for the printing the reverse one we just divide the array in 2 parts

first ~~0~~ 1 2 | 3 4 5

P1 | P2

$P_1 = 0 \rightarrow$ a.length - K - 1;

P_2 = a.length - K → arlength - 1;

if K=3

we reverse both array P_1 and P_2

i.e.. 21 / 543

then we reverse the full array

34512 require answer.

Subset of array

3

10, 20, 30

output

— — —

— — 30

— 20 —

— 20 30

10 — —

10 — 30

10 20 —

10 20 30

approach:-

we used the concept of binary here

0 0 0

total no of ~~sets~~ subsets $= 2^3 \Rightarrow 8$

for n $\Rightarrow 2^n$

0 → 0 0 0 so we replace

1 → 0 0 1 0 with "—"

2 → 0 1 0 and 1 with

3 → 0 1 1 array of element.

4 → 1 0 0

5 → 1 0 1

6 → 1 1 0

7 → 1 1 1

* also we ~~want~~ get binary digit of any no by ~~digit~~ dividing with 2

2 | 10	
2 | 5 → 0	
2 | 2 → 1	
1 → 0	

2 | 4	
2 | 2 → 0	
2 | 1 → 0	
0 → 1	

```
//
for (int j = ar.length-1 ; j >= 0 ; j--)
  int r = i%2;
  i = i/2;
  if (r == 0)
  { Set = "-" + set;
  }
  else
  { set = arr[j] + set;
  }
}
```

DELTA Page No. Date / /

* Whether number is prime or not

for the prime number ~~we check for~~ ~~not~~ number is only ~~two~~ ~~four~~ divisible by 2 number i.e... 1 or number itself.

⇒ We need not ~~to~~ to check the whole / or traverse all the no' just we need to check ~~to~~ $\sqrt{n}$ of that number if the no' is not divisible by under $\sqrt{n}$ then it is prime.

⇒ starting point = 2 to $\sqrt{n}$

```
~~if~~ for(i=2; i~~<=~~ i*i<=n; i++) {
    If (n%i==0)
    { not prime
    }
~~else~~
}
after that prime.
```

* Fibonnaci Series

0, 1, 1, 2, 3, 5, 8, 13, 21, 34 - - - -

approach ⇒ 0 — a Sum = 0 starting points ⇒ 0, 1

1 — b — a Sum = a+b

1 — a — b

2 — b — a c = 2

3

5

8

13

now we swap

~~b to Sum~~ a to b

~~a to a~~ b to Sum

then we move the window of two one by one

* The curious case of Benjamin Bulbs (Puzzle)

there are 100 bulbs and initially they are off, the voltage ~~flu~~ fluctuate n times at 1 fluctuation it on the all bulbs.

The bulb that remains on after n fluctuation are only the perfect squares, why because ~~if~~ it depends upon factor that bulb

I have if they have even then it remain eg and having odd factor then it remain on..
all the perfect squares have odd factor.

Ceil and floor (Binary search application)

In this question

array => 12 14 16 18 20 25

if ~~used~~

~~to if data~~ if data is given = 17

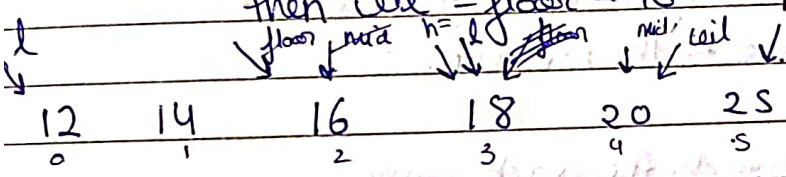
now data = 17

then ceil → 18

floor → 16

and If data = 16

then ceil = floor = 16



7 > 16 ; data > mid

l = mid + 1,

we make mid => floor

again

l = 18 , mid = 20

h = 25 20 > 17, data < mid

then h = mid - 1, h = 18 we set ceil = a[mid]

again

l = 18 => mid = 18

h = 18 then we set ceil = ~~mid~~ a[mid]

ceil = 18

floor = 17

and if data == mid

then we ~~#~~ simply make ~~dat~~ ceil = floor = a[mid]

now how binary search work

```
int l = 0;
int h = a.length - 1;
while (l <= h)
{ int m = (l+h)/2;
  If ( data > m)
    l = m+1;
  else if (data < m)
    h = m-1;
  else
    { } means
      data == mid;
  }
```

we divide the array every time

First Index and last Index

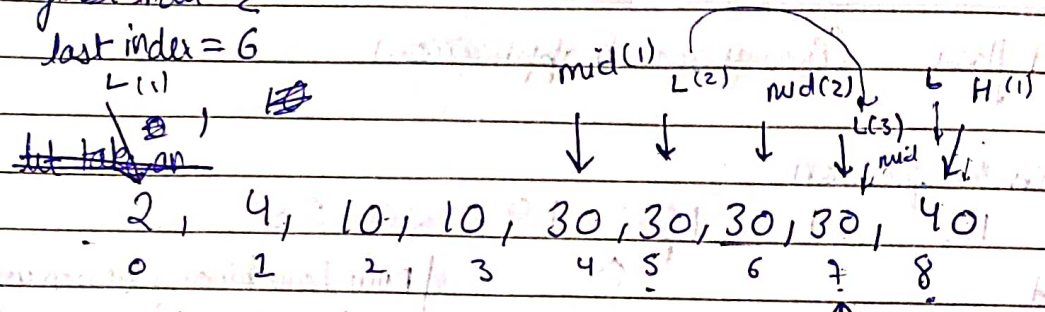
8

2 3 4 4 4 4 4 8

0 1 2 3 4 5 6 7

first index = 2

last index = 6



1. Binary Search (for finding last index)

In this algorithm we wasn't break the loop in third condition ie.. when (data == mid) we simply put the low index just to (mid + 1) we mark that mid as an potential answer and we check further on the right side of that mid if there exist we change that potential one to the new one.

Similarly for the first index we do the exact same but in that we move the H to mid - 1, here we need to check on the leftHand side of that mid, again we mark the mid as potential answer after all iteration we get the first index.

2-D arrays

Matrix Multiplication:

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

3x3 (n x m)

	0	1
0	4	5
1	2	3
2	1	2

3x2 (p x q)

=> 3x2

1x4+2*2+3x1	"
"	"
"	"

now multiplication only be possible when column of first one equal to row of second one.

3 x 2

=> ~~(0,0)+~~ (0,0)*(0,0)+(0,1)*(1,0)+(0,2)*(2,0)
(1,0)*(0,0)+(1,1)*(1,0)+(1,2)*(2,0)
(2,0)*(0,0)+(2,1)*(1,0)+(2,2)*(2,0)

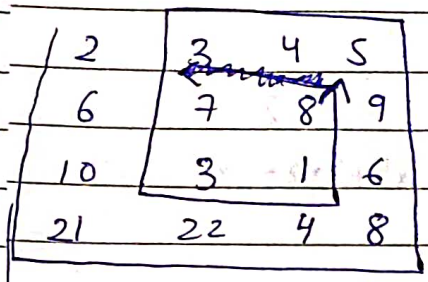
```
If(m==p)
{  c=new int[n][q]
   for(int i=0;i<n;i++)
   {  for(int j=0;j<q;j++)
      {     c[i][j]=0;
            for(int k=0;k<m;k++)
            {   c[i][j]=c[i][j]+a[i][k]*b[k][j];

            }
      }
   }
}
```

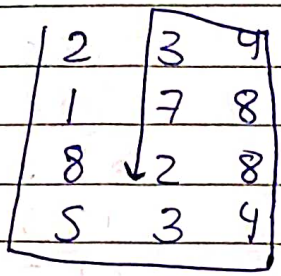
Wave traversal

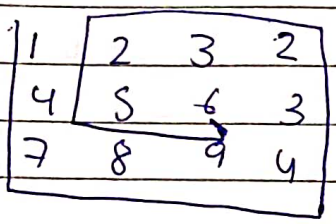
	0	1	2	3
0	1	2	3	4
1	5	6	7	5
2	9	8	2	1

If (j%2==0)
then we move up to down
if not
we move bottom to up.

Spiral display







approach :-

minr = 0;
minc = 0;
maxr = a.length - 1;
maxc = a[0].length - 1;

we first iterate the whole matrix until we reach or just print the all the number inside the matrix

=> Total = n*m ; count = 0 => to n*m

=> first we move left then bottom the right and then top., and we need to also eliminate the corners. that occuring 2 times.

=> minr => maxr , minc = constant ; count <= total every time we increase count ;

next minc+1 => maxc , maxr = constant ; count <= total

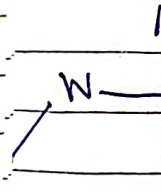
next maxr-1 => minr , maxc = constant ; count <= total

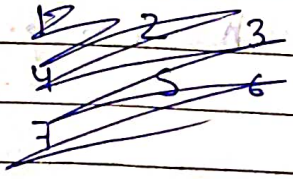
next maxc - 1 => minc+1 ; minr = constant ; count <= total.

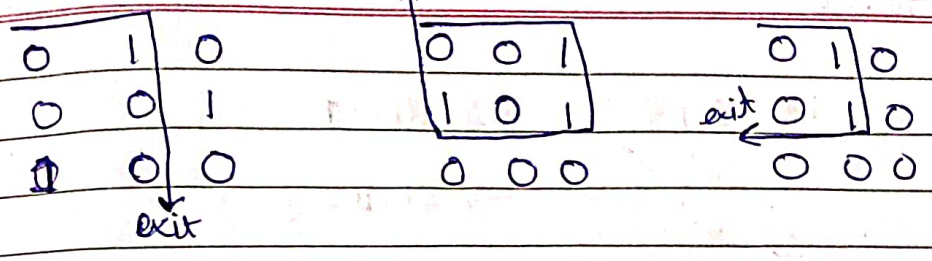
Exit point of A Matrix

Matrix consist of only 0 and 1

when ever 0 encounters it moves start no change in direction but when the 1 encounter it moves by 90° angle and carry on







approach :-

just initially the direction is east

we assign direction to an no: East => j++j

N — E — S

east = 0 west => j--j

south = 1 North => i--j

west = 2 South => i++j

north = 3

east = 0

we plus each and every element of the array the we decide and we take Mode of 4 because at 4 we need 0,

dir = (dir + a[i][j]) % 4;

and break the loop we exist i<0, j<0, i > a.length/, j > a[0].length-1

Rotate By 90 degree

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

→

9	13	9	5	1
8	14	10	6	2
12	15	11	7	3
	16	12	8	4

Transpose.

now approach first make transpose →

logic remain in upper triangle and swap the value of a[i][j] to a[j][i]

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

→ reverse the column → Value

13	9	5	1
14	10	6	2
15	11	7	3
16	12	8	4

reverse logic

i → j ← j

1	2	3	4

swap i and j and i++j j--i

DELTA Page No. Date / /

Shell rotate

11	12	13	14	15	16	17
21	22	23	24	25	26	27
31	32	33	24	35	36	37
41	42	43	44	45	46	47
51	52	53	54	55	56	57

→ Shell-1
→ Shell-2
→ Shell-3

If Shell = 2, rotation = 1

23 24 25 26 22

32 42 43 44 45

approach :→ Spiral concept or Rotation concept.

first we copy the element of ~~spiral~~ shell to 1-D array
rotate the 1-D array by r rotation.
for r = large ⇒ r = r % count;
if r < 0, r = count + rotation.
then we fill the shell from 1-D array.

rotation concept.

5 rotation = no rotation

	1	2	3	4	5
r = 1	5	1	2	3	4
r = 2	4	5	1	2	3
r = 3	3	4	5	1	2
r = 4	2	3	4	5	1
r = 5	1	2	3	4	5
r = -1	2	3	4	5	1

→ r = r % 5

and -1 rotation = 4 rotation

Approach ⇒ a1 (a.length - k - 1), a2 (a.length - k, a.length - 1)

r = 3

1 2 3 4 5 6 7

reverse reverse

4 3 2 1 7 6 5

reverse → require one.

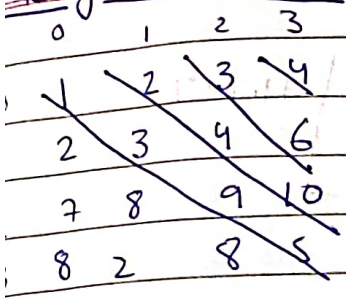
5 6 7 1 2 3 4

// this approach will work in

50 60 30 100

DELTA Page No. Date / /

Diagonal Traversal



→ 1, 3, 9, 5, 2, 4, 10, 3, 6, 4

0,0 0,1 0,2 0,3
1,1 1,2 1,3
2,2 2,3
3,3

gap=3 gap=2 gap=1 gap=0

Sanddle point

Sanddle point is that ~~point~~ value in a matrix in which it has lowest value in his row and ~~this~~ highest value in his column.

11	12	13	14
21	22	23	24
(31)	32	33	34

↓ Sandle point.

Approach :- ~~Sear~~ get the minimum value in a row and make as provisional answer until it became the larger ~~of~~ in that column. if it is the point and not proceed to another.

Search in sorted 2d array

12	13	(14)	15
16	17	18	19
20	21	(22)	23

(22)

approach :- Since the array is ~~sort~~ ~~Sear~~ sorted therefore check the number is ~~greater~~ less than the last ~~last~~ element and greater than the first element, if yes then check in that array. you don't need to traverse the whole array.

~~FS~~ is (22 < 15) no

(22 < 19) no

(23 < 23) yes then search in ~~s~~ last array

Recursions

1. first establish the expectation.
2) Then develop faith
3. Then try to achieve from faith → expectation.

H.L.T

1. Dry run, stack, Base case — L.L.T

Increasing number

n = 5,

⇒ 1
2
3
4
5

expe ⇒ f(5) = 1 2 3 4 5

faith f(4) = 1 2 3 4

If (n == 0) return — L.L.T

⇒ then Pdi(n-1);
Syso(n);

Decreasing no

n = 5

5
4
3
2
1

exp f(5) = 5 4 3 2 1

fail f(4) 4 3 2 1

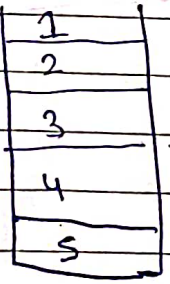
If (n == 0) return — L.L.T

⇒ H.L.T

⇒ Syso(n)
Pdi(n-1);

1. first expect that f(5) will give 5,4,3,2,1
2. faith in f(4) that give 4,3,2,1
3. Build relation ~~between~~ and achieve from faith to expectation
4) f(5) = 5 + f(4)
5) Dry run the code and check the base condition.

output :- 5,4,3,2,1



f(5)

If (n == 0)
return;
System.out.print(n)
fnc(n-1);

Increasing and decreasing

n=3

output:- 3 2 1 1 2 3

now operation:- f(3) = 3 2 1 1 2 3

faith:- f(2) = 2 1 1 2

now achieve faith → operation

```
System.out.print(n);   *1
fun(n-1);              *2
System.out.print(n);   3
```

dry run

base case now+3

1+3n =	0
1+2 n =	1 (+3's to above)
2 1+2n =	2 (+3)
1+2 n =	3 (+3)

3 2 1 1 2 3

2 n 0 → return 1

2 2 1×1 = 2

2 4 2×2 = 4

2 8 4×4 = 16

2 16

2 32

2 64

2 128

2 256

2 512

2 1024

n xn1 xnx

Power of Logarithmic

operation

P(2,1024) → 2^{1024}

1) f(2, 512) → 2^{512} → give answer

3) P(2,1024) = $2^{512} * 2^{512}$

and for odd no's we need to multiple by 2 once more ⇒ $2^{512} * 2^{512} * 2$

```
=> If (n==0)
   return 1;
   int xn1 = Power(x, n/2);
   int xnx = xn1 * xn1;
   If (n%2 == 1)
   { xnx = xnx * x;
   }
```

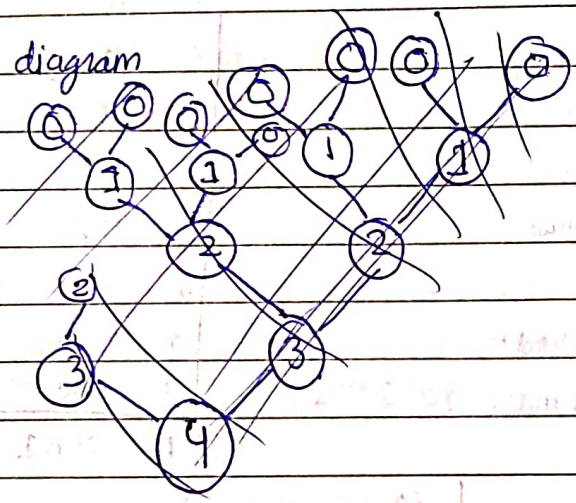
10	0 ↓ out		
10	1	1×10	10
10	2	10	10^2
10	4	10^2	10^4
10	8	10^4	10^8 → output

x n xn1 xnx

10^8

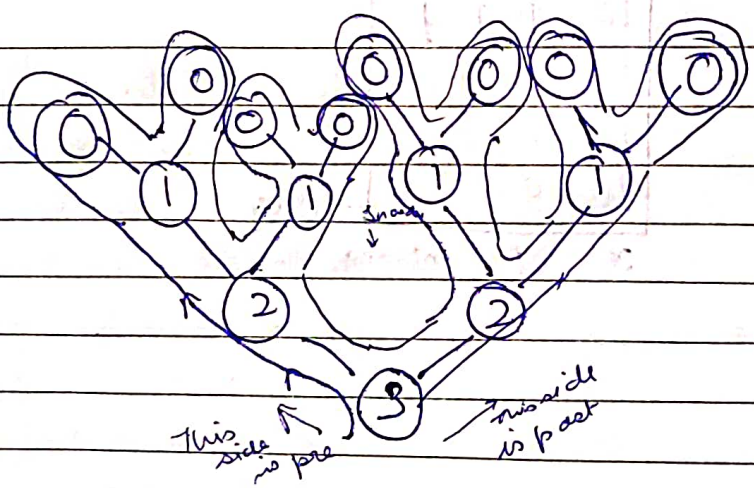
Print Zig-Zag

```
If (n==0)
return;
System.out.print("Pre " +n); → Pre order
nzz(n-1); → left call
System.out.print("In" +n); → In order
nzz(n-1); → Right call
System.out.print ("Post" +n); → Post order
```



we use euler's diagram

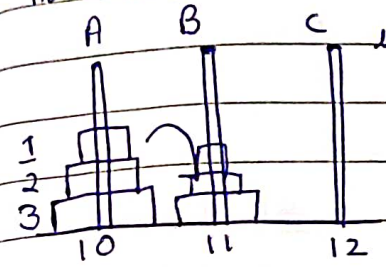
n=4



Pre → 3	In 1	Post 1
Pre - 2	Post 1	In 2
Pre 1	Post 2	Pre 1
In 1	In 3	In 1
Post 1	Pre 2	Post 1
In 2	Pre 1	Post 2
Pre 1	In 1	Post 3

Tower of Hanoi

There are 3 towers all the element in A have to be go in tower B using C.



But there are conditions:-

- Move one disk at a time
- Never place a smaller disk under a large disk
- you can move disk at the top.

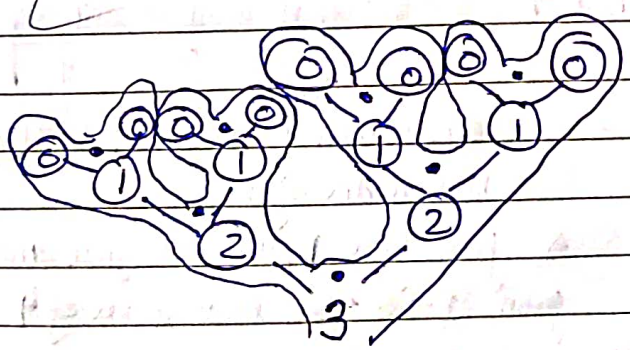
now output :-

1 [10 → 11]
2 [10 → 12]
1 [11 → 12]
3 [10 → 11]
1 [12 → 10]
2 [12 → 11]
1 [10 → 11]

approach :-
expectation :-
f(3) => will put all the element in tower B with following all rules using tower C
faith => f(2) => will put all the element in tower C will following all rules using tower B.
now achieve ~~faith~~ expectation from faith.

=>

```
fun(n-1, A, C, B)
System.out.print(n + A + "→" + C)
fun(n-1, C, B, A)
```



In 1 In 1
In 2
In 1
In 3
In 1
In 2

Recursion using arrays

DELTA Page No. Date / /

1. First Index

⇒ 2 4 7 8 9 4 4 15 6

0 1 2 3 4 5 6 7 8

now if the find value is 4 (Input)
then first Index of 4 is 1.

⇒ array, int x, find.

Exp = f(a, 0, 4) → first Index of 4 → 1

faith f(a, 1, 4) → this will give me from 1 → 8, the first index of 4.

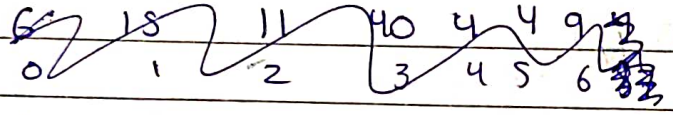
faith → expectation.

now we have to check that a[0] == x if yes then 0 is the first index else index that given by the faith is the first Index.

for the first Index we will check in prior before the call If the current value == x then we return else we call the next one.

a	i	x	
2	0	4	
(4)	1	4	return 1

2. Last Index



15 11 40 4 4 9

0 1 2 3 4 5

now if the find value is 4
then last index of 4 is 4

a	i	x	
10	6	4	base case
9	5	4	-1
4	4	4	-1, return 4
4	3	4	
40	2	4	
11	1	4	
15	0	4	

⇒ expectation f(a, 0, 4) will give last index of 4.

⇒ faith f(a, 1, 4) will give me the from 1 → 5 the last index of 4

now from faith → expectation.

If faith ~~will~~ there are no value in the array if the user enter then faith not able. Suppose, for Suppose 4 is the number present at the 0th index only then faith will not able to find the last index. It is possible that 0th index may have that no if they have return that index else return -1, (because faith not able to find ~~no~~ non first Index).

If faith find the last index then return that index.

⇒ for the last Index we check from the back of the array.

eturn lasttail;

... will work in $O(n^2)$

10 → 10-20-50-60-...
20 30 40 ... 100-40-90-120

DELTA Page No. Date / /

Revise all Indices of array

Arraylists

~~Trees~~

1.) Get Subsequences

Now

for ex, Str = abc

now the Subsequence are

--c

-b-

-bc

a--

a-c

ab-

abc

Now approach :-

operation (Str = abc) = [---, --c, -b-, -bc, a--, a-c, ab-, abc]

faith :- (Str bc) = [---, --c, -b-, -bc]

faith → operation :- now "a" will interact in two way with bc is either yes or no.

Code for sequence :-

```
for (String i : res)
    res.add("" + i);
for (String j : res)
    res.add(a + j);
```

2.) Get KPC

In this question

0 → .;

1 → abc

2 → def

3 → ghi

4 → jkl

5 → mno

6 → pqrs

7 → tu

8 → vwx

9 → yz

Now for ex. input 78

then output ⇒ [tv, tw, tx, uv, uw, ux]

⇒ 7 8

t → v ⇒ tv, uv

u → w tw, uw

x tx, ux

approach :- first we store this string in a single string

now operation f(Str 78) ⇒ [tv, tw, tx, uv, uw, ux]

faith f(Str 8) ⇒ [v, w, x] that it will give all KPC of 8.

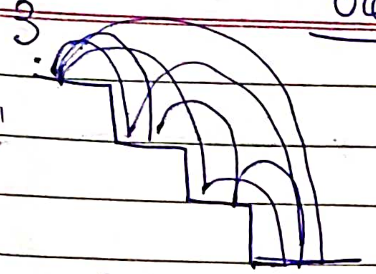
now faith → operation :- all the characters of 7, single by single interact with 8, first t = tv, tw, tx, then u → uv, uw, ux.

Code :-

```
char a = Str.charAt(0); // 7
String res = Str.Substring(1); // 8
Arraylist<String> rres = getKPC(res);
```

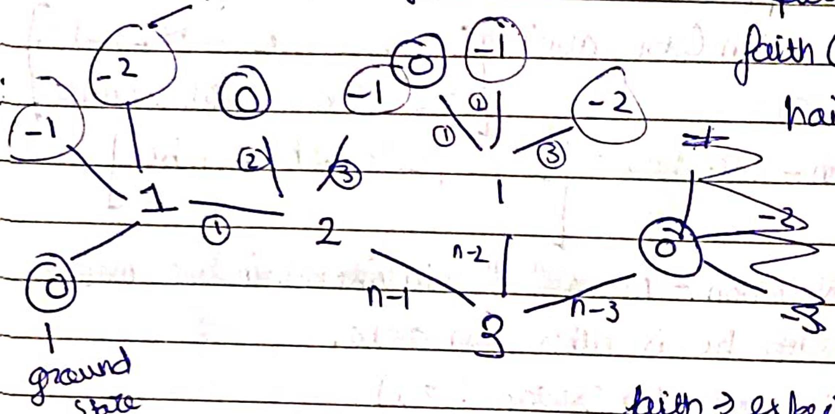
Get stair paths

3



If input = 3
no' of way that he can come to the ground.
~~[111]~~ [111, 12, 21, 3]

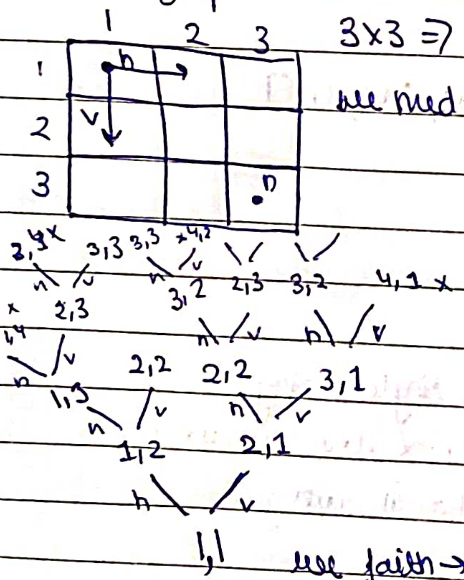
=> Expectation :- ~~3~~ 2 :- [11, 2]



expectation (3) = [111, 12, 21, 3]
faith (n-1)(2) = meraa yeh faith hai ki yeh mushj n-1 kay sarey pair lakey dey dega
Similarly for (n-2) (n-3)

faith -> expectation :- we put 1 in ~~front~~ of front of (n-1) and 2 in front of (n-2) and 3 in front of (n-3)

Get Maze path



3x3 => [hhvv, hvhv, hvvh, ~~f~~ vhhv, vhvh, vvhh]
we need to reach to the "D" Destination.
new expectation :-
faith :- If (j<3)
when hori = (1,2) will give me all the paths
and If (i<3)
vert = (2,1) will give me all the paths

we faith -> expectation we just put h in front of hori and v in front of vert.

Bubble Sort

1.1 5 9 8 2 1

1.2 5 9 8 2 1

1.3 5 8 9 2 1

1.4 5 8 2 9 1

1.5 5 8 2 1 [9]

2.1 5 8 2 1 [9]

2.2 5 8 2 1 [9]

2.3 5 2 8 1 [9]

2.4 5 2 1 [8 9]

3.

3.1 2 5 1 [8 9]

3.2 2 5 1 [8 9]

3.3 2 1 5 [8 9]

3.4 2 1 [5 8 9]

4.

4.1 2 1 [5 8 9]

4.2 1 [2 5 8 9]

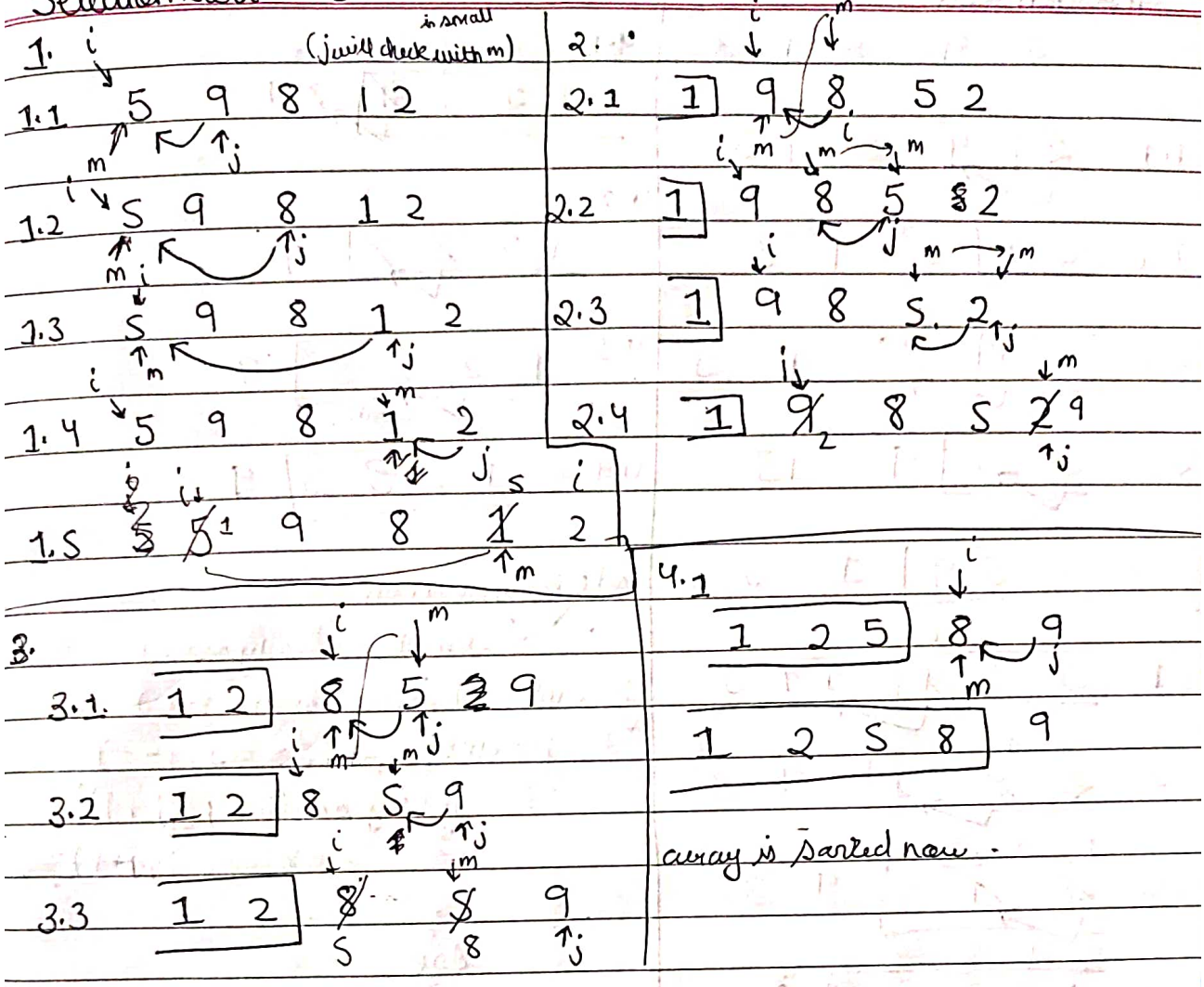
now for n number the outer most loop iterate to (n-1)

⇒ Code for bubble sort

```
for(int i , i<= arr.length -1, i++)
{   for(int j , j< arr.length -i ,j++)
    {   if (is smaller(arr, j+1, j))
        {   swap(arr, j+1, j);
        }
    }
}
```

Time complexity $O(n^2)$

Selection sort $O(n^2)$



code for selection sort

```
for(int i=0; i< arr.length; i++)        jn=5
{   int min = i;                        (0,1,2,3)
    for ( int j = i+1 ; j< arr.length ;j++){
        If(issmaller (arr, j, min){
            min = j;
        }
    }
    Swap(arr,i, min);
}
```

Quick or merge n log n

Time complexity = $O(n^2)$

DELTA Page No. Date / /

Insertion sort

2 9 5 1 3

1.

1.1 [2] 9* 5 1 3

1.1 [2 9] 5* 1 3

2.1 [2 9] 5* 1 3

2.2 [2 5] 9 1 3

=> [2 5 9] 1 3

3.1 [2 5 9] 1* 9 3

3.2 [2 5] 1 9 3

3.3 [2 1 5] 9 3

3.4 [1 2 5 9] 3

4.1 [1 2 5 9] 3 8* 9

4.2 [1 2 3 5] 8 9

4.3 [1 2 3 5] 9 (break.)

4.4 = [1 2 3 5 9]

Code for insertion

0th element is already sorted.

```
for(int j = 1 ;j<arr.length ; j++)
{ for(int j = j-1 ; j>=0 ; j--)
  { if(isgreater(arr, j, j+1))
    { Swap(arr, j, j+1); }
    else
    { break; }
  }
}
```

Merge two Sorted array –

a: 2 5 12 18 20 b: 8

b: 7 9 11 15 25 28 30 35

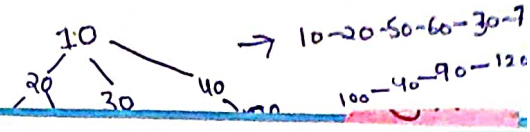
c: 2 5 7 9 11 12 18 18 20 25 28 30

now new array size [13]

1. If(a[i] < b[j])
   then we insert a[i] in c[k]
2. else we insert b[j] in c[k]
3. and rest element of A is store in c(k)
   and rest element of B is store in c[k]

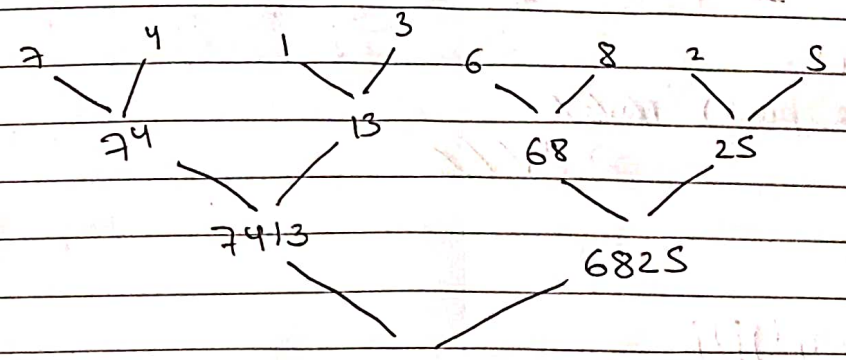
urn lasttail;

approach will work in O(n²)



Merge sort

DELTA Page No. Date / /



code

```
public static int[] mergesort(int[] arr, int lo, int hi)
{   If(lo==hi){
      int[] arr = new int[1];
      arr[0] = arr[lo];
      return arr;
    }
    int mid = (lo+hi)/2;        // get the middle element
    int[] a = mergesort(arr, lo, mid); // faith that it will give sorted array of leftpart
    int[] b = mergesort(arr, mid+1, hi); // faith that it will give sorted array of rightpart
    int[] res = mergetwosortedarray(a,b); // this function will merge the two sorted arra
    return res;
}
```

Partition of an array

Pivot = 5

3s ⇒ 7 9 4 8 3 6 2 1

Pivot

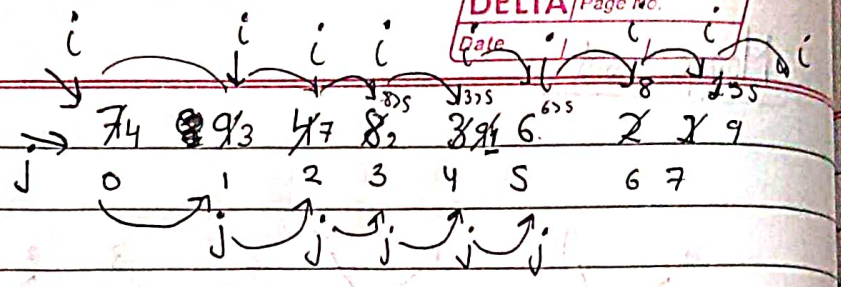
4 3 2 1 5 7 9 8

all the element are small

↳ all the element are greater.

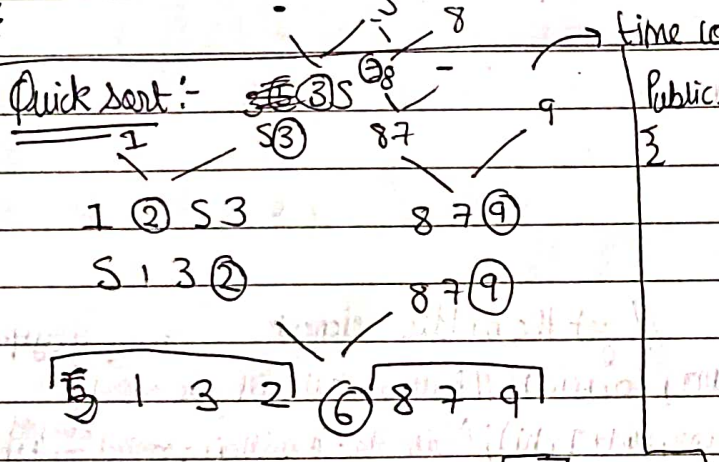
Pivot = 5

DELTA Page No. Date



```
int i=0;
int j=0;
while(i< arr.length)
{   If (arr[i] <=>pivot)   new/exp
    { i++j                 => 4/3
    }
    else
    { Swap(arr, i,j);
      i++j
      j++j
    }
}
```

Quick sort :-



8 5 1 3 7 2 9 6 → Pivot

Time complexity O(n log n)

```
Public static void quicksort (int[] arr, int lo, int hi)
{   If (lo >= hi)
      return;
    int Pivot = Partition (arr, lo, hi
    int Pivot = arr[hi];
    int PivotIndex = partition (arr, Pivot, lo, hi);
    quicksort (arr, lo, pivot-1);
    quicksort (arr, pivot+1, hi);
```

Count Sort :- Count Sort is used where data is present in large amount but they are defined in a particular range. (Stable sort)

Approach:

1. first find the minimum and maximum value of the data and from this we get the range (max − min + 1) is the range or the size of the new array which will store the frequency of the ~~elect~~ each element in the original array

like 2 2 0 0 1 3 (indices 0 1 2 3 4 5)

2 → 2 ~~freq/2~~ After that we sum every consecutive elements, 2, 4, 5, 6

0 → 2 → 4 to decrease by 1, 1, 3, 4, 5, this shows that

1 → 1 → 5 0 → 1 is the area of 2, 1 → 3 is the area of 0 and

3 → 1 → 6 4 → 5 is the area of 1, 3 is the area of 5.

now after that we take the and we decrea element.

code :-

```
int size = ma
int[] freq =
for(int i=0,
{   int ind
    freq[i
}
```

...march will work in $O(n^2)$

DELTA Page No. Date / /

now after that iterate the original array as in a reverse order, we take the last value from array and we check in frequency array (val-min) and we decrease the value by 1, also decrease the frequency of a particular element.

code :-

```
int size = max-min+1;
int[] freq = new int[size];
for(int i=0; i<arr.length; i++)
{   int index = arr[i] - min;
    freq[index]++;
}
```

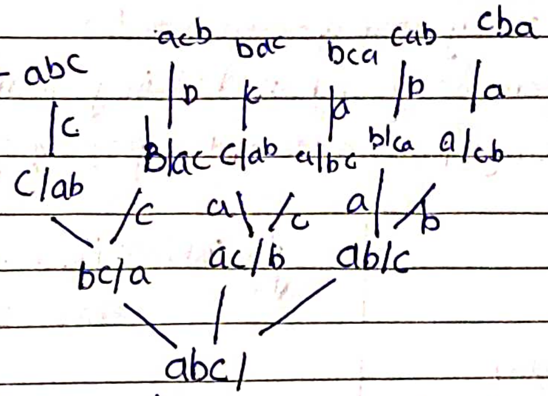
```
for(int i=1; i<freq.length; i++)
    freq[i] += freq[i-1];

int[] ans = new int[arr.length];
for(int i=arr.length; i>=0; i--)
{       int val = arr[i];
        int pos = freq[val-min];
        int index = pos-1;
        ans[index] = val;
        freq[val-min]--;
}
```

```
for(int i=0; i<arr.length; i++)
{       arr[i] = ans[i];
}
```

PRINT PERMUTATIONS

Using Recursion :- abc

string abc



approach :- we need to think that ~~wat~~ what character should be the starting point of each permutation.
therefore we check each and every character

* String ss = str.substring (0, i) + str.substring (i+1);

Iteratively :- (using String builder)

If string is a b c → length ~~fact~~ factorial → 6
(indices 0 1 2)

size / div

3	0
2	0-0
1	0-0
	0-0

abc

3	1
2	0-1
1	0-0
	0-0

$a_0 b_1 c_2$ bac

3	2
2	0-2
1	0-0
	0-0

$a_0 b_1 c_2$ cab

3	3
2	1-0
1	0-1
	0-0

$a_0 b_1 c_2$ acb

3	4
2	1→1
1	0-1
	0-0

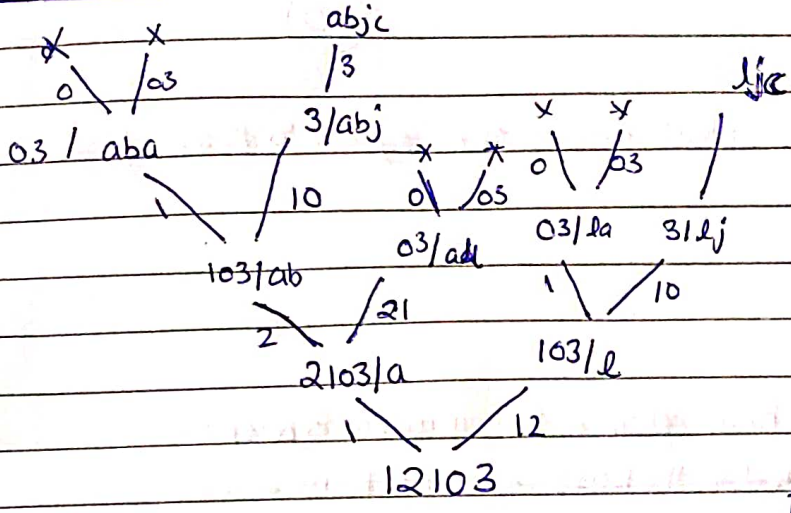
$a_0 b_1 c_2$ bca

3	5
2	1-2
1	0-1
	0-0

cba

```
for(int i=0; i<factorial; i++)
{   int n = length;
    int temp = i;
    While (n>0)
    {   int rem = temp%n;
        int quo = temp/n;
        System.out.print(Sb.charat(rem));
        Sb.deletecharat(rem);
        temp = quo;
        n--;
    }
    syso("\n");
    Sb.append(str);
}
```

PRINT ENCODING



Now exceptions :- 993, 303

```
public static void printEncodings(String str, String asf) {
    if (str.length() == 0) {
        System.out.println(asf);
        return;
    } else if (str.length() == 1) {
        char ch = str.charAt(0);
        if (ch == '0') {
            return;
        } else {
            int chv = ch - '0';
            char as = (char)('a' + chv - 1);
            System.out.println(asf + as);
        }
    } else {
        char ch = str.charAt(0);
        String ss = str.substring(1);
        if (ch == '0') {
            return;
        } else {
            int chv = ch - '0';
            char as = (char)('a' + chv - 1);
            printEncodings(ss, asf + as);
        }
        String ch12 = str.substring(0, 2);
        String res = str.substring(2);
        int chval = Integer.parseInt(ch12);
        if (chval <= 26) {
            char as = (char)('a' + chval - 1);
            printEncodings(res, asf + as);
        }
    }
}
```

Concept

'6' ≠ 6

↓

ascii

54

now if we want to convert '6' → 6

['6' - '0']

↓

54 − 48 → 6

Target Sum Subset

60

10, 20, 30, 40, 50,

We have to print all those subset whose sum equals to K or suppose 60

output :-

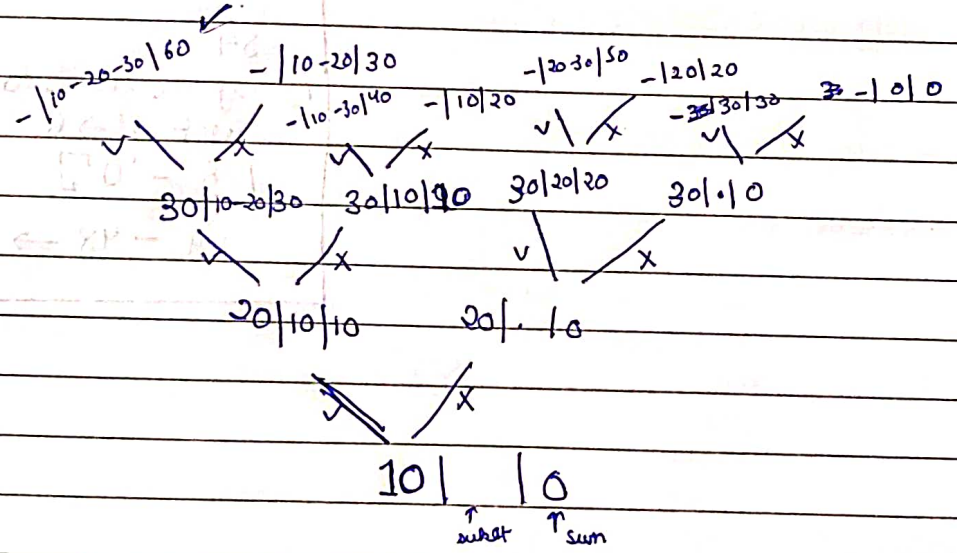
10, 20, 30

10, 50

20, 40

every option of array have option is that it wants to become part of subset or not ~~and in the base case we have s~~

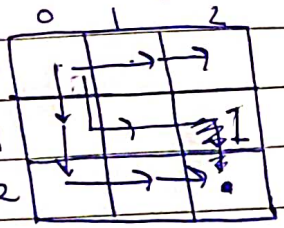
eg:- 10, 20, 30




```
printTargetSumSubset(int arr, int idx, String set, int sos, int tar)
{
    If (arr.length == idx)
    {
        If (sos == tar)
        { sos(set);
        }
    } return;
    printTargetSumSubset(arr, idx+1, set+arr[idx], sos+arr[idx], tar);
    printTargetSumSubset(arr, idx+1, set, sos, tar);
}
```

FLOOD FILL

* If You are given n x m matrix, you have to give all those path from source to destination such that same path does not repeat its path in anywhere.



✓ ddrrr x drrrd
x rrrdd x ddrrrd
x rdrrd

here 0 are are the path and 1 are walls.

things to remember

1. while visiting every cell we need to keep in mind that if that happen [row<0, col<0, row == arr.length, col == arr[0].length, maze[row][col] == 1 visited[row][col] == true]

we need to return if we encounter any of the above condition.

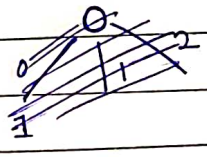
2. after we reach to the destination we need to keep in mind that we also have to unvisit all those cell that we have marked true (why) (because we need all paths not only one path).

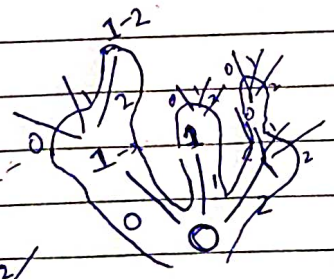
N Queens

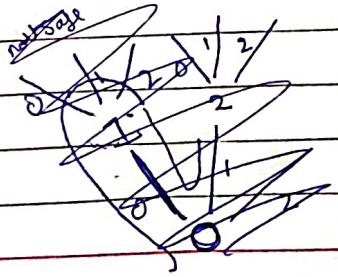
* You are given n*n matrix (chess) such that you have placed number of queen such that they will not kill each other.

	0	1	2
0	*	*	*
1	*	.	*
2	.	.	.

Output 0-0, 1-2 [we can only put 2 queen in 3x3 matrix]



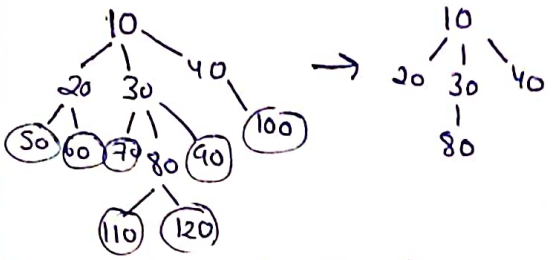




first are rows and options are the column.

* we check for each and every column that it is safe to place the queen.

Generic TREE



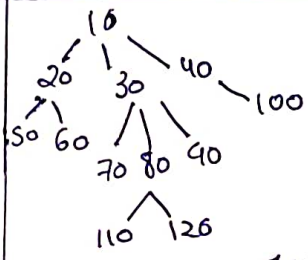

```
move a Leave in  a generic tree
blic static void removeLeaves(Node node) {                    }
// write your code here
for(int i = node.children.size()-1;i>=0;i--)
{
   Node child = node.children.get(i);
   if(child.children.size() == 0)
   {
      node.children.remove(child);
   }
}
for(Node child: node.children)
{
   removeLeaves(child); //faith that it will remove all leave nodes
}
```

//We have remove the node in reverse order because if we start from the front then it cause an ~~error~~ error.

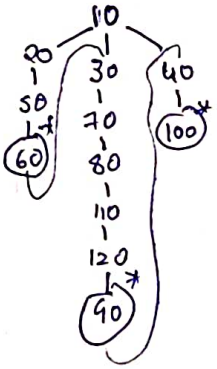
[2|3|4|5|6] → [3|4|5|6], then 3 will be left therefore this error will not occur in reverse.

this approach is use time of O(n)

```
nearize of a generic tree

ublic static Node linearize2(Node node){
 // write your code here
 if(node.children.size() == 0)
 {
    return node;
 }
 Node lasttail = linearize2(node.children.get(node.children.size()-
);
 while(node.children.size() > 1)
 {
    Node lastchild = node.children.remove(node.children.size()-1) ;
    Node secondlastchild = node.children.get(node.children.size()-
;
    Node secondlastchildtail = linearize2(secondlastchild);
    secondlastchildtail.children.add(lastchild);
 }
 return lasttail;
```

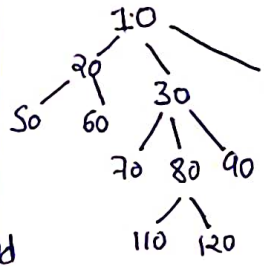


10-20-50-60-30-70-80-100-40-90-120-110

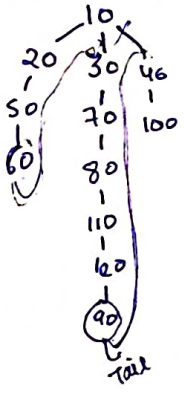


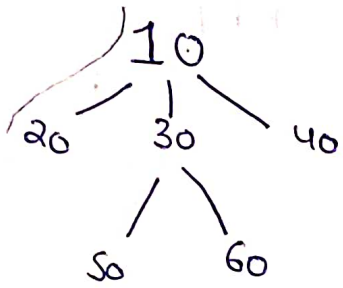
In this it will not only linearize there child but also give there tails also

```
econd approach  // this approach will work in O(n²)
iblic static void linearize(Node node){
 // write your code here
 for(Node child: node.children){
    linearize(child);
 }
 while(node.children.size() > 1)  // this thing have to be done until we get the last child
 {
    Node lastchild = node.children.remove(node.children.size()-1);
    Node secondlastchild = node.children.get(node.children.size()-

    Node secondlastchildtail = getTail(secondlastchild);
    secondlastchildtail.children.add(lastchild);
 }
 private static Node getTail(Node node)
 {
    while(node.children.size() == 1)
    {  node = node.children.get(0);
    }
    return node;
 }
```



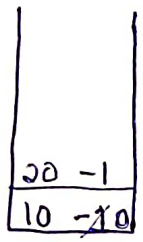
10-20-50-60-30-70-80-100-40-90-120-110



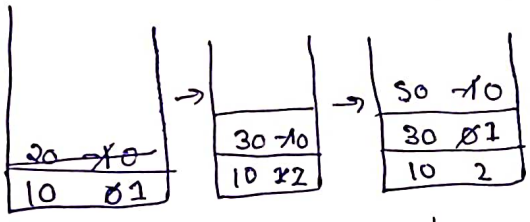


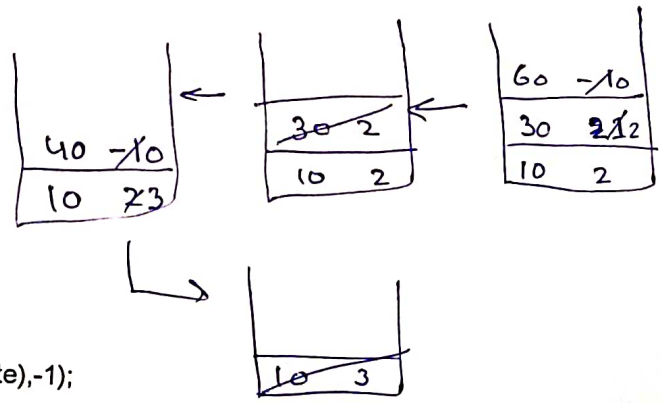
eorder and postorder iteratively

```
atic class Pair
{
    Node node;
    int state;
    Pair(Node node,int state)
    {
        this.node = node;
        this.state = state;
    }
}
ublic static void IterativePreandPostOrder(Node node) {
 // write your code here
Stack<Pair> st = new Stack<>();
st.push(new Pair(node,-1));
String pre = "";
String post = "";
while(st.size()!=0)
{
    Pair top = st.peek();

    if(top.state == -1)
    {
        pre += top.node.data + " ";
        top.state++;

    }
    else if(top.state == top.node.children.size())
    {
        post += top.node.data + " ";
        st.pop();
    }
    else
    {
        Pair cp = new Pair(top.node.children.get(top.state),-1);
        st.push(cp);
        top.state++;
    }
}
System.out.println(pre);
System.out.println(post);
```



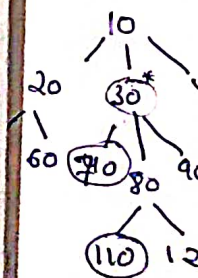
Pre = 10 20 30 50 60 40
Post = 20 50 60 30 40 10





Time complexity = O(n)
space → Height of the tree = O(n)

```
import java.lo

west common a
blic static Array
data) {
f (node.data ==
ArrayList<Integ
path.add(node.
return path;

r (Node child :
ArrayList<Intege
f (ptc.size() > 0)
ptc.add(node.d
return ptc;

isrn new ArrayL

lic static int lca(
vrite your code
rayList<Integer>
rayList<Integer>

i = path1.size()
j = path2.size()

ile(i>=0 && j>=0

--;
--;

urn path1.get(i);
```



pproach :- we
and
ew often that
that is our

Has Path

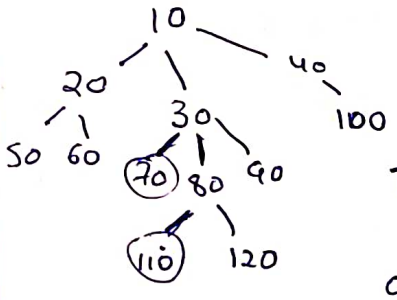
west common ancester

```
blic static ArrayList<Integer> nodeToRootPath(Node node,
 data) {
 if (node.data == data) {
  ArrayList<Integer> path = new ArrayList<>();
  path.add(node.data);
  return path;
 }

 for (Node child : node.children) {
  ArrayList<Integer> ptc = nodeToRootPath(child, data);
  if (ptc.size() > 0) {
   ptc.add(node.data);
   return ptc;
  }
 }

 rn new ArrayList<>();
```

```
ublic static int lca(Node node, int d1, int d2) {
 // write your code here
 ArrayList<Integer> path1 = nodeToRootPath(node,d1);
 ArrayList<Integer> path2 = nodeToRootPath(node,d2);

 int i = path1.size() - 1;
 int j = path2.size() - 1;

 while(i>=0 && j>=0 && path1.get(i) == path2.get(j))
 {
    i--;
    j--;
 }
 i++;
 j
 return path1.get(i);
```

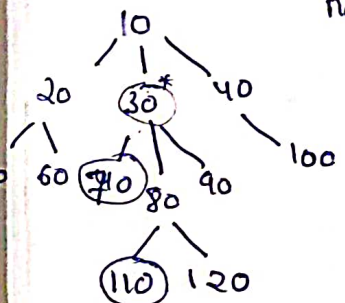
Distance between two node in a generic tree

```
 public static int distanceBetweenNodes(Node node, int d1,
d2){
   // write your code here
   ArrayList<Integer> p1 = nodeToRootPath(node, d1);
   ArrayList<Integer> p2 = nodeToRootPath(node, d2);

   int i = p1.size() - 1;
   int j = p2.size() - 1;

   while(i >= 0 && j >= 0 && p1.get(i) == p2.get(j)){
    i--;
    j--;
   }
   i++;
   j++;
   return i+j;
 }
```

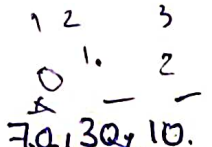


now distance between 70, 110 in 3

This question can also be done with same approach by using Node to root path and in that we can calculate the distance.



now lowest common ancestor of 70, 110 is 30.



approach :- we first get the node to root path of 70 → 70, 30, 10.
and also for the node to root path of 110 → 110, 80, 30, 10

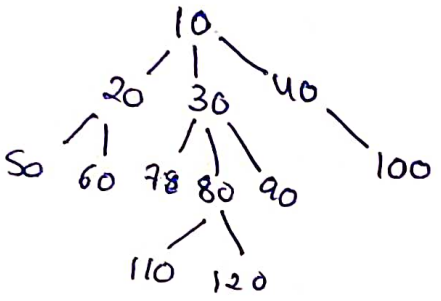
now after that we compare both the first node is not different that is our LCA

eil and floor of a node in a generic tree

```
tatic int ceil;//smallest among larger
static int floor;//largest among smaller

ublic static void ceilAndFloor(Node node, int data) {
 // Write your code here
 if(node.data>data)  {
    if(node.data<ceil)
    ceil = node.data;
 }
 if(node.data<data) {
    if(node.data>floor)
    floor = node.data;
 }
 for(Node child: node.children)
 {
    ceilAndFloor(child,data);
 }
```

// here we are finding the smaller among larger

// here we finding the larger among smaller.



(65)

ceil of 65 → 70

floor of 65 → 60

ceil → smallest among larger [just larger]

floor → largest among smaller [just smaller]

for 5

we have ceil → 10

floor → −∞

we set the

ceil → Integer. Max value

floor → Integer. Min value.

for 125

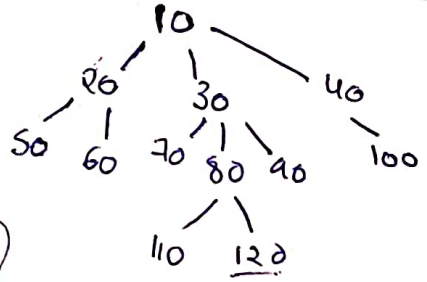
we have floor → 125

ceil → +∞

---

h Largest number in a tree

```
ublic static int kthLargest(Node node, int k){
 // write your code here
 int factor = Integer.MAX_VALUE;
 floor = Integer.MIN_VALUE;
 for(int i=0;i<k;i++)
 {
    ceilAndFloor(node,factor);//this will set the floor
    factor = floor;
    floor = Integer.MIN_VALUE;

 }
 return factor;
```



now if is the value of k=1

the largest value is 120

k=2

largest is 110

k=4

largest is 90

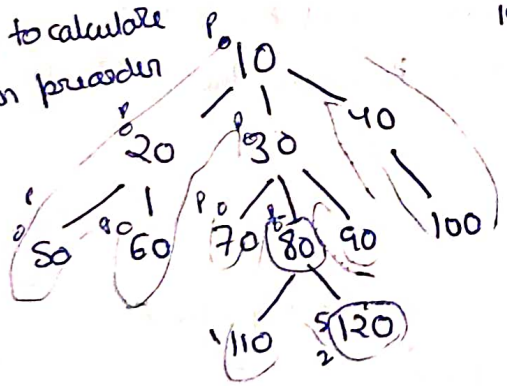
milarly this can also be modified into kth smallest node or mber in a generic tree and this can be done by taking the il(smallest among the largest). ie.. ceil

GRAPH

Has Path

```
avel and change strategy

edecessor and successor of a node in  generic tree
atic Node predecessor;
tatic Node successor;
tatic int state;
ublic static void predecessorAndSuccessor(Node node, int
ta) {
 // write your code here

 if(state == 0)
 {
    if(node.data == data)
    {
       state = 1;
    }
    else
    {
       predecessor = node;
    }
 }
 else if(state == 1)
 {  successor = node;
    state = 2;
 }

 for(Node child: node.children)
 {
    predecessorAndSuccessor(child,data);
 }
```

We have to calculate P & S in preorder



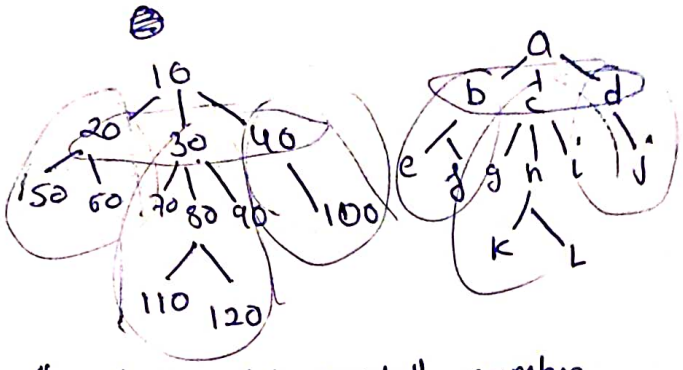
ie.. if data = 110
then S = 120
P = 80

and data = 100
~~S = 90~~ P = 90
S = null

=> data = 10
P = null
S = 20

Approach :- Travel and change Strategy

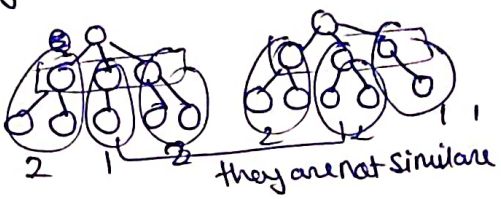
If the state is 0 we move in preorder
if the data found then we increase the state - 1
shows that the data is found and if the data
found the next will be the success than
increase the state.



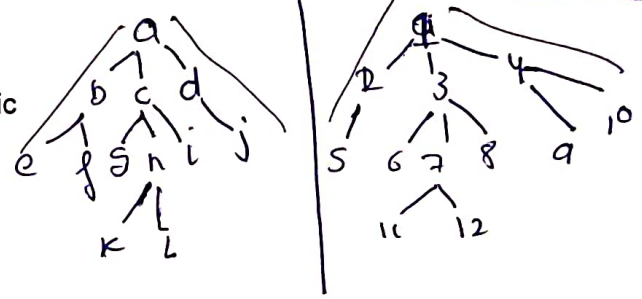

```
e generic tree similar
iblic static boolean areSimilar(Node n1, Node n2) {
 if(n1.children.size() != n2.children.size())
 {
    return false;
 }
 for(int i=0;i<n1.children.size();i++)
 {
    Node c1 = n1.children.get(i);
    Node c2 = n2.children.get(i);
    if(areSimilar(c1,c2)==false)
    {
       return false;
    }
 }
 return true;
}
```

// We just need to count the number of node have similar in both trees of the child.

Wrong case




```
re generic tree is mirror of each other
nd also this question can be a is generic tree is symmetric

iblic static boolean areMirror(Node n1, Node n2) {
 // write your code here
 if(n1.children.size()!=n2.children.size())
 {
    return false;
 }
 for(int i=0;i<n1.children.size();i++)
 {
    Node c1 = n1.children.get(i);
    Node c2 = n2.children.get(n2.children.size()-i-1);
    if(areMirror(c1,c2) == false)
    {
       return false;
    }
 }
 return true;
```



We just need to focus on Shape.

If the first node child equals to the last node child we didn't have to care ~~the~~ about the middle one

=>

nd also this question can be a is generic tree is symmetric

```
iblic static boolean IsSymmetric(Node node) {
// write your code here
return areMirror(node,node);
```

// for the Symmetriary one ~~is~~ thing is important that tree is mirror of itself if it is then it

```
ode with maxim sub tree sum
his can also be done by travel and change strategy)
atic int maxSumNode = 0;
static int maxSum = Integer.MIN_VALUE;
 public static int
eturnSumAndCalculateMaxiumSumInTree(Node node)
{
   int sum = 0;
   for(Node child: node.children)
   {
      int cn =
eturnSumAndCalculateMaxiumSumInTree(child); // faith that it will give sum of child and
      sum += cn;
   }
   sum += node.data;
   if(sum>maxSum)
   {
      maxSumNode = node.data;
      maxSum = sum;
   }
   return sum;
}
```

Same Strategy
travel and Solve

we have faith on

fun

node

sum = 40 60 120 140 170

masum = 40 60 120 140 170

node

faith that it will give sum of child and after that we check and find the maximum sum then we note that sum and that node.data.

```
ameter of a generic tree (travel and change)
static int dia = 0;
public static int calculateHeightDeepest(Node node)
{
  int dph = -1;//deepest height
  int sdh = -1;// second deepest height
  for(Node child: node.children){
     int height = calculateHeightDeepest(child);
     if(height > dph)
     {
        sdh = dph;
        dph = height;
     }
     else if(height > sdh)
     {
        sdh = height;
     }
  }
  int cand = dph + sdh + 2;
  if(cand > dia) {
    dia = cand;
  }
  dph += 1;
  return dph;
```

diameter is the largest number of edges between any two nodes in the diameter.

// approach from the child of A find the deepest child and second deepest child and add +2 of the node.

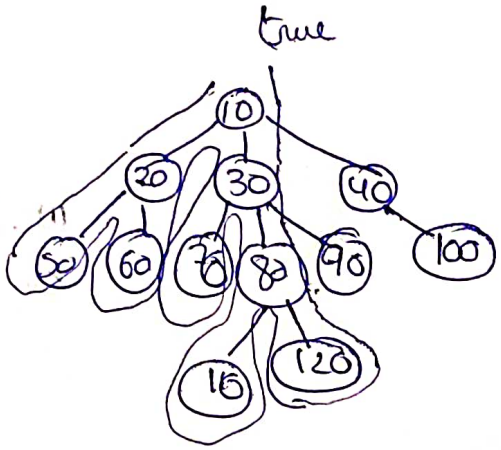
here this approach fail acc. to this max dia = 11 but there is max dia = 12

we need to do this for every node.

In every node it calculate the cand or // max diameter.

S

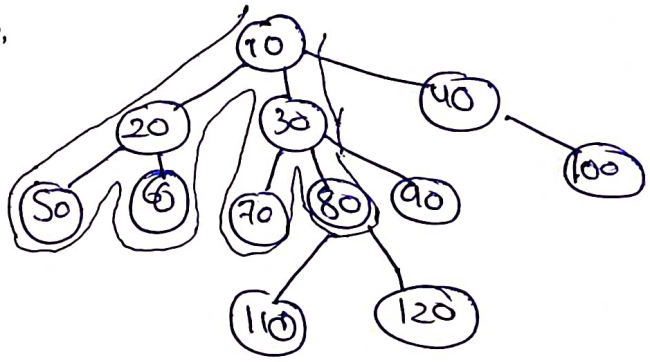
```
nd In generic tree
iblic static boolean find(Node node, int data) {
// write your code here
   if(node.data == data){
      return true;
   }
   for(Node child: node.children)
   {
      boolean res = find(child,data);
      if(res == true)
      return true;
   }
   return false;
```



120

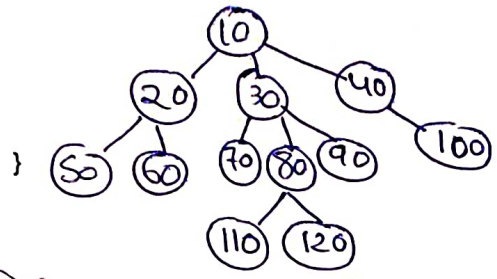
for 80

```
ode To Root Path In Generic Tree
ublic static ArrayList<Integer> nodeToRootPath(Node node,
t data){
 // write your code here
 if(node.data == data)
 {
    ArrayList<Integer> res = new ArrayList<>();
    res.add(node.data);
    return res;
 }
 for(Node child: node.children)
 {
    ArrayList<Integer> pathtillchild =
odeToRootPath(child,data);
    if(pathtillchild.size() > 0)
    {
       pathtillchild.add(node.data);
       return pathtillchild;
    }
 }
 return new ArrayList<>();
}
```



⇒ [80, 30, 10]

We follow the Strategy
Push, remove, print



ne Order Linewise

```
iblic static void levelOrderLinewise3(Node node){
// write your code here
Queue<Node> pq = new ArrayDeque<>();
pq.add(node);
while(pq.size()!=0)
{
   int size = pq.size();
   for(int i=0;i<size;i++){
      node = pq.remove();
      System.out.print(node.data+" ");
      for(Node child: node.children)
      {
         pq.add(child);
      }
   }
   System.out.println();
}
```

size queue

1 | 10 | 20 | 30 | 40

3 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100

6 | 50 | 60 | 70 | 80 | 90 | 100 | 110 | 120

2 | 110 | 120

output :-

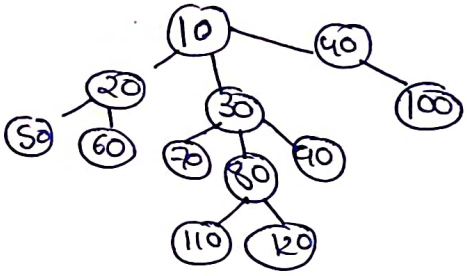
10
20 30 40
50 60 70 80 90 100
110 120

10
20 30 40
50 60 70 80 90 100
110 120

---

proach 2

```
iblic static void levelOrderLinewise4(Node node){
public static class Pair
{
   int level;
   Node node;
   Pair(){

   }
   Pair(Node node,int level){
      this.node = node;
      this.level = level;
   }
}
Queue<Pair> pq = new ArrayDeque<>();
Pair root = new Pair(node,1);
pq.add(root);
int level = 1;
while(pq.size()!=0)
{
   Pair p = pq.remove();
   if(p.level>level)
   {
      level = p.level;
      System.out.println();
   }
   System.out.print(p.node.data+" ");
   for(Node child: p.node.children)
   {
      pq.add(new Pair(child,p.level+1));
   }
}
```



10,1

20,2 | 30,2 | 40,2

40,2 | 50,3 | 60,3

50,3 | 60,3 | 70,3 | 80,3 | 90,3

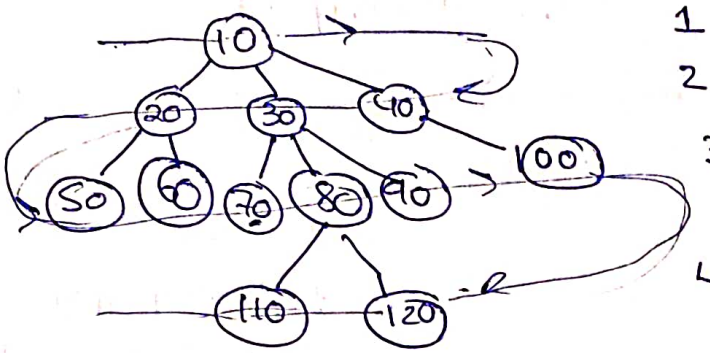
60,3 | 70,3 | 80,3 | 90,3 | 100,3

90,3 | 100,3 | 110,4 | 120,4

10
20 30 40
50 60 70 80 90 100
110 120

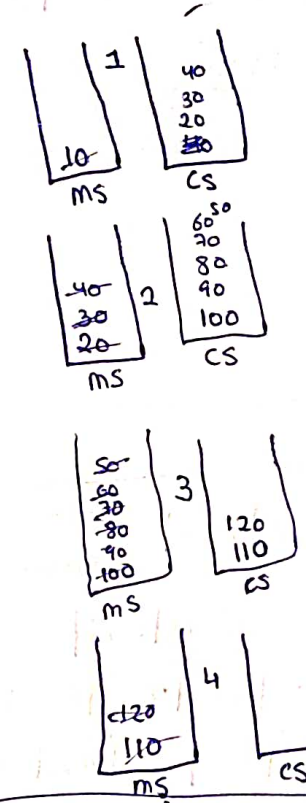
int Zig Zig

```
iblic static void levelOrderLinewiseZZ(Node node){
 // write your code here
 Stack<Node> mst = new Stack<>();
 Stack<Node> hst = new Stack<>();
 mst.push(node);
 int level = 1;
 while(mst.size()!=0)
 {
    node = mst.pop();
    System.out.print(node.data+" ");
    if(level%2 == 1)
    {
       for(int i=0;i<node.children.size();i++)
       {
          hst.push(node.children.get(i));
       }
    }
    else
    {

     for(int i=node.children.size()-1;i>=0;i--)
       {
          hst.push(node.children.get(i));
       }
    }
    if(mst.size() == 0)
    {
       mst = hst;
       level++;
       hst = new Stack<>();
       System.out.println();
    }
 }
```



answere :-

10
40 30 20
50 60 70 80 90 100
120 110

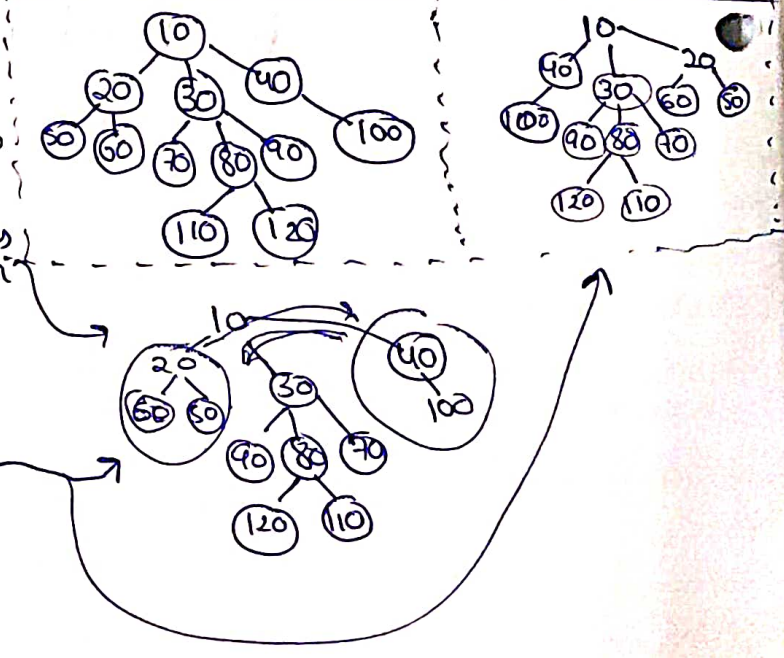


10
40 30 20
50 60 70 80 90 100
120 110

---

rror of a tree

```
iblic static void mirror(Node node){
 // write your code here
 // on mirror function i had a faith that this will give me the data to
e repective node.children to be reverse
 for(Node child: node.children)
 {
    mirror(child); // faith on every child to convert into it's mirror
 }
 // to meet the faith from expectation we need to move all the
ide.children with there family in the reverse order

 Collections.reverse(node.children);//this function will entirely
ange the whole node with there family in the reverse order
```

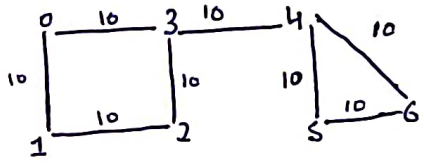


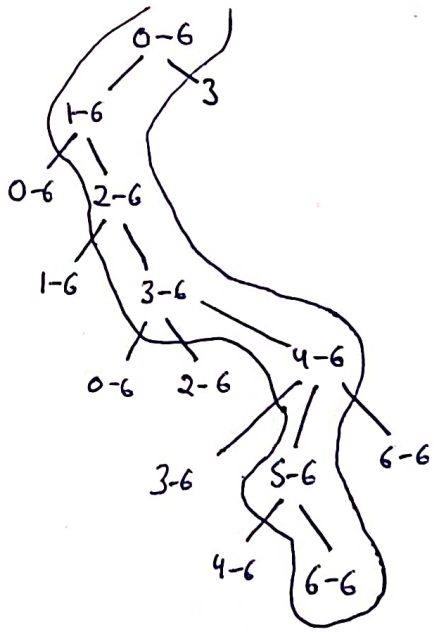
GRAPH

Has Path

```
import java.io.*;
import java.util.*;

public class Main {
  static class Edge {
    int src;
    int nbr;
    int wt;

    Edge(int src, int nbr, int wt){
      this.src = src;
      this.nbr = nbr;
      this.wt = wt;
    }
  }
  public static void main(String[] args) throws
Exception {
    BufferedReader br = new
BufferedReader(new
InputStreamReader(System.in));

    int vtces = Integer.parseInt(br.readLine());
    ArrayList<Edge>[] graph = new
ArrayList[vtces];
    for(int i = 0; i < vtces; i++){
      graph[i] = new ArrayList<>();
    }

    int edges = Integer.parseInt(br.readLine());
    for(int i = 0; i < edges; i++){
      String[] parts = br.readLine().split(" ");
      int v1 = Integer.parseInt(parts[0]);
      int v2 = Integer.parseInt(parts[1]);
      int wt = Integer.parseInt(parts[2]);
      graph[v1].add(new Edge(v1, v2, wt));
      graph[v2].add(new Edge(v2, v1, wt));
    }

    int src = Integer.parseInt(br.readLine()); //Source
    int dest = Integer.parseInt(br.readLine()); //destination
    boolean[] visited = new boolean[vtces]; //boolean array
    boolean res =
hasPath(graph,src,dest,visited);
    System.out.println(res);
    // write your code here
  }
```

```
  public static boolean
hasPath(ArrayList<Edge>[] graph,int src,int
dest,boolean[] visited)
  {
    if(src == dest)        // If src == dest
    {
      return true;
    }
    visited[src] = true;  // else mark visit
    for(Edge edge : graph[src])
    {
      if(visited[edge.nbr] == false){
        boolean haspathnbr = hasPath(graph,
edge.nbr,dest,visited); //fain an nbr that it will give or tells
        if(haspathnbr == true)  us that they have path
        {                       from source to destination.
          return true;
        }
      }
    }
    return false;
  }
}
```





Iterative Depth First Search

```
public static void
IterativeDepthFS(ArrayList<Edge>[] graph,int
src,boolean[] visited)
 {
    Stack<Pair> st = new Stack<>();
    st.push(new Pair(src,src+""));
    while(st.size()!=0)
    {
       Pair rem = st.pop();
       if(visited[rem.v] == true)
       {
          continue;
       }
       visited[rem.v] = true;
       System.out.println(rem.v+"@"+rem.psf);
       for(Edge edge: graph[rem.v])
       {
          if(visited[edge.nbr] == false)
          {
             st.push(new
Pair(edge.nbr,rem.psf+edge.nbr));
          }
       }
    }
```

DFS in ~~Zero~~ Iterative order.

=> It is similar to the BFS but here we use stack instead of Queue

How DFS is different from BFS

In DFS first we go down an reach to the deepest value in the tree or graph and the ender goes up.

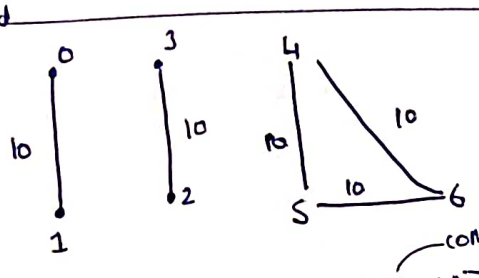
where as in BFS it is more like level order of tree, it increase in a gradual manner like radius.

Get Connected component of Graph

```
 public static void main(String[] args) throws
Exception {
  // same as the first main in the solution//
ArrayList<ArrayList<Integer>> comps = new
ArrayList<>();

    // write your code here
    boolean[] visited = new boolean[vtces];
    for(int v=0;v<vtces;v++) // we check every Vertices
    {
      if(visited[v] == false) // If the Vertices are not visited
      {                       then we make a call
        ArrayList<Integer> comp = new
ArrayList<>();

getallConnectedPair(graph,v,comp,visited);
        comps.add(comp);
      }
    }

    System.out.println(comps);
  }

  public static void
getallConnectedPair(ArrayList<Edge>[]
graph,int src,ArrayList<Integer> comp,boolean[]
visited)
  {

    visited[src] = true;
    comp.add(src);
    for(Edge edge: graph[src])
    {
      if(visited[edge.nbr] == false)
      {

getallConnectedPair(graph,edge.nbr,comp,visit
ed);
      }
    }
  }
}
```

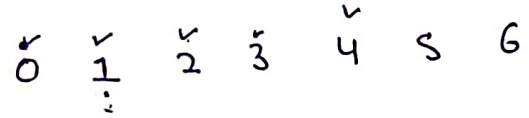
Is Graph is connected

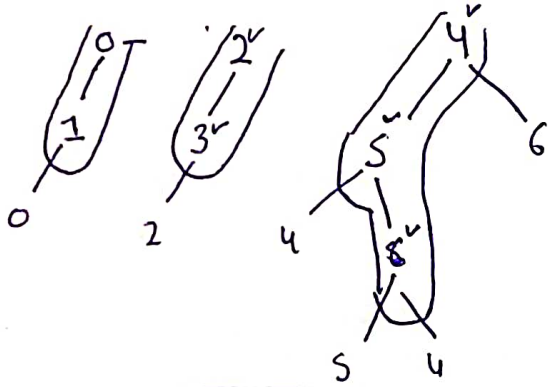
```
if(comps.size() == 1)
    {
      System.out.println("true");
    }
    else
    {
      System.out.println("false");
    }
```



[(0,1), (3,2), (4,5,6)] → comps

comp.



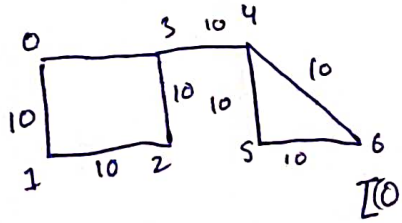


Is Graph connected :-

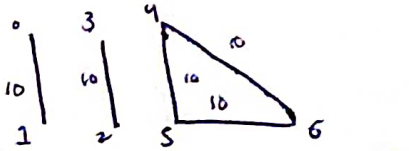
This question is almost equals to the get component of graph.

⇒ If the graph is connected the arraylist of comps have only ~~comp's~~ one long comp component

⇒ If the graph is not connected then it has more that one component. like



[0,1,2,3,4,5,6]



[(0,1), (3,2), (4,5,6)]

we are required to find out how many people will get infected in time t, if the infection spreads to neighbour of infected person in 1 unit of time

SPread Of Infection

```
public static void
getcountofinfectedpeople(ArrayList<Edge>[]
graph,int src,int t,boolean[] visited)
 {
    ArrayDeque<Pair> queue = new
ArrayDeque<>();
    queue.add(new Pair(src,src+"",1));
    int count = 0;
    while(queue.size()!=0)
    {

      Pair rem = queue.removeFirst();
      if(visited[rem.v]  == true)
      {
        continue;
      }
      visited[rem.v] = true;
      if(rem.t > t)
      {
        break;
      }
      count++;
      for(Edge edge: graph[rem.v])
      {
        if(visited[edge.nbr] == false)
        {
          queue.add(new
Pair(edge.nbr,rem.psf+edge.nbr,rem.t + 1));
        }
      }
    }
    System.out.println(count);
}
```

DIJKSHTRA ALGORITHM

Shortest path in Weights (Dijkshtra Algorithm)

```
public static void
DijkstraALgorithm(ArrayList<Edge>[] graph,int
src,boolean[] visited)
 {
    PriorityQueue<Pair> pq = new
PriorityQueue<>();
    pq.add(new Pair(src,src+"",0));
    while(pq.size()!=0)
    {
      Pair rem = pq.remove();
      if(visited[rem.v] == true)
      {
        continue;
      }
      visited[rem.v] = true;
      System.out.println(rem.v+" via
"+rem.psf+" @ " +rem.wt);
      for(Edge edge: graph[rem.v])
      {
        if(visited[edge.nbr] == false)
        {
          pq.add(new
Pair(edge.nbr,rem.psf+edge.nbr,rem.wt+edge.
wt));
        }
      }
    }
```

You are given with graph and vertex
The vertices represent cities and edges represent distance in kms.
You are required to find out the shortest path to each city from the source to destination.

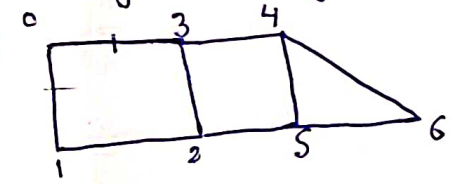
Here we use Priority Queue instead of Queue

• You are required to generate the all moves of a Knight starting in (row, col) such that the knight visits all cells of the board exactly once.

Hamiltonian and cycle

```
if(visited.size() == graph.length-1) // why graph.length-1 bcz in initalizing the function we pass the string psf as src+" " where as we pass the visited as blank therefore psf is always in lead with 1 with respect to visited
    {
        boolean closingedge = false;
        for(Edge edge:graph[src]) //we check in every nbr that i get any edge with original src.
        {
            if(edge.nbr == osrc)
            {
                closingedge = true;
            }

        }
        if(closingedge)
        {
            System.out.println(psf+"*");
        }
        else
        {
            System.out.println(psf+".");
        }
        return;
    }
```

A Hamiltonia path (.) is such which visits all vertices without visiting any twice.
A hamiltonian path become a cycle if there is an edge between first and last vertex.
Hami path (.)
Hami cycle (*)



0 1 2 3 4 5 6 .
0 1 2 5 6 4 3 *
edge

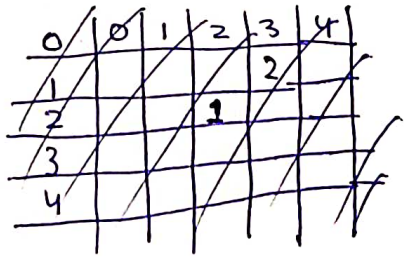
0 3 4 6 5 2 1 *

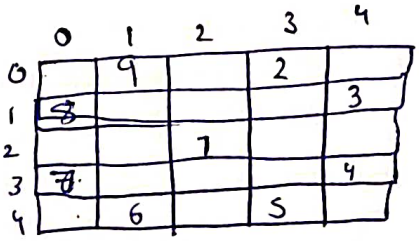
Approach :- This question is very similar to print all path question. only difference we just need to check if there is any edge with source if yes then print * else .

Knight Tour

```
public static void printKnightsTour(int[][] chess,int r,int c,int move) {
        if(r<0 || c<0 || c>=chess.length || r>=chess.length || chess[r][c]>0)
        {
            return;
        }
        else if(move == chess.length*chess.length)
        {
            chess[r][c] = move;
            displayBoard(chess);
            chess[r][c] = 0;
            return;
        }
        chess[r][c] = move;
        printKnightsTour(chess,r-2,c+1,move+1);
        printKnightsTour(chess,r-1,c+2,move+1);
        printKnightsTour(chess,r+1,c+2,move+1);
        printKnightsTour(chess,r+2,c+1,move+1);
        printKnightsTour(chess,r+2,c-1,move+1);
        printKnightsTour(chess,r+1,c-2,move+1);
        printKnightsTour(chess,r-1,c-2,move+1);
        printKnightsTour(chess,r-2,c-1,move+1);
        chess[r][c] = 0;
    }
```

↳ visited once.





Remove mark* work add*

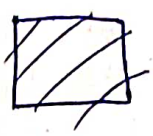
⇒ Breadth First Search

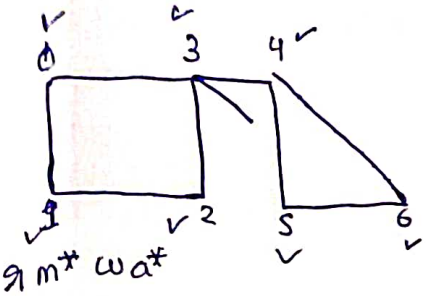
```
public static void main(String[] args) throws
Exception {
    BufferedReader br = new
BufferedReader(new
InputStreamReader(System.in));

    int vtces = Integer.parseInt(br.readLine());
    ArrayList<Edge>[] graph = new
ArrayList[vtces];
    for (int i = 0; i < vtces; i++) {
      graph[i] = new ArrayList<>();
    }

    int edges = Integer.parseInt(br.readLine());
    for (int i = 0; i < edges; i++) {
      String[] parts = br.readLine().split(" ");
      int v1 = Integer.parseInt(parts[0]);
      int v2 = Integer.parseInt(parts[1]);
      graph[v1].add(new Edge(v1, v2));
      graph[v2].add(new Edge(v2, v1));
    }

    int src = Integer.parseInt(br.readLine());
    boolean[] visited = new boolean[vtces];
    ArrayDeque<Pair> queue = new
ArrayDeque<>();
    queue.add(new Pair(src,src+""));
    while(queue.size()!=0)
    {
      // remove mark* work add*
      Pair rem = queue.removeFirst();
      if(visited[rem.v] == true)
      {
        continue;
      }
      visited[rem.v] = true;
      System.out.println(rem.v+"@"+rem.psf);
      for(Edge edge: graph[rem.v])
      {
        if(visited[edge.nbr] == false)
        {
          queue.add(new
Pair(edge.nbr,rem.psf+edge.nbr));
        }
      }
    }
  }
```

- Breadth first search expand in radius,
- BFS is also equals to level order in tree.

Is graph Cyclelic

```
boolean[] visited = new boolean[vtces];
    for(int v=0;v<vtces;v++)
    {
      if(visited[v] == false)
      {
        //traversal
        boolean cycle =
ispair(graph,v,visited);
        if(cycle){
        System.out.println("true");
        return;
        }
      }
    }
    System.out.println("false");
  // write your code here
}

public static boolean ispair(ArrayList<Edge>[]
graph,int src,boolean[] visited)
{
    ArrayDeque<Pair> queue = new
ArrayDeque<>();
    queue.add(new Pair(src,src+""));
    //remove mark* work add*
    while(queue.size()!=0)
    {
      Pair rem = queue.removeFirst();
      if(visited[rem.v] == true)
      {
        return true;
      }
      visited[rem.v] = true;
      for(Edge e : graph[rem.v])
      {
        if(visited[e.nbr] == false)
        queue.add(new
Pair(e.nbr,rem.psf+e.nbr));
      }
    }
    return false;
}
```





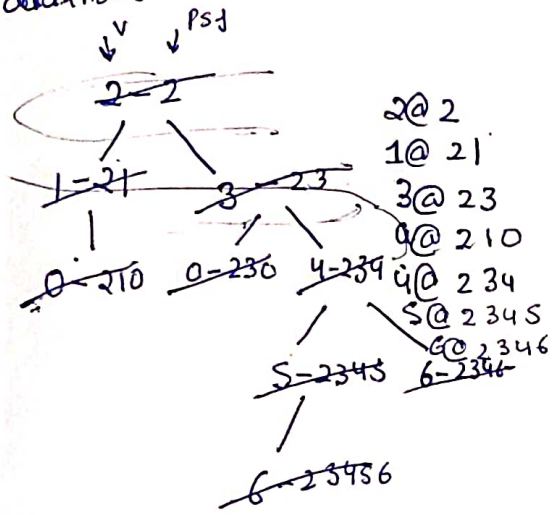
$\mathcal{R}$ m* w a*

$\mathcal{R}$ = remove

mark* = mark those who are not visited

w = work

a* = add those who are not visited




```
Is Graph is BiPartite
public static boolean
iscycleisbipartite(ArrayList<Edge>[] graph,int
src,int[] visited)
 {
    ArrayDeque<Pair> queue = new
ArrayDeque<>();
    queue.add(new Pair(src,src+"",0));
    while(queue.size()!=0)
    {
      Pair rem = queue.removeFirst();
      if(visited[rem.v] != -1)
      {
        if(visited[rem.v] != rem.level)
        {
          return false;
        }
      }
      else{
        visited[rem.v] = rem.level;
      }
      for(Edge edge: graph[rem.v])
      {
        if(visited[edge.nbr] == -1)
        queue.add(new
Pair(edge.nbr,rem.psf+edge.nbr,rem.level+1));
      }
    }
    return true;
 }
```

A graph is called bipartite if it is possible to split it's vertices in two sets of mutually exclusive and exhaustive vertices such that all edges are across sets.

⇒ It is the extension of graph cycle.

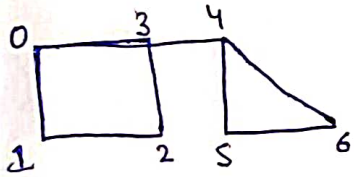
⇒ every non-cyclic graph is bipartite

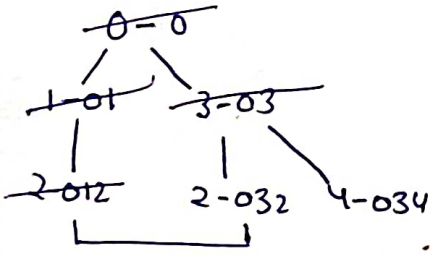
⇒ If cycle is there then it should be even.

⇒ every odd cycle is not bipartite graph.

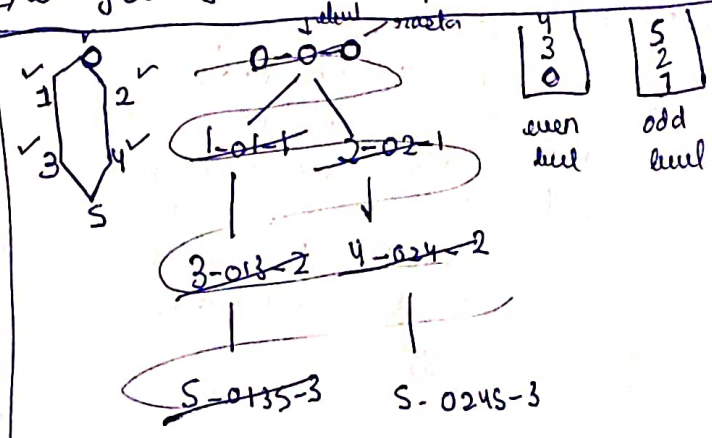
Is graph is cyclic

We use BFS

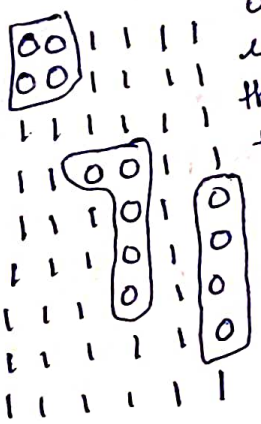




2 is added two times that shows that we access 2 from two path that means. graph is cyclic.



```
Number of Island
{
  boolean[][] visited = new
boolean[arr.length][arr[0].length];
    int count = 0;
    for(int i=0;i<arr.length;i++)
    {
      for(int j=0;j<arr[i].length;j++)
      {
        if(arr[i][j] == 0 && visited[i][j] == false)
        {
          getislandPair(arr,i,j,visited);
          count++;
        }
      }
    }
    System.out.println(count);
    // write your code here
  }
  public static void getislandPair(int[][] arr ,int
i,int j,boolean[][] visited)
  {
    if(i<0 || j<0 || i>=arr.length || j>=arr[0].length
|| arr[i][j] == 1 || visited[i][j] == true)
    {
      return;
    }
    visited[i][j] = true;
    getislandPair(arr,i-1,j,visited);
    getislandPair(arr,i,j+1,visited);
    getislandPair(arr,i,j-1,visited);
    getislandPair(arr,i+1,j,visited);

  }
}
```



0 are the island and 1 is water, we have to determine that how many islands are there in a 2d matrix.

now the simple approach is that we need to take

- care of visited one
- Jha call laga rahan hain wo water toh nahi hai
- Or call marke waha wo call board se bahar na jay.

- You have to find in how many way can we Select a pair of students such that both Students are from different club.

```
Perfect Friend
boolean[] visited = new boolean[n];
    ArrayList<ArrayList<Integer>> comps =
new ArrayList<>();
    for(int i=0;i<n;i++)
    {
      if(visited[i] == false)
      {
        ArrayList<Integer> comp = new
ArrayList<>();
        getallpathPair(graph,i,comp,visited);
        comps.add(comp);
      }
    }
    // System.out.println(comps);
    int count = 0;
    for(int i=0;i<comps.size();i++)
    {
      for(int j = i+1;j<comps.size();j++)
      {
        int pair = comps.get(i).size() *
comps.get(j).size();
        count += pair;
      }
    }
    System.out.println(count);
  }
  public static void
getallpathPair(ArrayList<Edge>[] graph,int
src,ArrayList<Integer> comp,boolean[] visited)
  {
    visited[src] = true;
    comp.add(src);
    for(Edge edge: graph[src])
    {
      if(visited[edge.n] == false)
      {
getallpathPair(graph,edge.n,comp,visited);
      }
    }
  }
}
```

Hamey batay a gya hai ki (0-1) aek club main and (2,3),(4,5,6) aek club hain.

(0-1) C_1
(2-3) C_2
(4-5-6) C_3

Now the ways we can select the clubs $^{2}C_1\ C_2^{2} \rightarrow 4$ ways
$C_1\ C_3 \rightarrow 6$ ways
$C_2\ C_3 \rightarrow 6$ ways

Total ways = 16 pairs

PRIMS ALGORITHM

Minimum wire required to connect all pisces (Prims algorithm)

```
public static void
PrimsAlgorithm(ArrayList<Edge>[] graph,int
src,boolean[] visited)
 {
    PriorityQueue<Pair> pq = new
PriorityQueue<>();
    pq.add(new Pair(src,-1,0));
    while(pq.size()!=0)
    {
       Pair rem = pq.remove();
       if(visited[rem.v] == true)
       {
          continue;
       }
       visited[rem.v] = true;
       if(rem.av!=-1)
       System.out.println("["+rem.v+"-
"+rem.av+"@"+rem.wt+"]");

       for(Edge edge: graph[rem.v])
       {
          if(visited[edge.nbr] == false)
          {
             pq.add(new
Pair(edge.nbr,rem.v,edge.wt));
          }
       }
    }
```

src av wt.

In Prims algorithm we ~~are~~ keep track the acquire vertex, and for the first vertex we haven't print.

How dijkshtra is diffrent from Prims

In Dijshtra we add the ~~edge~~ weight of edge and we add the path so far.

Order Of Compilation (TOPOLOGICAL SORT)

⇒ work in post order of the euler Stack is used here for topological sort to ~~the~~ satisfy the conditions.

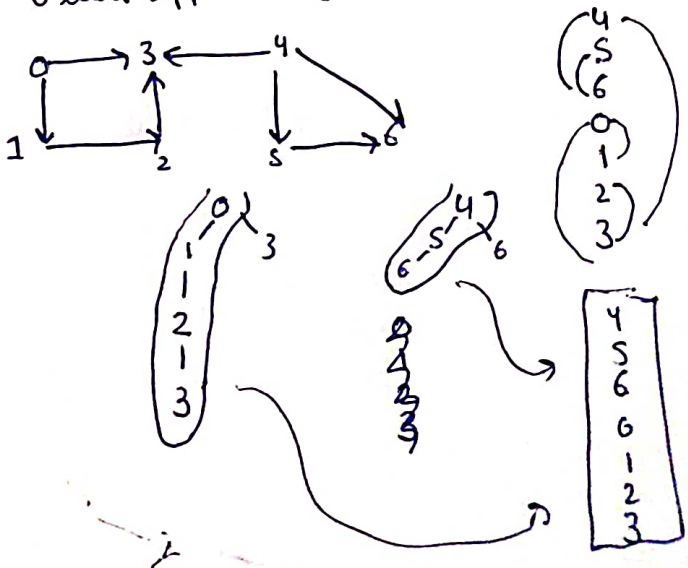
but order of compilation is in reverse.

order of compilation // DFS

```
boolean[] visited = new boolean[vtces];
    Stack<Integer> st = new Stack<>();
    for(int v=0;v<vtces;v++)
    {
       if(visited[v] == false)
       {
          TopoLogicalSort(graph,v,visited,st);
       }
    }
    while(st.size()!=0)
    {
       System.out.println(st.pop());
    }

   // write your code here
 }
  public static void
TopoLogicalSort(ArrayList<Edge>[] graph,int
src,boolean[] visited,Stack<Integer> st)
 {
    visited[src] = true;
    for(Edge edge: graph[src])
    {
       if(visited[edge.nbr] == false)
       {

TopoLogicalSort(graph,edge.nbr,visited,st);
       }
    }
    st.push(src);
 }
```

⇒ order of compilation is always in reverse order of stack print.

⇒ Topological sort :- A permutation of vertices for a directed acyclic graph is called topological sort if for all directed edges UV, U will appears before V in the graph.



```
Multisolver
static String spath;
  static Integer spathwt = Integer.MAX_VALUE;
  static String lpath;
  static Integer lpathwt = Integer.MIN_VALUE;
  static String cpath;
  static Integer cpathwt = Integer.MAX_VALUE;
  static String fpath;
  static Integer fpathwt = Integer.MIN_VALUE;
  static PriorityQueue<Pair> pq = new
PriorityQueue<>();
  public static void
multisolver(ArrayList<Edge>[] graph, int src, int
dest, boolean[] visited, int criteria, int k, String
psf, int wsf) {
    if(src == dest)
    {
      if(wsf<spathwt)
      {
        spathwt = wsf;
        spath = psf;
      }
      if(wsf>lpathwt)
      {
        lpathwt = wsf;
        lpath = psf;
      }
      if(wsf<criteria)  //floor
      {
        if(wsf>fpathwt)
        {
          fpathwt = wsf;
          fpath = psf;
        }
      }
      if(wsf>criteria)  //ceil
      {
        if(wsf<cpathwt)
        {
          cpathwt = wsf;
          cpath = psf;
        }
      }
      if(pq.size()<k)     // Kth largest value.
      {
        pq.add(new Pair(wsf,psf));
      }
      else
      {
        if(wsf > pq.peek().wsf)  // If the incoming player
        {                          is greater than the
          pq.remove();             peek of priority
          pq.add(new Pair(wsf,psf));  then we remove
        }                             and add the
      }                               new player.

      return;
    }
    visited[src] = true;
    for(Edge edge: graph[src])
    {
      if(visited[edge.nbr] == false) // only for those who
      {                                 not visited

multisolver(graph,edge.nbr,dest,visited,criteria,k
,psf+edge.nbr,wsf+edge.wt);

      }

    }
    visited[src] = false; // this can be used when we want to
  }                          visit all the destination from
}                                all different path.
```

* In this question we visit all paths
* Now while travelling we can be able to find out Minimum weight path, maximum weight path, ceil and floor of a particular criteria, and K^{th} largest value.

Ceil → smallest among larger values.
floor → largest among smallest values.

Now for K^{th} largest number we used the approach of team selection.

- first we add all the members till K^{th}
- after that we check in team (Priority Queue) that
- If incoming player have more value or greater than the top of priority queue.
- then we remove and add new player
- approach is like the funnel if good then remove and add new.