**Name – Nithyapriya S**
**RollNo – CS21M510**

**1. A group of n persons have an independent information for gossip known only to himself. Whenever a person calls another person in the group, they exchange all the gossip information they know at that time of calling. What is the minimum number of calls they have to make in order to ensure that everyone of them knows all the information.**

```
#include <iostream>
#include <string>
#include <cmath>
#include <vector>
#include <map>
using namespace std;

#define LEN 6

int main()
{
    string arr[LEN];
    int c = ceil(LEN/2) - 1;
    int count = 0;
    for (int i=0; i< LEN; i++)
    {
        arr[i] = std::to_string(i);
    }

    for (int i=1; i< LEN; i++)
    {
        if(i<=c)
        {
            string g = arr[i-1]+"-"+arr[i];
            arr[i-1]  = arr[i] = g;
            printf("\n %d and %d in call", i-1, i);
        }
        else
        {
            int j = LEN - i+c;
            string g = arr[j-1]+"-"+arr[j];
            arr[j-1]  = arr[j] = g;
            printf("\n %d and %d  in call", j-1, j);
```

```
        }
        count++;
    }

    for (int i=0; i< LEN; i++)
    {
        if(i<c-1 || i> c+2)
        {
            arr[i] = arr[c];
            printf("\n %d and %d  in call", i, c);
        }
        else if (i == c-1)
        {
            arr[c - 1] = arr[c+2] = arr[c - 1] +"-"+ arr[c+2];
            printf("\n %d and %d  in call", c-1, c+2);
        }
        else continue;
        count++;
    }

    printf("\n\n No of calls made %d \n\n After %d calls...\n", count, count);
    for (int i=0; i< LEN; i++)
    {
        printf("\n%d know %s",i, arr[i].c_str());
    }

    return 0;
}
```

Name - Nithyapriya.S

Roll No - CS21M510

# ASSIGNMENT-1

9. GOSSIP PROBLEM

1. Suppose ~~if~~ n = ~~5~~ people, say a b c d e

a b c d e

ab ab c d e

ab ab c de de

ab abc abc de de

ab abc abcde abcde de.

ab abcde abcde abcde abcde.

abcde abcde abcde abcde abcde

→ 1. a to b

2. b to c

3. d to e

4. c to d → c d knows all

5. b to e → b e knows all

6. a to c → a knows all

In first n-1 communication, 2 members know all gossips.

Again, n-2 members need to know all gossips. So, n-3 communications are made among n-2 people.

For $n \geq 4$, time complexity is $2n-4$.

$(n-1+n-3 = 2n-4)$.

For n = 2 ⟹ 1 communication

For n = 3 ⟹ 3 communication.

Time complexity = $O(n)$.

minimum calls = $2n-4$.

Algorithm

If $n \geq 4$,
midperson = ceil (N/2)-1
for person=1 to N-1 ~~person~~
if (person <= midperson)
{
person-1 & person communicates

}
else (person > midperson)
{ p = N - person + mid.
p and p-1 communicates.
}

In the above for loop, n-1 calls are made among n people, and midperson and midperson+1 knows all secret.
Now, n-3 calls are made among others except midperson and midperson + 1.
(person 0 to N-1)
for person = 0 to ~~person~~ N-1 ~~person~~
if person < midperson -1 (or) midperson +2 < person
person and midperson communicates.
(midperson already knows all gossip).
else if person == midperson -1.
midperson -1 and midperson +2 communicates.
(Here midperson knows gossips from person 1 to midperson.
midperson+2 knows gossips from midperson+1 to N) So, after call, they know all gossips.

1st for loop runs for n-1 times.
2nd for loop runs for n-3 times.

So, 2n-4 is the minimum call made between n persons to exchange the gossips.

Take n=6, a b c d e f

As per algo.

In 1st for loop:

a → b. (a = ab, b = ab)

b → c (b = abc, c = abc)

~~c → d~~ (~~c = abcd, d = abcd~~).

f → e [f = ef, e = ef].

d → e (d = ~~abcdef~~ def, e = ~~abcdef~~ def).

c → d (c = abcdef, d = abcdef).

5 times

n-1

6-1=5

Here, c,d knows all gossip.

In 2nd,

a to c (a knows all)

b to e (Earlier, b = abc
e = def.
after call b,e knows all).

3

n-3

6-3=3

Here, c = midperson

d = midperson + 1

b = midperson - 1

e = midperson + 2.

So, 5+3 = 8 ⇒ 2×6-4 = 8 (2n-4); For 6 people 8 calls is the min

**2. If A(x) = anxn + · · · + a1x + a0, then the derivative of A(x), A0(x) = nanxn−1 + · · · + a1. Devise an algorithm which produces the value of a polynomial and its derivative at a point x = v. Determine the number of required arithmetic operations**

```
#include <iostream>
#include <string>
#include <cmath>
#include <vector>
#include <map>
using namespace std;

int main()
{
    #define P 4
    int a[P] = {9,8 , 6, 4};
    int n = P-1;
    long long int val = a[n];
    int p = 5;
    long long int d = 0;
    for (int i=n-1; i>=0; i--)
    {
        val = val * p +a[i];
        d = d*p+ a[i+1] *  (i+1);
    }
    printf("\n value of polynomial =%lld \n value of derivative=%lld", val, d);

    return 0;
}
```

4. $A(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$.

2\. Consider array A stores coefficient of $A(x)$ in following order

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 2 | ... | n |
| $a_0$ | $a_1$ | $a_2$ | | $a_n$ |

Algorithm to find value of polynomial and its derivative at point $x=V$

(Here, n is ~~power of~~ degree of polynomial. So, size of array is n+1)

val = A[n]
derivative = 0
~~size = size of array = n+1~~

~~for i = size-1 to 0~~

for j = n-1 to 0.
  val = val * x + A[j]
  derivative = derivative * x + a[j+1] * (j+1)

Here val is value of polynomial at point $x=V$
derivative is A(x) derivative at point $x=V$

The algorithm involves

(i) Value of polynomial.
  n multiplication
  n addition.

(ii) value at derivative.
  2n multiplication
  n addition.

$A(x) = x_0 + a_1 x + a_2 x^2 + a_3 x^3$.

In Itr 0,

$val = a_3$.

$der = 0$.

In Itr 1, $j = 2$

$val = a_3 * x + a_2$.

$der = 3a_3$

In Itr 2, $j = 1$.

$val = (a_3 x + a_2) x + a_1$.

$der = 3a_3 x + 2a_2$.

In Itr 3, $j = 0$.

$val = (a_3 x^2 + a_2 x + a_1) x + a_0$.

$der = (3a_3 x + 2a_2) x + 1 a_1$

Finally,

$val = a_3 x^3 + a_2 x^2 + a_1 x + a_0$.

$der = 3a_3 x^2 + 2a_2 x + a_1$.

Time complexity $= O(n)$.

Total computation = $3n$ multiplication $+ 2n$ addition.

$n$ - degree of polynomial.

**3. Consider an n ⇥ n array A containing integer elements (positive, negative, and zero). Assume that the elements in each row of A are in strictly increasing order, and the elements of each column of A are in strictly decreasing order. (Hence there cannot be two zeroes in the same row or the same column.) Describe an efficient algorithm that counts the number of occurrences of the element 0 in A. Analyze its running time.**

```
#include <iostream>
#include <string>
#include <cmath>
#include <vector>
#include <map>
using namespace std;

int main()
{
    #define N 3
    #define M 3
    int a[M][N] = { {7 ,8 ,9 },
                    {0 ,4 ,5 },
                    {-1 ,0 ,3 }};

    int i=M-1,j=N-1;
    int n=0, count =0;
    while(i>=0 && j>=0)
    {
        if(a[i][j] == n)
        {
            count++;
            i--; j--;
        }
        else  if(a[i][j] < n)
        {
            i--;
        }
        else  if(a[i][j] > n)
        {
            j--;
        }
    }
    printf("\n No of occurence of 0 %d", count);
    return 0;
}
```

13. n×n array with rows in strictly increasing order and column with strictly decreasing
3. order. Find occurance of 0.

~~A~~lgorithm.

```
i = row -1
j = column -1.
count = 0.
while i ≥ 0 and j ≥ 0.
    if a[i][j] = = 0.
        count ++
        j --
        i --
    else if a[i][j] < n
        i --
    else if a[i][j] > n
        j --
```

Time complexity - O(n)

The algorithm start searching for 0 from the bottom right of the n×n array.

1. if element is 0, then increment the count. start searching the bottom row and left column. It will eliminate current row and column.

2. if element is less than 0, eliminate current row, start searching in next row.

3. If element is greater than 0, eliminate current column, start searching in immediate left column.

Consider below matrix.

4 8 9
0 4 5
-1 0 3.

i = 2, j = 2.

Try 1,

arr[2][2] = 3.

~~se~~ eliminate current column.

j = 1.

Try 2.

arr[2][1] = 0.

count = 1.

i = 1, j = 0 (eliminate current row, column)

Try 3.

arr[1][0] = 0.

count = 2

i = 0, j = -1.

end of loop.

**4. Generalisation : Given a matrix A[1 · · · n][1 · · ·m] where each row is sorted and there is an element common in all rows, find its position. Naive algorithm generalising problem 1 will be O(nm2). Can you obtain a O(nm) algorithm?**

```
#include <iostream>
#include <string>
#include <cmath>
#include <vector>
#include <map>
using namespace std;


int main()
{
    #define row 4
    #define column 5
    int a[row][column] = {
        { 1, 2, 3, 4, 5 },
        { 2, 4, 5, 8, 10 },
        { 3, 5, 7, 9, 11 },
        { 1, 3, 5, 7, 9 },
    };
    std::map<int, std::vector<int>> h;
    for(int i=0;i<row;i++)
    {
        for(int j=0;j<column;j++)
        {
            h[a[i][j]].push_back(j);
        }
    }

    for(auto i: h)
    {
        if(i.second.size() == row)
        {
            printf("\nThe common element in all row is %d. \nThe positions are",
i.first);
            for(auto j:i.second)
            {
                printf("  %d, ", j);
            }
```

```
        }
    }

    return 0;
}
```

2. A[M][N] where each row is sorted.
Find common element and its position.

A. Using a map/ hash table, the problem can be solved in O(mn).

Map with index as key and value as vector can be used.

The vector can be used to store the positions.

Algorithm

```
for i =0 to row-1.
    for j=0 to column-1.
        // To avoid duplicates in a row.
        if arr[i][j] != arr[i][j-1]

            if arr[i][j] is in hash.

                hash[arr[i][j]].push(j);
                //stores current column value.

            else
                insert arr[i][j] in hash.
                hash[arr[i][j]].push(j).


for element in hash:
    if element.vector.size == row.
        print element key.
        //To print position.
        for pos in element.vector
            print pos.
```

Time complexity O(mn)

**5. (a) Two arrays A[1 · · · p] and B[1 · · · q] are strictly increasing.Find the number of common elements in both. That is number of t such that t = A[i] = B[j] for some i, j.**
**(b) How is the solution altered if A[1] <= A[2] <= · · · <= A[p] and B[1] <= B[2] <= · · · <= B[q], that is, the arrays are non-decreasing rather that strictly increasing.**

```
#include <iostream>
#include <string>
#include <cmath>
#include <vector>
#include <map>
using namespace std;

int main()
{
    #define A 8
    #define B 6
    int a[A] = {2,5,5,5,8,9,10,11};
    int b[B] = {2,5,8,8,8,11};
    int i=0, j=0, count=0;
    while(i<A && j<B)
    {
        while(a[i+1] == a[i])
        {
            i++;
        }

        while(b[j+1] == b[j])
        {
            j++;
        }

        if(a[i] > b[j])
        {
            j++;
        }
        else if(a[i] < b[j])
        {
            i++;
        }
        else if(a[i] == b[j])
```

```
        {
            count++;
            i++;
            j++;
        }
    }

    printf("\nNo of common elements %d", count);
    return 0;
}
```

3.
(a) A [P] , B[q] are in increasing order (strictly).
(5) Number of common elements in both.

(1) Without duplicate (strictly increasing).

Algorithm

Given A[P], B[q].

c = 0.
i = 0, j = 0.
while i<p and j<q.

if A[i] > B[j]
j++
else if A[i] < B[j]
i++
else if A[i] == B[j]

c++
i++
j++

c contains common element count.

(1) if $a_i$ is greater than $b_j$, increase b index
(2) if $a_i$ is less than $b_j$, increase a index
(3) if $a_i$ is equal to $b_j$, increase a, b index.

Time complexity - O(n).

(ii) Algorithm for non decreasing.

Along with above algorithm, add below steps

(i) if current element of $a_i$ is same as $a_{i+1}$, increase a index, and continue until different element is found

(ii) if current element of $b_j$ is same as $b_{j+1}$, increase b index and continue until $b_j$, $b_{j+1}$ are different.

```
c=0
i=0
j=0.
while i<p and j<q.
    while a[i+1]==a[i]
        i++
    while b[j+1]==b[j]
        j++.
    if a[i] > b[j]
        j++
    else if a[i]<b[j]
        i++
    else if a[i]==b[j]
        c++
        i++
        j++
c contains count.
```