TV CHANNEL SCHEDULER

P9

GitHub Repo Link: https://github.com/ChhabhayaManan/CipherX
Team ID: CipherX

Member Names:

Manan Chhabhaya - 202301222
Kresha Vora - 202301231
Anushka Prajapati - 202301224
Kashyap Ajudiya - 202301239

QUESTION DESCRIPTION

P9

We have given a text file containing M Distinct Series, and N family members, each having a favorite series and Availability slots. We have to schedule their favorite series into their availability slots such as The Scheduler needs to minimize total episode misses. There can be the same favorite.

PSEUDO CODE

Assumptions:

We have assumed that there are 168 slots each hour, and 24 slots each day. Our scheduler has already recorded all episodes of the series. We have assumed that here the number of favorite series which has been recorded cannot exceed 168 (total slots) because the recorder can record one series at a time.

Algorithm Functions:

```
Function storeSlotVsSeries(memTotal, slotTotal, seriesTotal):


slotVsSeries.resize(slotTotal)
For (i = 0 To slotTotal) then
   SlotVsSeries[i].resize(seriesTotal+1){0}
endFor
For i = 0 To slotTotal
    For j = 0 To seriesTotal
        For k = 0 To memTotal
```

```
        If (memVsSlot[k][i] == True && memVsSeries[k][j] == True ) then
        slotVsSeries[i][j]++
        Endif
            endFor
        slotVsSeries[i][seriesTotal] += slotVsSeries[i][j]
    endFor
endFor

Function findMax(memTotal, slotTotal, seriesTotal):
        Initialize mAX and cURR as empty vectors
        While (mAX[2] != 0) do
                Clear mAX and set mAX[0], mAX[1], mAX[2] to 0
                For ( I = 0 To 167) do
                        For ( j = 0 To 167) do
                        If (slotVsSeries[i][j] > mAX[2])  then
                                mAX.clear()
                                mAX.push_back(i)
                                mAX.push_back(j)
                                mAX.push_back(slotVsSeries[i][j])
                                For ( k 0 To memTotal)
                                If (memVsSlot[k][i] == true && memVsSeries[k][j] == true) then
                                mAX.push_back(k)
                                Endif
                                endfor
                        Endif
                        Else if ( slotVsSeries[i][j] == mAX[2] ) then
                        cURR.clear();
                        for( k = 0 To memTotal){
                        if(memVsSlot[k][i] == true && memVsSeries[k][j] == true) then
                                cURR.push_back(k);
                        endif
                        endfor
                        integer count1=0,count2=0
```

```
                For ( k = 0 To slotTotal)
                        For each member l in cURR:
                        If (memVsSlot[k][i] == True && memVsSeries[k][j] == True ) then
                        count1++
                        Endif
                        Endfor
                        For ( l = 3 to mAX.size()) do
                        If (memVsSlot[mAX[l]][k] == true && memVsSeries[mAX[l]][j] == true) then
                        count2++
                        Endif
                        Endfor
                Endfor
                If count1 < count2 then
                mAX.clear()
                mAX.push_back(i)
                mAX.push_back(j)
                mAX.push_back(slotVsSeries[i][j])
                For (k from 0 to memTotal)
                        If (memVsSlot[k][i] == True && memVsSeries[k][j] == True) then
                        mAX.push_back(k)
                        Endif
                Endfor
                Endif
        If (mAX[2] == 0) then
                Return
        endif
 pq.push(make_pair(mAX[0], mAX[1]))    // pq is priority queue that stores pair slotwise.
For (l = 3 to mAX.size())
        For (k = 0 to slotTotal)
        If (memVsSlot[mAX[l]][k] == true && memVsSeries[mAX[l]][mAX[1]] == true) then
                slotVsSeries[k][mAX[1] - -
                slotVsSeries[k][seriesTotal]- -

        Endif
```

```
            endfor
             memVsSeries[l][mAX[1]] = false
        endfor

        For (k = 0 to memTotal)
            memVsSlot[k][mAX[0]] = false
        endfor
        For (k = 0 to seriesTotal+1)
            slotVsSeries[mAX[0]][k] = 0
        Endfor
Endwhile
```

TIME COMPLEXITY

N = number of member

M = number of series

Time complexity of each functions:

Input() : O(N*M*168)

storeSlotVsSeries() : O(N*M*168)

findMax() : O(168*168*M*(N+(168*N)+N))

{As I have mentioned in assumptions, The number of series M <= 168 because we can't show more than 168 series in 168 slots}

Overall Time complexity: O(N) with assumption

O(N*M) if M is variable(can exceed 168)

SPACE COMPLEXITY

Space complexity of arrays created into the code:

memVsSlot : O(N*168)

memVsSeries : O(M*N)

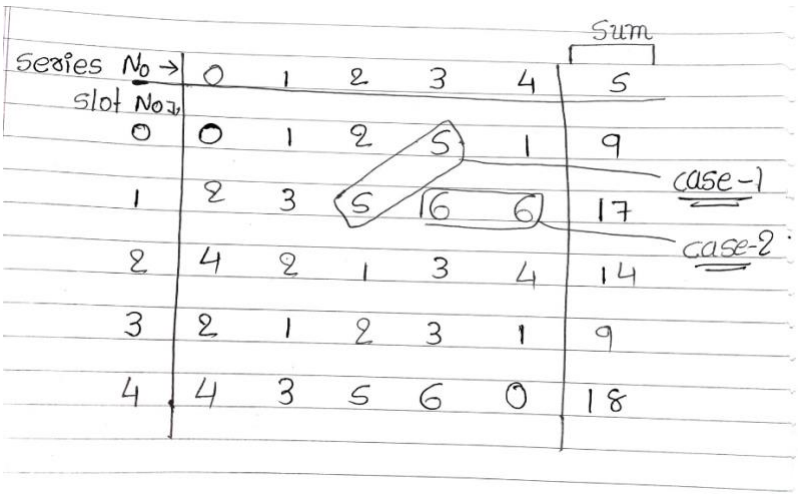
slotVsSeries: O(168*M)

as per our assumption

Overall Space Complexity: O(N) {M<=168}

O(M*N) if M is variable and exceeds 168

EXPLAINATION OF ALGORITHM

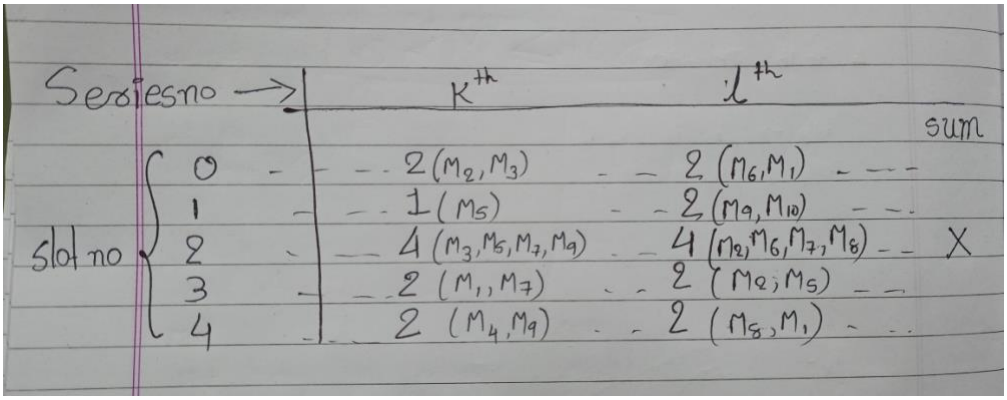


- SlotvsSeries array contains the int number of members who are available at that slot number(Row) and also want to watch that Series(Column Number).
- The value in the last column is the sum of all values in that row.
- Number represents that if we telecast that series on that slot then that number of members is reduced from that series.
- Function **findMax()** is used to find the maximum number.
-
- Now we need to traverse the 2d array to find the maximum number.

Case 1–

For different numbers, we have to store the maximum number, row, column and the members.

Case 2 – (to prioritize one maximum number from two same numbers)



For the Kth series if we telecast at slot 2 then every member at that place(M3, M6, M7, M9) should be removed from that column after the telecast. (Ex. From slot 0 M3, from slot 1 M5 etc.)

Count 1 = number of members removed from Kth column.

Likewise for Lth series,

Count 2 = numbers of members removed from Lth column.

Count represents the number of ways in which the members of that place can watch that series in all slots

Reason: if we telecast series that have a higher count number then all members in that place still have more available slots so members can watch that series in other slots also. So we should prefer to telecast that series which has a lesser count(they don't have more opportunity to watch that series in other slots).

THE RATIONALE BEHIND DATA STRUCTURE

Initially, we thought of storing the input data as a 3D array (the array made using a linked list), but then we realized that it would be much better and efficient to use 2 2D-arrays to store the input data. The space complexity of a 3D array in P9 is N x M x 168. We can reduce it using 2 2D arrays of N x M and N x 168.

Here we have used 1D and 2D vectors. Vector is a dynamic array. Unlike static arrays, a vector allows for efficient insertion and deletion of elements at the end or in the middle of the sequence. Vector uses array-like indexing which makes it easy to use. We can access any element of the vector in constant time(by using an index). This property makes vectors suitable for our capstone problem where fast random access to elements is required. In the findMax function initially, we don't know the number of members available for ith in slot and jth series. Hence, it is beneficial to use vectors for cURR and mAX as it allows dynamic memory allocation.

Also, we have used a priority queue to store the pair(slot, series) . The priority queue stores the pair in a slot-wise order. The time complexity for insertion in Priority Queue is O(log n) and for dequeue it is O(1).

OUTPUT

Terminal Based output:

Input file --

in copy.txt

```
6 6
TMKOC, Anupama, FRIENDS, ChotaBheem, Vir, SharkTank

Manan
MON 01-02, MON 02-03, MON 03-04, MON 00-01
Anupama, FRIENDS, ChotaBheem
Naman
MON 01-02, MON 03-04
TMKOC, Vir
Keval
MON 00-01, MON 01-02, MON 02-03, MON 05-06
FRIENDS, ChotaBheem, Vir, SharkTank
Parthiv
MON 02-03
Vir
Prince
MON 01-02, MON 02-03, MON 03-04
TMKOC, Vir, SharkTank
Prem
MON 00-01, MON 04-05, MON 05-06
Vir, SharkTank
```

memVsSlot and memVsSeries array as output –

```
memVsSlot array:
0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
memVsSeries array:
0 1 1 1 0 0
1 0 0 0 1 0
0 0 1 1 1 1
0 0 0 0 1 0
1 0 0 0 1 1
0 0 0 0 1 1
```

Process on SlotvsSeries array :

```
SlotvsSeries array :
0 1 2 2 2 2
2 1 2 2 3 2
1 1 2 2 3 2
2 1 1 1 2 1
0 0 0 0 1 1
0 0 1 1 2 2
```

```
Slot 3 Series 5 :
after a step
0 1 2 2 1 2
2 1 2 2 1 2
0 0 0 0 0 0
2 1 1 1 1 1
0 0 0 0 1 1
0 0 1 1 1 2
```

```
Slot 1 Series 3 :
0 0 0 0 0 0
2 1 0 2 1 2
0 0 0 0 0 0
2 1 0 1 1 1
0 0 0 0 1 1
0 0 0 1 1 2
```

```
Slot 2 Series 1 :
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 1 0 1 1 1
0 0 0 0 1 1
0 0 0 1 1 2
```

```
Slot 6 Series 6 :
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 1 0 1 1 1
0 0 0 0 1 0
0 0 0 0 0 0
```

```
Slot 4 Series 2 :
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 1 0
0 0 0 0 0 0
```

```
Slot 5 Series 5 :
after a step
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
```

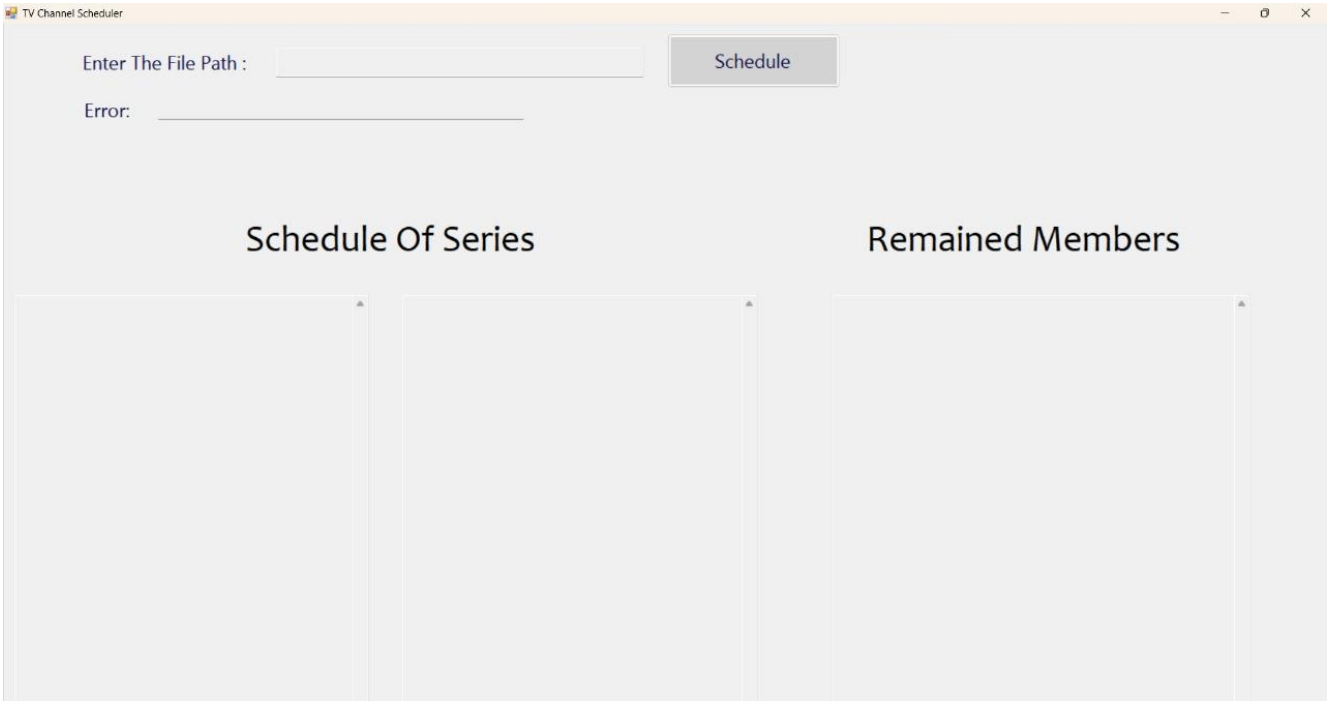
Main output :

```
================================================================================
Remaining members:
================================================================================
Manan       : ChotaBheem
Naman       : Vir
Keval       : ChotaBheem
Prince      : SharkTank
================================================================================
The schedule is as follows:
================================================================================
MON 0-1 : FRIENDS
MON 1-2 : TMKOC
MON 2-3 : Vir
MON 3-4 : Anupama
MON 4-5 : Vir
MON 5-6 : SharkTank
================================================================================
```

It can run for any big file, but for simplicity, we used small text input.

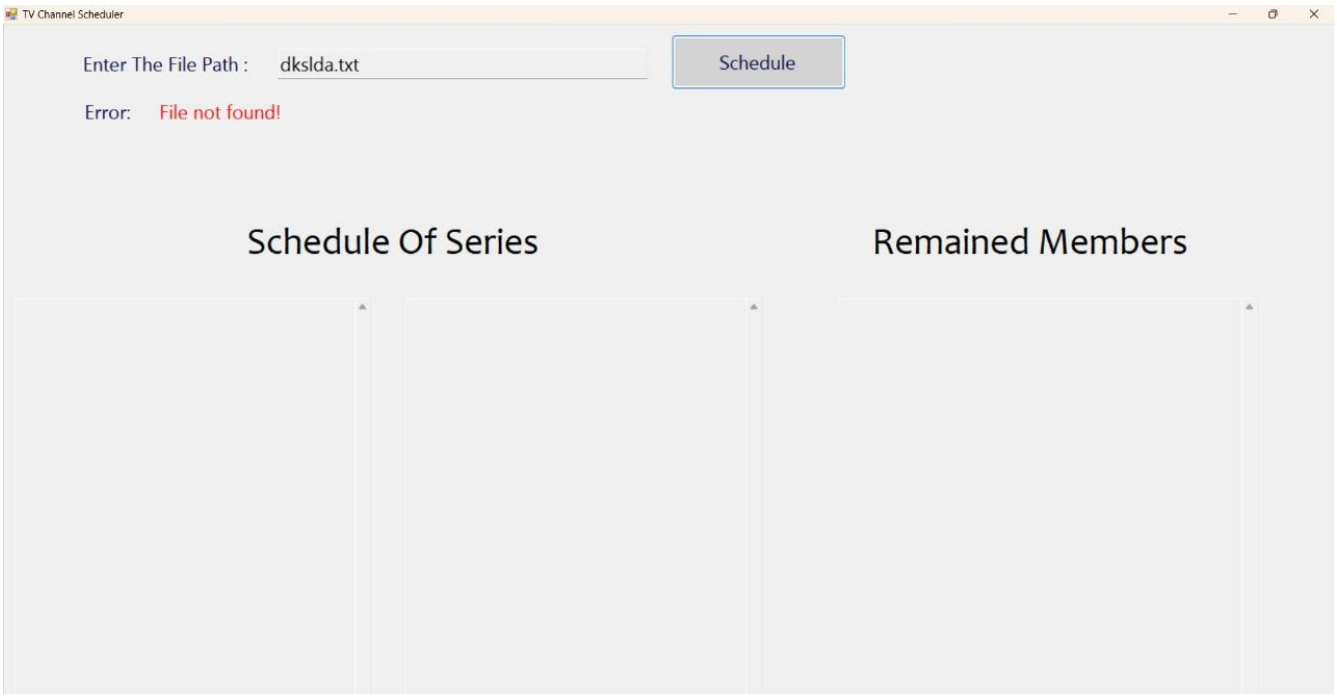
GUI based output:

Interface:



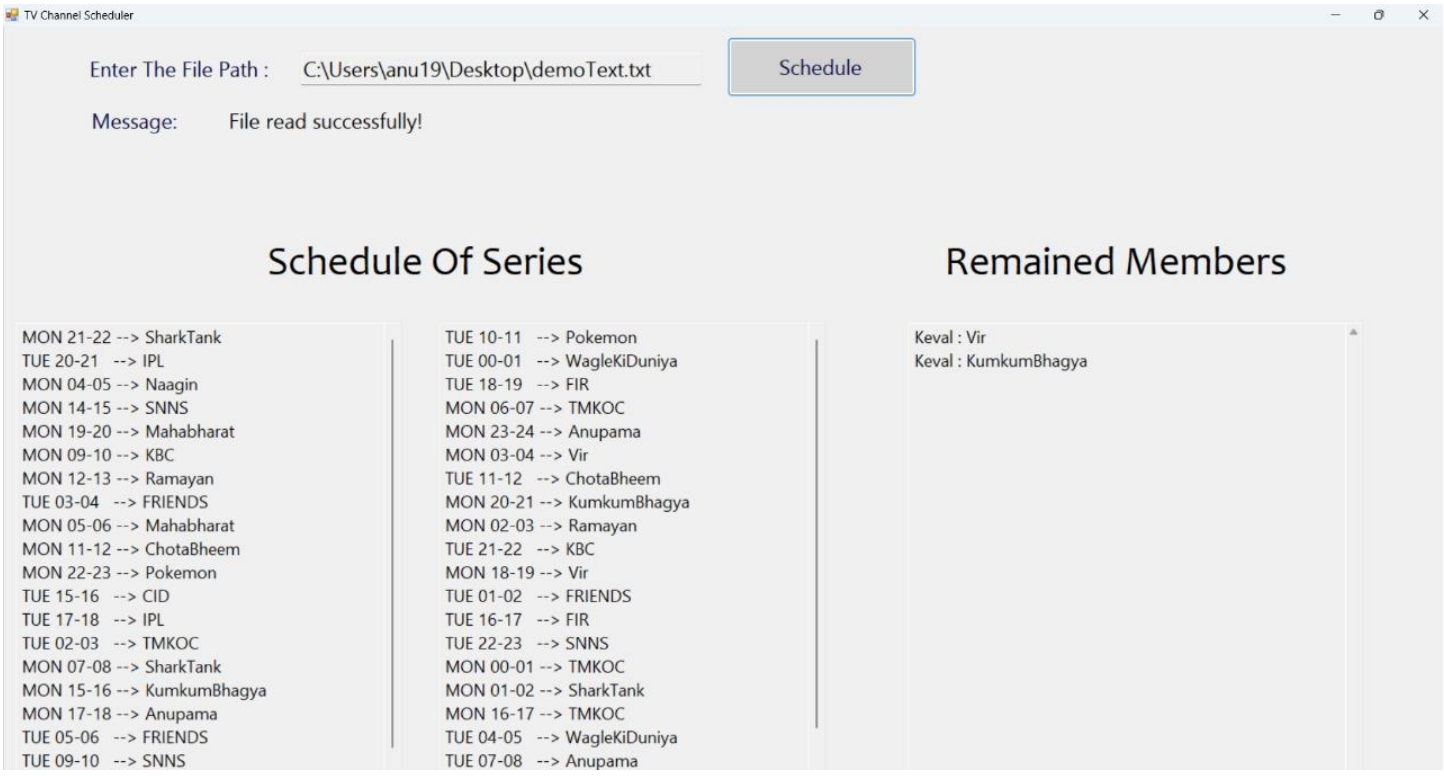
After entering a file path and push the schedule button.

If we enter wrong path it gives error –

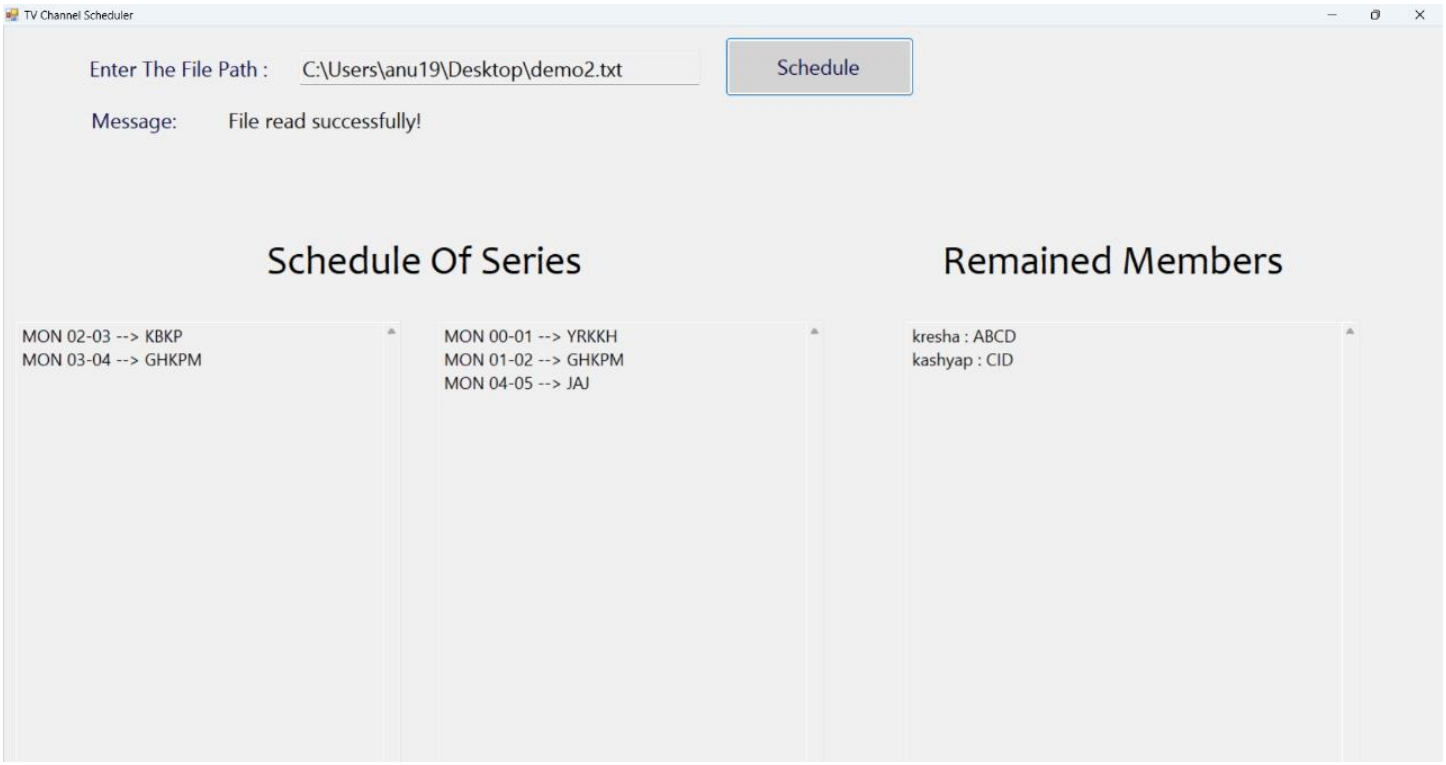


If we enter our input file path.

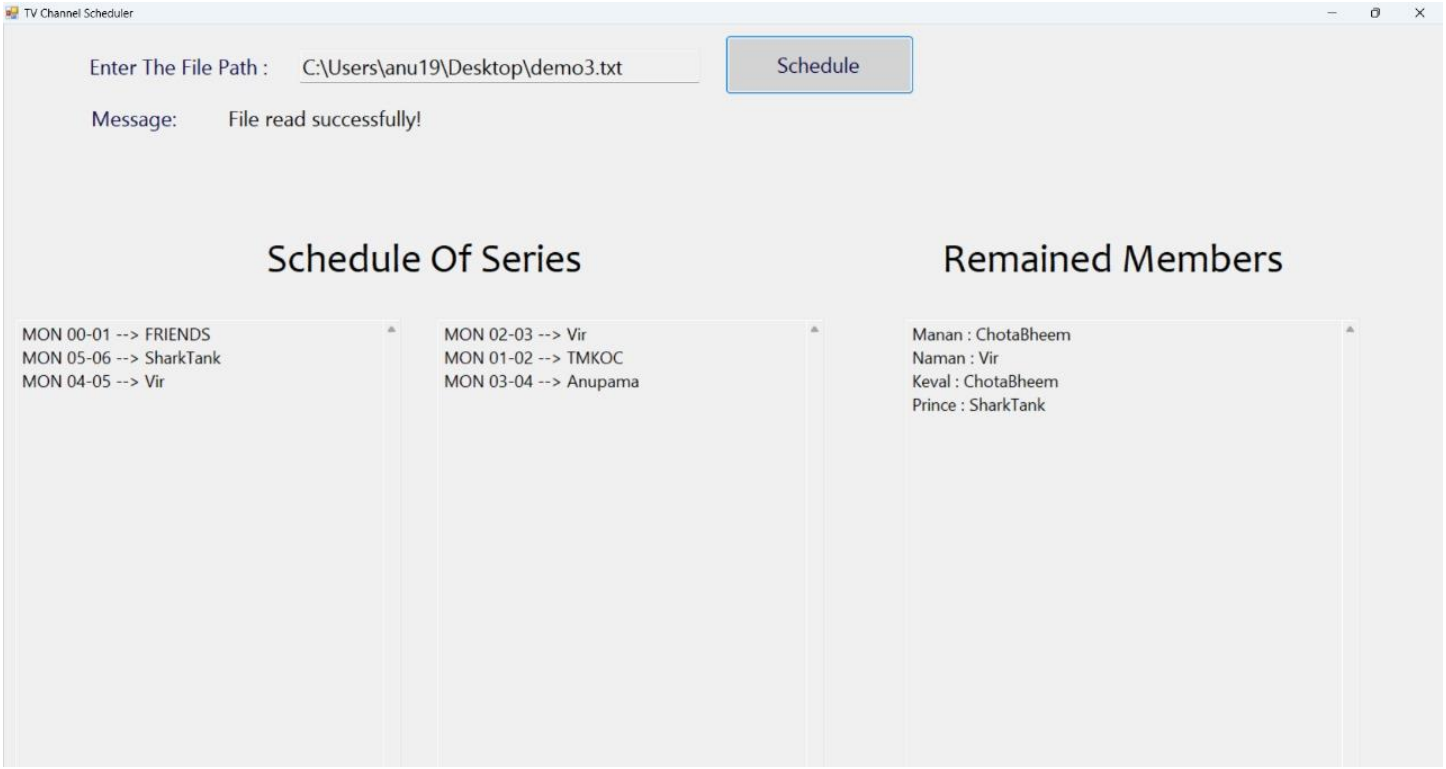
DemoText.txt



Demo2.txt



Demo3.cpp



...