Striver’s SDE Sheet

Day 7: Greedy Algorithm

N meetings in one room

There is one meeting room in a firm. There are N meetings in the form of (start[i], end[i]) where start[i] is start time of meeting i and end[i] is finish time of meeting i.
What is the maximum number of meetings that can be accommodated in the meeting room when only one meeting can be held in the meeting room at a particular time?

Note: Start time of one chosen meeting can't be equal to the end time of the other chosen meeting.

Ans:

Observation and approach : process and trick

It’s a standard greedy question in which we try different way like choose smallest time or choose with accessing starting time but all fails

And the correct approach will be choose with smallest finish time

Code: O(N) and Space O(n)

```
int maxMeetings(int start[], int end[], int n)
    {
        vector<pair<int,int>>temp;
        for(int i=0;i<n;i++)
        {
            temp.push_back({end[i],start[i]});
        }
        sort(temp.begin(),temp.end());
        int count=0;
        int starting;
        for(int i=0;i<n;i++)
        {
            if(i==0)
            {
                count++;
                starting=temp[i].first;
            }
            else if(temp[i].second>starting)
            {
                count++;
                starting=temp[i].first;
            }
```

```
        }

        return count ;
    }
```

variations :

1) Came under standard type of questions of ranges

Minimum Platforms

Given arrival and departure times of all trains that reach a railway station. Find the minimum number of platforms required for the railway station so that no train is kept waiting.
Consider that all the trains arrive on the same day and leave on the same day. Arrival and departure time can never be the same for a train but we can have arrival time of one train equal to departure time of the other. At any given instance of time, same platform can not be used for both departure of a train and arrival of another train. In such cases, we need different platforms.

Ans:

In this question just have to compare leaving time and arrival time if smaller do nothing if greater then ctn++;

Code: O(N)

```
int findPlatform(int arr[], int dep[], int n)
    {


       int ans=0,count=0;
    sort(arr,arr+n);
    sort(dep,dep+n);
    int i=0,j=0;
    while(i<n&&j<n)
    {
        if(arr[i]<=dep[j])
        {
           count++;
           i++;


        }
        else
        {
           ans = max(count,ans);
```

```
            count--;
            j++;
        }
    }
    ans = max(count,ans);
    return ans;
      // Your code here
    }
```

Variation:

1) Same type of category

Job Sequencing Problem

Given a set of N jobs where each job_i has a deadline and profit associated with it.

Each job takes 1 unit of time to complete and only one job can be scheduled at a time. We earn the profit associated with job if and only if the job is completed by its deadline.

Find the number of jobs done and the maximum profit.

Note: Jobs will be given in the form (Job_{id}, Deadline, Profit) associated with that Job.

Ans:

Just have to sort in descending order according to profit which we get and always complete it on last day od deadline

To achieve this we maintain one array instantly fill with -1 and if can do that job in given time then make it 1 and at the end just find the sum

Code:

```
 static bool comp(Job &a ,Job &b){
        return a.profit>b.profit;
    }
    vector<int> JobScheduling(Job arr[], int n)
    {
        // your code here
        sort(arr,arr+n,comp);
        int profit=0,c=0;
        vector<int> vect;
        vector<bool> vis(n ,false);
        for(int i=0;i<n;i++){

            int d=arr[i].dead;
            for(int j=min(n,d)-1;j>=0;j--){
                if(vis[j]==false)
                {
                    vis[j]=true;
                    profit+=arr[i].profit;
                    c++;

                    break;
                }
            }
```

```
        }
        vect.push_back(c);
        vect.push_back(profit);
        return vect;

    }
```

variation:

same type with little bit variation

Fractional Knapsack

Given *weights* and *values* of N items, we need to put these items in a knapsack of capacity W to get the *maximum* total value in the knapsack.
Note: Unlike 0/1 knapsack, you are allowed to break the item.

Ans:

Just have to put item with greater value by weight

Code:

```
static bool cmp(Item a,Item b)
    {
        return ((a.value/a.weight)>(b.value/b.weight));
    }
    double fractionalKnapsack(int W, Item arr[], int n)
    {
      int totalValue = 0.0;
    sort(arr, arr + n, cmp);
    int i = 0;
    while (W > 0 && i < n)
    {
        if (arr[i].weight <= W)
        {
            totalValue += arr[i].value;
            W -= arr[i].weight;
        }
        else
        {
            totalValue += (W * 1.0 / arr[i].weight) * arr[i].value;
            break;
        }
        i++;
    }
    return (double)totalValue;
    }
```

Variation:

A variation that is solved by dp

Find minimum number of coins that make a given value

Given a value V, if we want to make a change for V cents, and we have an infinite supply of each of C = { C1, C2, .., Cm} valued coins, what is the minimum number of coins to make the change? If it's not possible to make a change, print -1.

Ans:

Greedy approach: sort in increasing order and take coin one by if condition occur then yes else no

Dp approach: try all possibility

Code:

```
int minCoins(int coins[], int m, int V)
{
   // base case
   if (V == 0) return 0;

   // Initialize result
   int res = INT_MAX;

   // Try every coin that has smaller value than V
   for (int i=0; i<m; i++)
   {
     if (coins[i] <= V)
     {
         int sub_res = minCoins(coins, m, V-coins[i]);

         // Check for INT_MAX to avoid overflow and see if
         // result can minimized
         if (sub_res != INT_MAX && sub_res + 1 < res)
            res = sub_res + 1;
     }
   }
   return res;
```

```
}

// Driver program to test above function
int main()
{
    int coins[] =  {9, 6, 5, 1};
    int m = sizeof(coins)/sizeof(coins[0]);
    int V = 11;
    cout << "Minimum coins required is "
         << minCoins(coins, m, V);
    return 0;
}
```

Variation:

A variation that is solved with help of dp