Day 3 Jul 11, 2022

From leetcode

Interval based questions

1. Merge Intervals - LeetCode
2. Insert Interval - LeetCode
3. My Calendar I - LeetCode
4. My Calendar II - LeetCode
5. My Calendar III - LeetCode
6. Car Pooling - LeetCode
7. Employee Free Time
8. Minimum Number of Arrows to Burst Balloons - LeetCode
9. Non-overlapping Intervals - LeetCode
10. Find Right Interval - LeetCode

Merge interval

→ Input [1,3], [2,6], [8,10], [15,18]

Algo

1. Sort the list w.r.t first value

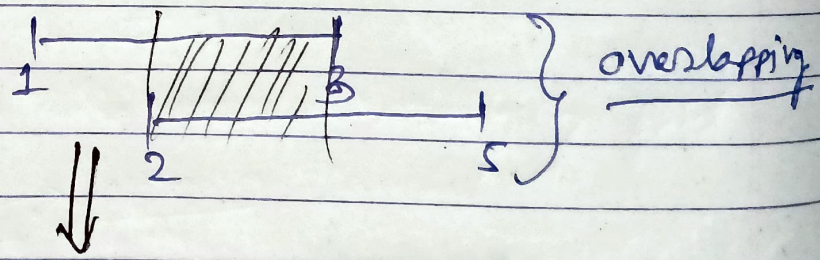
→ [1,3], [2,5], [8,10], [15,18]

Result vector

result.push([1,3]) ← Starting pair.

Now on running loop you compare.

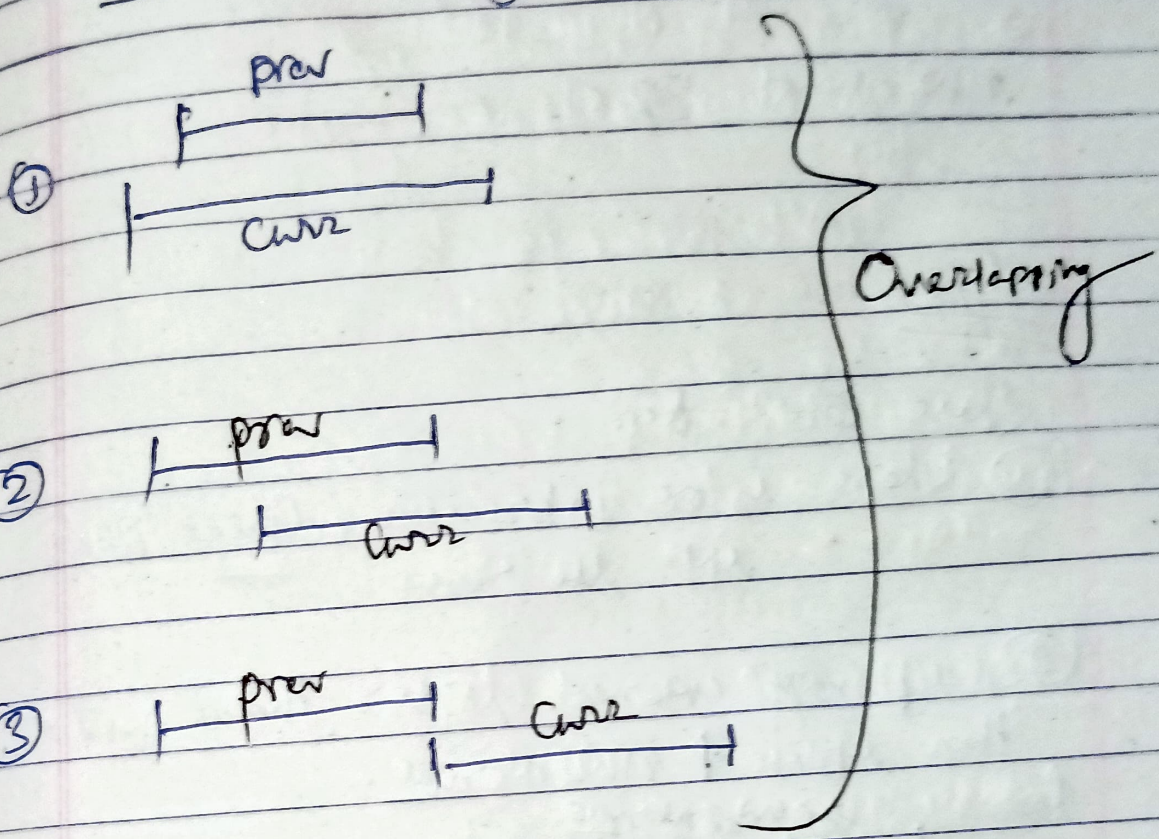
prev = [1,3]
curr = [2,5]



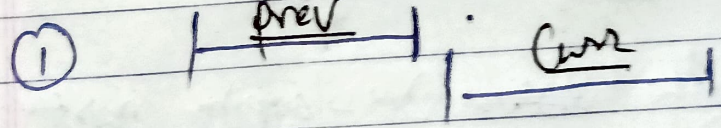
final output = min(starting of both) = 1
max(ending of both) = 5

= [1,5]

Now Condn of overlapping



NOT Overlapping




```
if (prev[end] < curr[starting])
{
    no overlapping
}
```

```cpp
/*
Given an array of intervals where intervals[i] = [starti, endi], merge all
overlapping intervals, and return an array of the non-overlapping intervals that
cover all the intervals in the input.
*/
#include<bits/stdc++.h>
using namespace std;

class Solution
{
public:
    vector<vector<int>> merge(vector<vector<int>> &intervals)
    {

        vector<vector<int>> result;
        sort(intervals.begin(), intervals.end());

        int n = intervals.size();

        result.push_back(intervals[0]);
        vector<int> prev;

        for (int i = 1; i < n; i++)
        {
            prev = result.back();

            if (prev[1] < intervals[i][0])
            {
                result.push_back(intervals[i]);
            }
            else
            {
                result.pop_back();
                prev[0] = min(prev[0], intervals[i][0]);
                prev[1] = max(prev[1], intervals[i][1]);
                result.push_back(prev);
            }
        }

        return result;
    }
};

int main()
{
    Solution s;
    vector<vector<int>> intervals = {{1, 3}, {2, 6}, {8, 10}, {15, 18}};
    vector<vector<int>> result = s.merge(intervals);
    for (auto i : result)
    {
        cout << i[0] << " " << i[1] << endl;
    }
    return 0;
}
```

Insert Interval

→ Intervals = [[1,3],[6,9]]

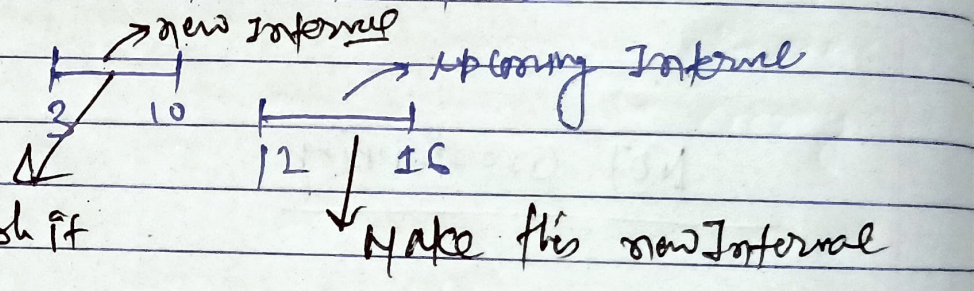
newInterval = [2,5]

~~Algo~~

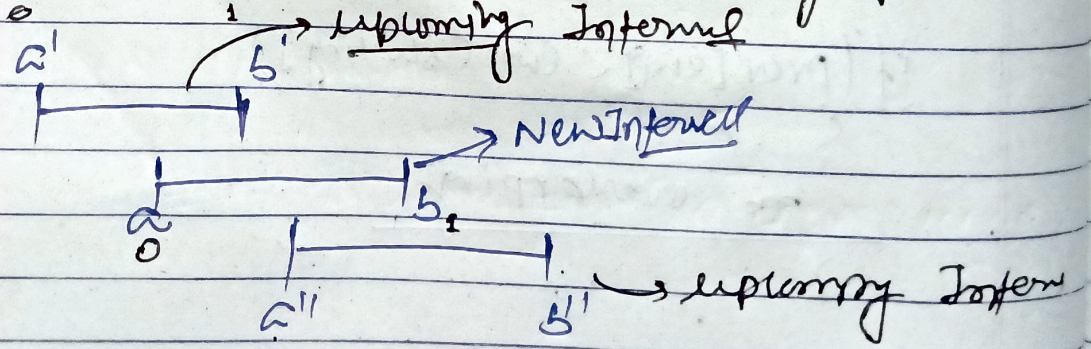
There Could be 4 cases

① There is no intersection, direct push into the result

② upcoming interval larger than newInterval then update newInterval.



③ Handle the ~~Inter~~ Overlapping part



if (b' >= a or a'' <= b')

?

```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution
{
public:
    vector<vector<int>> insert(vector<vector<int>> &intervals, vector<int> &newInterval)
    {
        vector<vector<int>> result;
// 1 3   6 9   or new interval 2 5
        for (size_t i = 0; i < intervals.size(); i++)
        {
            //  the new interval is after the range of other interval, so we can leave the current interval baecause the new one does not overlap with it
            if (intervals[i][1] < newInterval[0])
            {
                result.push_back(intervals[i]);
            }
            // the new interval's range is before the other, so we can add the new interval and update it to the current one
            else if (intervals[i][0] > newInterval[1])
            {
                result.push_back(newInterval);
                newInterval = intervals[i];
            }
            // the new interval is in the range of the other interval, we have an overlap, so we must choose the min for start and max for end of interval
            else if (intervals[i][1] >= newInterval[0] || intervals[i][0] <= newInterval[1])
            {
                newInterval[0] = min(intervals[i][0], newInterval[0]);
                newInterval[1] = max(newInterval[1], intervals[i][1]);
            }
        }
/*
Input: intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]], newInterval = [4,8]
Output: [[1,2],[3,10],[12,16]]
*/
        result.push_back(newInterval);
        return result;
    }
};

int main()
{
    Solution s;
    vector<vector<int>> intervals = {{1, 3}, {6, 9}};
    vector<int> newInterval = {2, 5};
    vector<vector<int>> result = s.insert(intervals, newInterval);
    for (auto i : result)
    {
        cout << i[0] << " " << i[1] << endl;
    }
    return 0;
}
```

Find right interval

front = [[1,12],[2,9],[3,10],[13,14],[15,16],[16,17]]

NOTE:- Map also uses the lower bound property likewise vector.

= m.lower_bound(interval[i][1])

map<int, int> store.

store[1] = 0
store[2] = 1
store[3] = 2
store[13] = 3
store[15] = 4
store[16] = 5

mapping First Number with index.

→ The lower bound of 12 in this store map is 13. Then there index is 3.

Similarly

9 Lower bound 13 then index 3
10 lower bound 13 ——— 3
:
:
:

This is the easiest way

lower Bound

↳ इसमें वो Num मिल जाए then वही number ~~return~~ index return होगा।

Otherwise just ~~less~~ बड़ा वाला का index return होगा.

0 1 2 3
10, 20, 30, 40,

→ lower Bound for 35 at index 3.

Upper Bound

return just Greater element otherwise end pointer from the array

* Bound in Map

= mp.lower_bound(num)

* Bound in vector

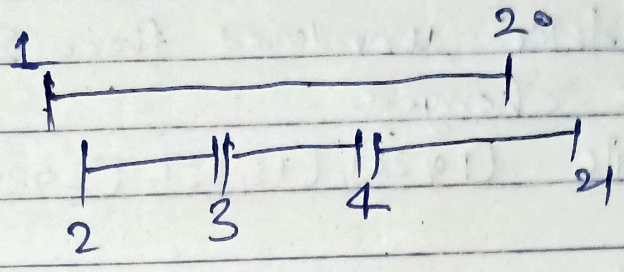
= ~~vctl~~ lower_bound(vct.begin(), vct.end(), num)

```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution
{
public:
    vector<int> findRightInterval(vector<vector<int>> &intervals)
    {
        vector<int> ans(intervals.size());
        map<int, int> m;
        for (int i = 0; i < intervals.size(); i++)
            m[intervals[i][0]] = i;
        for (int i = 0; i < intervals.size(); i++)
            ans[i] = m.lower_bound(intervals[i][1]) != end(m) ? m.lower_bound(intervals[i][1])->second : -1;
        return ans;
    }
};

int main()
{
    Solution s;
    vector<vector<int>> intervals = {{1, 4}, {2, 3}};
    vector<int> result = s.findRightInterval(intervals);
    for (auto i : result)
    {
        cout << i << endl;
    }
    return 0;
}
```

Non overlapping Intervals

Here we are trying to remove intervals.

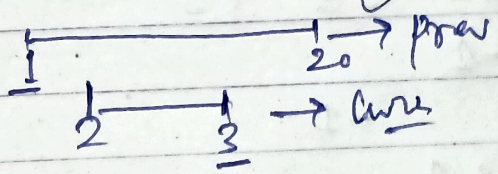
for example [1,20], [2,3], [3,4], [2,21]



prev = 0 (index)

curr = i (index)

check the condn of overlapping.



if(prev[0] < curr[1])

{ res++;

Remove the larger part i.e whose value is maximum

if(curr[i][1] < prev[1])

prev = i;

(removing prev interval (larger))

```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution
{
public:
    int eraseOverlapIntervals(vector<vector<int>> &interval)
    {
        /*
        ----------------- (larger end value)
        -----|-----|----|-------(smaller end value)
        */

        //         sorting by first value
        sort(interval.begin(), interval.end());
        int n = interval.size();
        int prev = 0;

        int result = 0;

        for (int i = 1; i < n; i++)
        {
            //             now condition of non orevlapping
            if (interval[i][0] >= interval[prev][1])
            {
                prev = i;
            }
            else
            {
                result++;
                //                 Here i am removing the larger one keep the smaller one
                if (interval[i][1] < interval[prev][1])
                {
                    prev = i;
                }
            }
        }
        return result;
    }
};

int main()
{
    Solution s;
    vector<vector<int>> interval = {{1, 2}, {2, 3}, {3, 4}, {1, 3}};
    int result = s.eraseOverlapIntervals(interval);
    cout << result << endl;
    return 0;
}
```

Minimum Arrows to burst the ballon

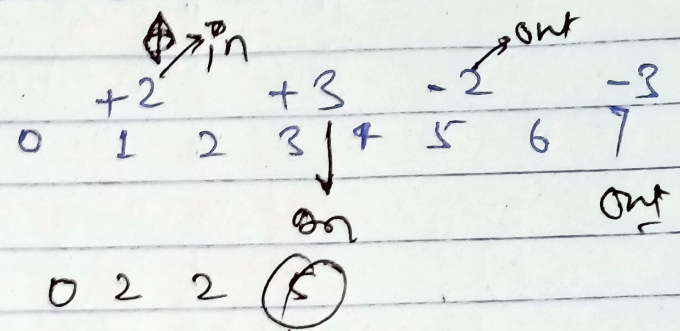
```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution
{
public:
    int findMinArrowShots(vector<vector<int>> &interval)
    {
        sort(interval.begin(), interval.end());
        int n = interval.size();
        int prev = 0;

        int result = 0;

        for (int i = 1; i < n; i++)
        {
            //             now condition of non orevlapping
            if (interval[i][0] > interval[prev][1])
            {
                prev = i;
            }
            else
            {
                result++;
                //                 Here i am removing the larger one keep the smaller one
                if (interval[i][1] < interval[prev][1])
                {
                    prev = i;
                }
            }
        }
        return n - result;
    }
};


int main()
{
    Solution s;
    vector<vector<int>> interval = {{10, 16}, {2, 8}, {1, 6}, {7, 12}};
    int result = s.findMinArrowShots(interval);
    cout << result << endl;
    return 0;
}
```

overall approach is same except equal to sign and here you need to return remaining balloon

Car Pooling

→ like Calendar problem, here we can face array because constraint is less.

[2, 1, 5] [3, 3, 7] cap = 4



false 0 2 2 5

greater than cap then return false.

```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution
{
public:
    bool carPooling(vector<vector<int>> &trips, int capacity)
    {
        vector<int> cache(1001, 0);

        int n = trips.size();

        for (int i = 0; i < n; i++)
        {
            int no = trips[i][0];
            int start = trips[i][1];
            int end = trips[i][2];
            cache[start] += no;
            cache[end] -= no;
        }

        int sum = 0;
        for (int i = 0; i <= 1000; i++)
        {
            sum += cache[i];
            if (sum > capacity)
            {
                return false;
            }
        }
        return true;
    }
};

int main()
{
    Solution s;
    vector<vector<int>> trips = {{2,1,5},{3,3,7}};
    int capacity = 4;
    bool result = s.carPooling(trips, capacity);
    cout << result << endl;
    return 0;
}
```

My Calendar I, II, III all are same pattern

Since range of element is large then Cannot store array. better to ~~take~~ store map not sort unorder, if you will store unordered then sequence will be change

interval like [10, 20], [15, 25], [20, 30]

m[10] ++ m[25] --

m[15] ++ m[25] --

m[20] ++ m[30] --

10	15	20	25	30
+1		-1		
	+1		-1	
		+1		-1
1	2	0	-1	-1

2 is overlapping count.

```cpp
#include<bits/stdc++.h>
using namespace std;

class MyCalendarThree
{
public:
    map<int, int> events;
    MyCalendarThree()
    {
    }

    int book(int start, int end)
    {

        events[start]++;
        events[end]--;

        int maxbooking = 0;
        int sum = 0;
        for (auto [num, fre] : events)
        {
            sum += fre;
            maxbooking = max(maxbooking, sum);
        }
        return maxbooking;
    }
};

int main()
{
    MyCalendarThree s;
    int result = s.book(10, 20);
    cout << result << endl;
    return 0;
}
```

My calender I II III uses the same approach

```cpp
#include<bits/stdc++.h>
using namespace std;

class MyCalendarTwo
{
public:
    map<int, int> events;
    MyCalendarTwo()
    {
    }

    bool book(int start, int end)
    {
        events[start]++;
        events[end]--;
        int sum = 0;
        for (auto [num, fre] : events)
        {
            sum += fre;
            if (sum == 3)
            {
                events[start]--;
                events[end]++;
                return false;
            }
        }

        return true;
    }
};

int main()
{
    MyCalendarTwo s;
    bool result = s.book(10, 20);
    cout << result << endl;
    return 0;
}
```

```cpp
#include<bits/stdc++.h>
using namespace std;

class MyCalendar
{
public:
    map<int, int> events;
    MyCalendar()
    {
    }
    bool book(int start, int end)
    {
        /*  auto next = events.upper_bound(start);
          if(next != events.end() &&   (*next).second < end)return false;
          events.insert({end,start});
          return true;
          */
        events[start]++;
        events[end]--;
        int sum = 0;
        for (auto [num, fre] : events)
        {
            sum += fre;
            if (sum == 2)
            {
                events[start]--;
                events[end]++;
                return false;
            }
        }

        return true;
    }
};


int main()
{
    MyCalendar s;
    bool result = s.book(10, 20);
    cout << result << endl;
    return 0;
}
```

```cpp
#include<bits/stdc++.h>
using namespace std;

class Solution {
public:
    bool makesquare(vector<int>& nums) {
        int sum = 0;
        int k=4;
        sum = accumulate(nums.begin(), nums.end(), sum);
        if (nums.size() < k || sum % k) return false;

        vector<int> visited(nums.size(), false);
        return backtrack(nums, visited, sum / k, 0, 0, k);
    }

    bool backtrack(vector<int>& nums,vector<int>& visited, int target, int curr_sum, int i, int k) {
        if (k == 0)
            return true;

        if (curr_sum == target)
            return backtrack(nums, visited, target, 0, 0, k-1);

        for (int j = i; j < nums.size(); j++) {
            if (visited[j] || curr_sum + nums[j] > target) continue;

            visited[j] = true;
            if (backtrack(nums, visited, target, curr_sum + nums[j], j+1, k)) return true;
            visited[j] = false;
        }

        return false;
    }
};

int main()
{
    Solution s;
    vector<int> nums = {1,1,2,2,2};
    bool result = s.makesquare(nums);
    cout << result << endl;
    return 0;
}
```

Leetcode daily question
And i forgot to draw recursive tree 🌲