Merge Overlapping Intervals

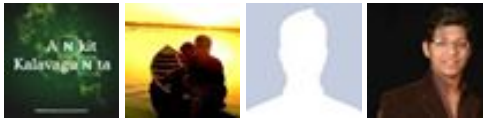
Given a set of time intervals in any order, merge all overlapping intervals into one and output the result which should have only mutually exclusive intervals. Let the intervals be represented as pairs of integers for simplicity.
For example, let the given set of intervals be {{1,3}, {2,4}, {5,7}, {6,8} }. The intervals {1,3} and {2,4} overlap with each other, so they should be merged and become {1, 4}. Similarly {5, 7} and {6, 8} should be merged and become {5, 8}

Write a function which produces the set of merged intervals for the given set of intervals.

A **simple approach** is to start from the first interval and compare it with all other intervals for overlapping, if it overlaps with any other interval, then remove the other interval from list and merge the other into the first interval. Repeat the same steps for remaining intervals after first. This approach cannot be implemented in better than O(n^2) time.

An **efficient approach** is to first sort the intervals according to starting time. Once we have the sorted intervals, we can combine all intervals in a linear traversal. The idea is, in sorted array of intervals, if interval[i] doesn't overlap with interval[i-1], then interval[i+1] cannot overlap with interval[i-1] because starting time of interval[i+1] must be greater than or equal to interval[i]. Following is the detailed step by step algorithm.

1. Sort the intervals based on increasing order of starting time.
2. Push the first interval on to a stack.
3. For each interval do the following
……..**a.** If the current interval does not overlap with the stack top, push it.
……..**b.** If the current interval overlaps with stack top and ending time of current interval is more than that of stack top, update stack top with the ending time of current interval.



4. At the end stack contains the merged intervals.

Below is a C++ implementation of the above approach.

```
// A C++ program for merging overlapping intervals
#include <iostream>
#include <vector>
#include <algorithm>
#include <stack>
using namespace std;

// An interval has start time and end time
struct Interval
{
    int start;
    int end;
};

// Compares two intervals according to their staring time.
// This is needed for sorting the intervals using library
// function std::sort(). See http://goo.gl/iGspV
bool compareInterval(Interval i1, Interval i2)
{  return (i1.start < i2.start)? true: false; }

// The main function that takes a set of intervals, merges
// overlapping intervals and prints the result
void mergeIntervals(vector<Interval>& intervals)
{
    // Test if the given set has at least one interval
    if (intervals.size() <= 0)
        return;

    // Create an empty stack of intervals
    stack<Interval> s;

    // sort the intervals based on start time
    sort(intervals.begin(), intervals.end(), compareInterval);

    // push the first interval to stack
    s.push(intervals[0]);

    // Start from the next interval and merge if necessary
    for (int i = 1 ; i < intervals.size(); i++)
    {
        // get interval from stack top
        Interval top = s.top();
```

Geometric Algorithms

Popular Posts

```
        // if current interval is not overlapping with stack top,
        // push it to the stack
        if (top.end < intervals[i].start)
        {
            s.push( intervals[i] );
        }
        // Otherwise update the ending time of top if ending of curren
        // interval is more
        else if (top.end < intervals[i].end)
        {
            top.end = intervals[i].end;
            s.pop();
            s.push(top);
        }
    }

    // Print contents of stack
    cout << "\n The Merged Intervals are: ";
    while (!s.empty())
    {
        Interval t = s.top();
        cout << "[" << t.start << "," << t.end << "]" << " ";
        s.pop();
    }

    return;
}

// Functions to run test cases
void TestCase1()
{
    // Create a set of intervals
    Interval intvls[] = { {6,8}, {1,9}, {2,4}, {4,7} };
    vector<Interval> intervals(intvls, intvls+4);

    // Merge overlapping inervals and print result
    mergeIntervals(intervals);
}
void TestCase2()
{
    // Create a set of intervals
    Interval intvls[] = { {6,8},{1,3},{2,4},{4,7} };
    vector<Interval> intervals(intvls, intvls+4);

    // Merge overlapping inervals and print result
    mergeIntervals(intervals);
}
```

```
void TestCase3()
{
    // Create a set of intervals
    Interval intvls[] = { {1,3},{7,9},{4,6},{10,13} };
    vector<Interval> intervals(intvls, intvls+4);

    // Merge overlapping inervals and print result
    mergeIntervals(intervals);
}

// Driver program to test above functions
int main()
{
    TestCase1();
    TestCase2();
    TestCase3();
    return 0;
}
```

Output:

```
The Merged Intervals are: [1,9]
The Merged Intervals are: [1,8]
The Merged Intervals are: [10,13] [7,9] [4,6] [1,3]
```

Time complexity of the method is O(nLogn) which is for sorting. Once the array of intervals is sorted, merging takes linear time.

This article is compiled by Ravi Chandra Enaganti. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recent Comments

loop. We can do it...

Find subarray with given sum · 1 hour ago

Related Tpoics:

- Remove minimum elements from either side such that 2*min becomes more than max
- Divide and Conquer | Set 6 (Search in a Row-wise and Column-wise Sorted 2D Array)
- Bucket Sort
- Kth smallest element in a row-wise and column-wise sorted 2D array | Set 1
- Find the number of zeroes
- Find if there is a subarray with 0 sum
- Divide and Conquer | Set 5 (Strassen's Matrix Multiplication)
- Count all possible groups of size 2 or 3 that have sum as multiple of 3

Writing code in comment? Please use **ideone.com** and share the link here.

32 Comments GeeksforGeeks

Sort by Newest

Nilesh Kumar • 3 days ago

Hi Ravi,

Instead of using vector and stack, you can use a std::list . Internally, this conta
delete in between from a list rather than a vector. So, no stack is required, if yo

If interested, you can view the same here : http://ideone.com/m11lDb

Regards,
Nilesh

 • Reply • Share ›

Sreek • a month ago

It has to be
if (top.end + 1 < intervals[i].start)

to support the use case [1,2] [3,4] to give the output [1,4].

 • Reply • Share ›

anonym2 • 5 months ago

Another solution: A min heap of the pairs with respect to the start element can
gives O(nlogn) solution.

2  • Reply • Share ›

Deepak Shrivastava • 7 months ago

I wrote a java implementation of the problem. Please find it below...

```
import java.util.Arrays;

import java.util.Comparator;
```

```
public class Solution {

public static void main(String[] args) {

Integer arr[][] = { { 1, 3 }, { 6, 8 }, { 2, 4 }, { 5, 7 } };

Arrays.sort(arr, new Comparator<integer[]>() {

@Override

public int compare(Integer[] o1, Integer[] o2) {

if (o1[0] > o2[0])

return 1;
```

see more

• Reply • Share ›

Sriharsha g.r.v • 7 months ago

can some one

explain me how is the complexity O(nlogn)..what is the worst
case complexity i.e when neither of intervals overlap? is the complexty
nlogn occuring because of sorting the intervals??

• Reply • Share ›

zzer ➚ Sriharsha g.r.v • 17 days ago

yeah, the O(NlogN) is based on sorting intervals.

• Reply • Share ›

Guest • 7 months ago

can some explain me how is the complexity O(nlogn)..what is the worst case
overlap? is the complexty nlogn occuring because of sorting the intervals??

• Reply • Share ›

Guest • 7 months ago

can some explain me how is the complexity O(nlogn)..what is the worst case overlap?

^ | v • Reply • Share ›

Ankit Chaudhary • 7 months ago

Variation of above problem: suppose intervals are coming one by one and u ha time when new interval comes.
Means at every instant u have to show merged intervals.

This can be implemented using doubly linked list.

1 ^ | v • Reply • Share ›

DRAGONWARRIOR • 8 months ago

IN BODY OF ELSE IF
SINCE END IS UPDATED FOR TOP ELEMENT(top() function return ref to top
WHY WE NEED POP STATEMENT

^ | v • Reply • Share ›

rakitic • 8 months ago

```
#include<iostream>

#include<conio.h>

#include<stack>

#include<vector>

#include<algorithm>
```

```
using namespace std;

struct interval{

       int start;

       int end;
```

see more

• Reply • Share ›

rakitic • 8 months ago

```
#include<iostream>

#include<conio.h>

#include<stack>

#include<vector>

#include<algorithm>

using namespace std;

struct interval{

int start;

int end;

};

stack<struct interval="">S;
```

see more

Guest • 8 months ago

this can be done in O(n lg n) by building a segment tree.

Neha Modi • 9 months ago

how about this....

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
void main()
{ int arr[4][2]={{6,8},{1,9},{2,4},{4,7}}, i,j, temp, temp1;.
for(i=0;i<4;i++).
{ for(j=i+1;j<4;j++).

{ if(arr[j][0]<arr[i][0]).

{ temp=arr[j][0];.
temp1=arr[j][1];.
arr[j][0]=arr[i][0];.
arr[j][1]=arr[i][1];.
arr[i][0]=temp;.
arr[i][1]=temp1;.

}.
```


Cracking The Code • a year ago

how about this.

start loop

if (top.end >= intervals[i].start) // 1,9, 2,4.

{.

top.end = top.end>intervals[i].end? top.end:intervals[i].end;

s.pop();.

s.push(top);.

}.

else if(top.end<intervals[i].start).

s.push(intervals[i]);.

see more

• Reply • Share ›

nikhil • a year ago

// assuming i have a sort function to sort the interval array

```
struct Interval{
int first;
int second;
};
void merge(struct Interval a[],int n){
int i=0, j;
sort(a,n);
while(i<n){
j=i;
```

```
while( a[j].second > a[j+1].first )
j++;
cout<<"["<<a[i].first<<","<<a[j].second<<"]"<<endl;
i=j+1;
}
}
```

Time complexity will be O(nlogn(sort)+n(merge)).
Space complexity O(1).

nikhil • a year ago

//assuming i have a sort function to sort the interval array
struct Interval{
int first;
int second;
};
void merge(struct Interval a[],int n){
int i=0,j;
sort(a,n);
while(i a[j+1].first)
j++;
cout<<"["<<a[i].first<<","<<a[j].second<<"]"<<endl;
i=j+1;
}
}

Time complexity will be O(nlogn(sort)+n(merge)).
Space complexity O(1).

Mani • a year ago

I tried to implement in java.

```
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;

public class MergeIntervals {

/**
* @param args
*/
public static void main(String[] args) {
// TODO Auto-generated method stub
int a[][] =new int[][]{ {2,3}, {7,9}, {4,6}, {1,13}};
//int a[][] =new int[][]{{6,8}, {1,9}, {2,4}, {4,7}};
//int a[][] =new int[][]{{1,3},{4,6},{5,10}};
//int a[][] =new int[][]{{1,4},{2,4},{5,5},{6,8}};
int b[][] =new int[4][];
```

see more

Bamftubb • a year ago

Or another shorter one:

```
 var sorted = ranges.OrderBy(r => r[0]).ToArray();
var result = new LinkedList<int[]>();
result.AddFirst(sorted[0]);
foreach (var range in sorted) {
```

```
    var top = result.Last.Value;
    if (range[0] <= top[1])
        top[1] = top[1] < range[1] ? range[1] : top[1];
    else
        result.AddLast(range);
}
return result.ToArray();
```


Bamftubb • a year ago

Here C# variant without using stack. It uses List for collecting ranges.

[sourcecode language="C#"]
class Program
{
static void Main(string[] args)
{
var given = new int[][] {
new[] { 0, 5 },
new[] { -1, 5 },
new[] { 6, 10 },
new[] { 3, 4 },
new[] { 15, 20 },
new[] { -10, 6 }
};
Console.WriteLine("Given: [{0}]", string.Join("], [", given.Select(r => string.Forr
Console.WriteLine("Result: [{0}]", string.Join("], [", Merge(given).Select(r => st
}

see more

Daddy • a year ago

Here is implementation in C :

```
#include<iostream>
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
using namespace std;

struct interval{
int start;
int end;
};

struct stack{
struct interval* arr;
int top;
};

struct stack* createstack(int n)
{
```

see more

Reply • Share ›

Sambasiva • a year ago

http://effprog.wordpress.com/2...

Reply • Share ›

Anon • a year ago

Here is another implementation in C++.

```
/* Paste your code here (You may delete these lines if not writing c
#include <iostream>
```

```
#include <algorithm>
#include <vector>
#include <stack>
using namespace std ;

void print ( vector <pair<int,int> > v )
{
    vector <pair<int,int> > :: iterator it ;
    for ( it = v.begin() ; it != v.end() ; ++it )
        cout << (*it).first << " " << (*it).second << endl ;
    return ;
}

int main()
```

see more

• Reply • Share ›

Gupt • a year ago

An O(n) in time and O(n) in memory approach could be this, where n is the hig

```
 create an array of size n, say sets
fill it with -1

current setnb = 1;
for (each interval)
{
    if (set[interval.start == -1])
         currentmarker = setnb++;
    else
         currentmarker = set[interval.start]
```

```
     for (i = interval.start to interval.end)
      {
           sets[i] = currentmarker;
      }
}

now the sets array contains different interval, we dont need any sort:
just extract those intervals by traversing in array.
```


vishal • a year ago

Please correct me if i'm wrong but, what will be the output for {{6,8},{1,3},{2,4} {1,9}.....but as per current code it will be {1,8}.

GeeksforGeeks ➦ vishal • a year ago

@vishal: Thanks for pointing this out. The post is updated to handle thi

aasshishh • a year ago

The above code has a bug.
I changed the inputs.
Test Case 1: {6,8}, {1,9}, {2,4}, {4,7}
Test Case 2: {2,3}, {7,9}, {4,6}, {1,13}

Output were:
Test case 1 : [1,8]

Outputs should be :
Test case 1 : [1,9]
Test case 2 : [1,13]

The code should be as follows

```
if (top.end < intervals[i].start)
     {
         s.push( intervals[i] );
     }
```

see more

• Reply • Share ›

GeeksforGeeks ➚ aasshishh • a year ago

@aasshishh: Thanks for pointing this out and suggesting the resolutior

• Reply • Share ›

aasshishh • a year ago

There is a bug in the program.

I changed the inputs.
Test Case 1: {6,8}, {1,9}, {2,4}, {4,7}
Test Case 2: {2,3}, {7,9}, {4,6}, {1,13}

Output were:
Test case 1 : [1,8]
Test case 2 : [7,9] [4,6] [1,3]

Outputs should be :
Test case 1 : [1,9]
Test case 2 : [1,13]

The following code has bug.

```
if (top.end < intervals[i].start)
      {
          s.push( intervals[i] );
```

see more

Reply • Share ›

Ashwini Chaudhary • a year ago

Implementation in python:

[sourcecode language="Python"]
from stack import Stack
from operator import itemgetter
lis=[(1,3),(2,4),(5,7),(6,8)]
lis.sort(key=itemgetter(0))
s=Stack()
s.push(lis[0])
for x,y in lis[1:]:
topx,topy=s.top()
if topx<x<=topy:
s.pop()
s.push((topx,y))
else:
s.push((x,y))
print s.stack

Reply • Share ›

Ashwini Chaudhary ➚ Ashwini Chaudhary • a year ago

Edit: the push should be

```
s.push((min(x,topx),max(y,topy)))
```

Reply • Share ›

Somu • a year ago

Good work Ravi Chandra!

Reply • Share ›

Subscribe Add Disqus to your site