How to find the biggest common range?

Asked 9 years, 4 months ago Modified 9 years, 4 months ago Viewed 138 times

0

I have a list of hundreds of tuples like this:

```
list1 = [(epoch1, epoch2), (epoch3, epoch4), (epoch5, epoch6)]
```

every tuple contains the beginning and the end of the session in a form of an epoch.

What I want to find is the maximum number of sessions that occurred at the same time. So if `epoch1 = 16:00, epoch2=16:30, epoch3=16:15, epoch4=16:45, epoch5=18:00, epoch6=19:00`, the answer would be `2`, because from `16:15 to 16:30` there were two sessions active (the first one and the second one).

I thought that one way to do this is to create a new list containing all the dates like this:

```
list2 = [epoch1, epoch2, epoch3, epoch4, epoch5, epoch6]
```

Then iterate through every pair of elements in `list2` and check for how many of `list1` tuples contains them in its session borders. But I suppose this solution is very slow. Does anyone have other ideas?

python **algorithm** **datetime** **intervals**

Share
Improve this question
Follow

edited Aug 21, 2015 at 8:57
jfs
413k ● 205 ● 1k ● 1.7k

asked Aug 21, 2015 at 0:59
human
755 ● 2 ● 8 ● 17

why `16:15 to 16:30` ? – Padraic Cunningham Aug 21, 2015 at 1:08

because session 1 lasts from 16:00 to 16:30, and session 2 lasts from 16:15 to 16:45, so between 16:15 and 16:30 both sessions are active. – human Aug 21, 2015 at 1:12

sort the list and you can do it in `n log n` time – Padraic Cunningham Aug 21, 2015 at 1:32

3 Answers

Sorted by: Highest score (default)

Two O(n log n) ways

3

Setup:

```
sessions = [('16:00', '16:30'), ('16:15', '16:45'), ('18:00', '19:00')]
```

First way: Turn the starts and ends into events (epoch+change pairs), go through them in ascending order, and update `active` and `maxactive` appropriately.

```
events = (event
          for start, end in sessions
          for event in ((start, 1), (end, -1)))
active = maxactive = 0
for _, change in sorted(events):
    active += change
    maxactive = max(maxactive, active)
print(maxactive)
```

Second way: Sort starts and ends independently, then go through them "in parallel", updating the number of needed slots and how many are currently free.

```
starts, ends = map(sorted, zip(*sessions))
slots = free = e = 0
for start in starts:
    while ends[e] <= start:
        free += 1
        e += 1
    if free:
        free -= 1
    else:
        slots += 1
print(slots)
```

Share

Improve this answer

Follow

edited Aug 21, 2015 at 2:06

answered Aug 21, 2015 at 1:47

Stefan Pochmann

28.3k ● 8 ● 46 ● 111

The first way is very direct and elegant. On reflection, I was remembering the algorithm for coloring an interval graph, but all the OP requires is the chromatic number, which can be calculated more directly as you suggest. – saulspatz Aug 21, 2015 at 2:18

Ah, ok. Though I think the coloring can also be done pretty much this directly, at least with my second way. Instead of `free` being a number, make it a set (of colors). And give the starts and ends a reference to their intervals. Then instead of `slots += 1`, create a new color and assign it to the starting interval. And instead of `free -= 1`, pop a color from the set and assign it to the starting interval. And instead of `free += 1`, put the color of the ending interval into the set. – Stefan Pochmann Aug 21, 2015 at 18:56

I'll need to look at this when I have more time than I do just at the moment. Of course, one can make the algorithm I gave O(n log n) by using a min heap, and it's very intuitive. BTW in your coloring algorithm, I assume slots is initialized to the empty set, correct? – saulspatz Aug 21, 2015 at 20:39

@saulspatz Sorry, I forgot that aspect. Yes, "slots" starts as the empty set, although I'd rename it "colors". So "colors" is the set of all colors used so far (its final size is the answer for the OPs question), and "free" is the set of colors currently free to be reused. And when creating a new color, put it in "colors". – Stefan Pochmann Aug 21, 2015 at 22:02

Conceptually, it's the same algorithm as the one I gave, just a slightly different implementation. There's a slight problem with "assign the color to the starting interval" You just have the starting times, and there's no obvious way to relate them to the starting intervals. Since two intervals can start at the same time, you can't assign colors to the starting times, then later figure out which intervals start at which times. This is just an implementation detail, of course. – saulspatz Aug 23, 2015 at 17:40

1

This is a classical problem. Suppose that a hotel needs to schedule rooms for meetings whose start and ending times are given. How many rooms are required? Let me try to remember the solution.

First, sort the meetings in order of starting time. Now pretend for the moment that an unlimited number of rooms are available. Schedule the first meeting in room one. For the second meeting, if the meeting in room one is ended when meeting two starts, schedule it in room one, otherwise, schedule it in room two. For each successive meeting, look in all the meeting rooms. If one is vacant, schedule the meeting there. If not, add a new room.

To see that this works, note that the minimum number of rooms we need is the maximum number of simultaneous meetings, since each occurs in a different room. The algorithm provides a way of scheduling the meetings in exactly this many rooms, since we never add a new room unless all the existing rooms are occupied when a meeting ahs to start. `len(ending)` is the high-water mark for the number of simultaneous meetings.

To do this in python, we just need a list of ending times. Set `ending[0] = epoch2`. For each meeting, loop through the ending list. If you find one earlier than the starting time of the new meeting, change it to the ending the time of the new meeting. If not, append a new element to ending. At the end, `len(ending)` is the answer.

It might be worth while to sort the ending list every time it changes, because it clearly enough to inspect the earliest ending time.

FURTHER INFORMATION: The problem can be modeled as a graph, where the intervals are the vertices, and two vertices are adjacent if and only if they intersect. Now we can color the vertices with the colors representing rooms, and no two

adjacent vertices can have the same color. The minimum number of colors required is the answer. A graph of this type is called an interval graph, oddly enough. Interval graphs have many applications. https://en.wikipedia.org/wiki/Interval_graph Finding the chromatic number of a graph is NP-hard in general, but this algorithm shows that the chromatic number of an interval graph can be computed in polynomial time.

Share

Improve this answer

Follow

edited Aug 21, 2015 at 1:49

answered Aug 21, 2015 at 1:16

saulspatz

5,251 ● 5 ● 41 ● 49

Not mine but how would the `len(ending)` be the answer? – Padraic Cunningham Aug 21, 2015 at 1:23

I don't know who downvoted you, but I definitely think that you hit the spot here. This is the kind of algorithm that I was looking for. Thanks! – human Aug 21, 2015 at 1:47

1 @Padraic Cunningham I added an explanation of why it works. I should have done so at first. Thanks for pointing this out. – saulspatz Aug 21, 2015 at 1:50

Doesn't this take up to quadratic time, though? If all sessions overlap at the same time? Not sure whether and how much it helps to *"sort the ending list every time it changes"*, depends on how you do so. – Stefan Pochmann Aug 21, 2015 at 1:53

@StefanPochmann It might take up to quadratic time in the worst case with the implementation I suggested. I'm would assume one would use the python sort built-in, which uses timsort, whose performance is said to be "almost surrealistic" on nearly sorted lists. I was going to suggest using a minheap, but it seems like overkill unless there are a great many intervals. Probably even sorting isn't necessary in practice. The OP's list has "hundreds" of intervals. – saulspatz Aug 21, 2015 at 2:04

0

Well lets think about the easiest breakdown of what you are asking.
How many sessions are running at the same time?
How would you look at this as a human?

Well for the epochs to be at the same time either:

- One epoch would The beginning of the epoch would need to be come between the start of another epoch and before the end of it.
- Or the end of of an epoch would need to be come between the start of another epoch and before the end of it.

But when you think about these are the same thing, because for one to be true the other has to be true.

So this means you can check if the first element in each tuple is between the two elements of any other tuple:

```
count = 0
for tuple1 in list1:
    for tuple2 in list1:
         if tuple2[1] > tuple1[0] > tuple2[0]:
             count += 2 #this means two overlap
```

Share Improve this answer Follow

answered Aug 21, 2015 at 1:25

ThatGuyRussell
1,381 • 9 • 18