

Application Telespazio VEGA Deutschland as "Junior Java Engineer"

Test assessment : WY Space exercise

Cesar Augusto Guzman Alvarez

Source code: https://github.com/cguz/Interview_VEGA

1. Question to solve and Assumptions

1.1. Question to solve

The two question to solve are:

1. WY Space would like to take a text based schedule (detailed below) and use a program that can find the 30 minute period where the total downlink (all satellite passes) will be at its maximum.
2. Furthermore, they would like the program to determine if the ground station has the bandwidth to support this.

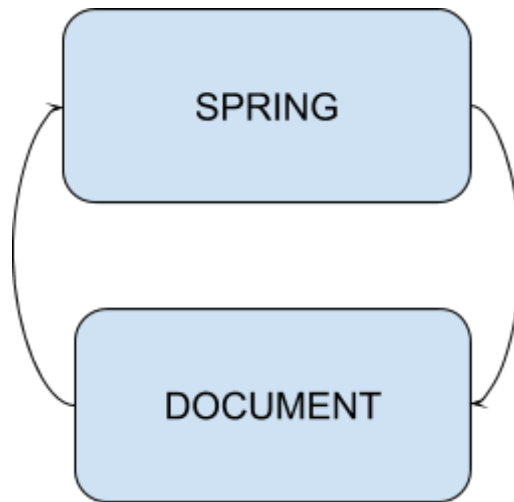
1.2. Assumptions

I assume the following:

- All the pass schedules are related to the same day.
- Column **bandwidth per 30 minute period** in the pass schedule file means that during the 30 minute the satellite consumes X bandwidth of the ground station from Earth.
- There can be more than one 30 minute period where the total downlink will be at its maximum, for instance, if they are equals.
- The time starts at 00:00, and it increases by 30 minutes. That is, the minutes in the start and end times can be only 00 or 30.

2. Methodology

I am going to follow the Agile development process. I will combine the development of the Spring with writing the present document.



I propose to develop the following three Spring. I am not sure if I am going to have the time to finish all the spring, but I am going to try it.

- Spring 1 : Develop a prototype in OOP that solve the problem
- Spring 2 : Implement the prototype in the framework Spring
- Spring 3 : Implement some GUI

3.Design and Development

3.1. Spring 1: Develop a prototype in OOP that solve the problem

Analysis of the problem

Figure 1 shows a sketch of the problem. In general, we have a ground station, and several satellites. The satellites can communicate to the ground station. Each time the satellite shows the ground station it communicates some information, occupying the bandwidth of the ground station.

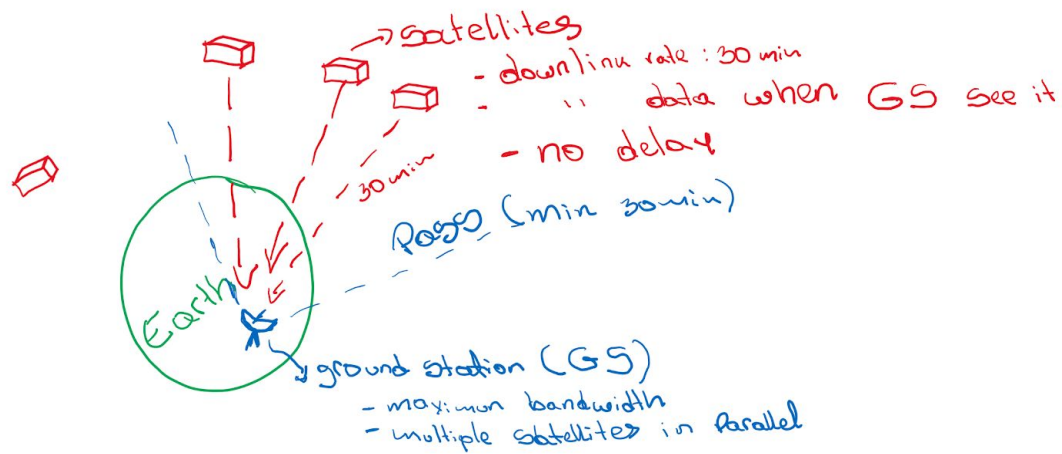


Figure 1. Sketch of the problem.

Satellite Name	bandwith per 30 minute preiod	Start time	End time	Duration
RedDwarf	2	0:00	1:30	1:30
RedDwarf	2	2:30	4:00	1:30
RedDwarf	2	5:00	6:30	1:30
RedDwarf	2	7:30	9:00	1:30
RedDwarf	2	10:00	11:30	1:30
RedDwarf	2	12:30	14:00	1:30
RedDwarf	2	15:00	16:30	1:30
RedDwarf	2	17:30	19:00	1:30
RedDwarf	2	20:00	21:30	1:30
RedDwarf	2	22:30	0:00	1:30
Narcissus	5	0:30	3:30	3:00
Narcissus	5	5:30	8:30	3:00
Narcissus	5	10:30	13:30	3:00
Narcissus	5	15:30	18:30	3:00
Narcissus	5	20:30	23:30	3:00
Nostromo	3	0:00	0:00	0:00
Sulaco	10	3:00	3:30	0:30
Sulaco	10	15:00	15:30	0:30
Rocinante	30	12:00	16:30	4:30
Moya	10	0:00	3:00	3:00
Moya	10	8:30	11:30	3:00
Moya	10	17:00	20:00	3:00

Odyssey	15	2:00	8:00	6:00
Odyssey	15	16:30	22:30	6:00
Enterprise	30	9:00	10:30	1:30
Normandy	2	12:00	15:00	3:00

Table 1. pass schedule

Table 1 describes the pass schedule file. In order to try to understand this table, I draw the points. Figure 2 shows the chart of points of the pass schedule file.

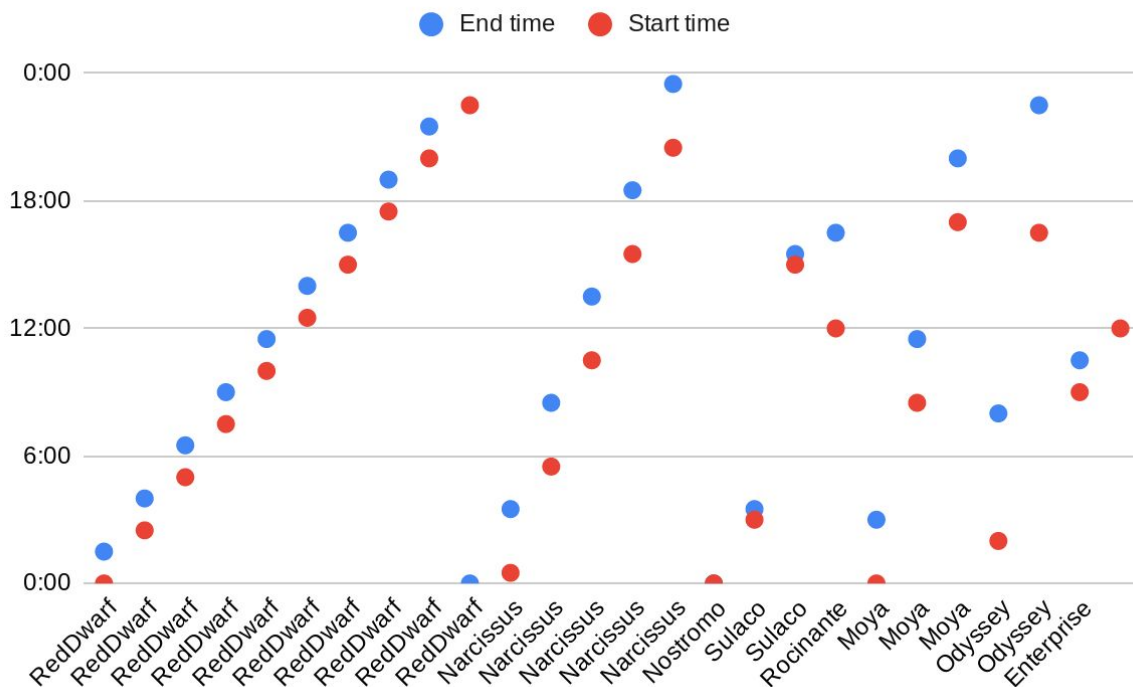


Figure 2. Chart of point of the pass schedule file

With the Figure it is not easy to understand the problem. One approach to understand better the problem is to draw the data in a timeline chart.

For this, I first sort the data by start time. See Table 2.

Satellite Name	bandwith per 30 minute preiod	Start time	End time	Duration
RedDwarf	2	0:00	1:30	1:30
Nostromo	3	0:00	0:00	0:00
Moya	10	0:00	3:00	3:00
Narcissus	5	0:30	3:30	3:00

Odyssey	15	2:00	8:00	6:00
RedDwarf	2	2:30	4:00	1:30
Sulaco	10	3:00	3:30	0:30
RedDwarf	2	5:00	6:30	1:30
Narcissus	5	5:30	8:30	3:00
RedDwarf	2	7:30	9:00	1:30
Moya	10	8:30	11:30	3:00
Enterprise	30	9:00	10:30	1:30
RedDwarf	2	10:00	11:30	1:30
Narcissus	5	10:30	13:30	3:00
Rocinante	30	12:00	16:30	4:30
Normandy	2	12:00	15:00	3:00
RedDwarf	2	12:30	14:00	1:30
RedDwarf	2	15:00	16:30	1:30
Sulaco	10	15:00	15:30	0:30
Narcissus	5	15:30	18:30	3:00
Odyssey	15	16:30	22:30	6:00
Moya	10	17:00	20:00	3:00
RedDwarf	2	17:30	19:00	1:30
RedDwarf	2	20:00	21:30	1:30
Narcissus	5	20:30	23:30	3:00
RedDwarf	2	22:30	0:00	1:30

Table 2. pass schedule sorted by start time

With Table 2, I generated the timeline chart as follows.

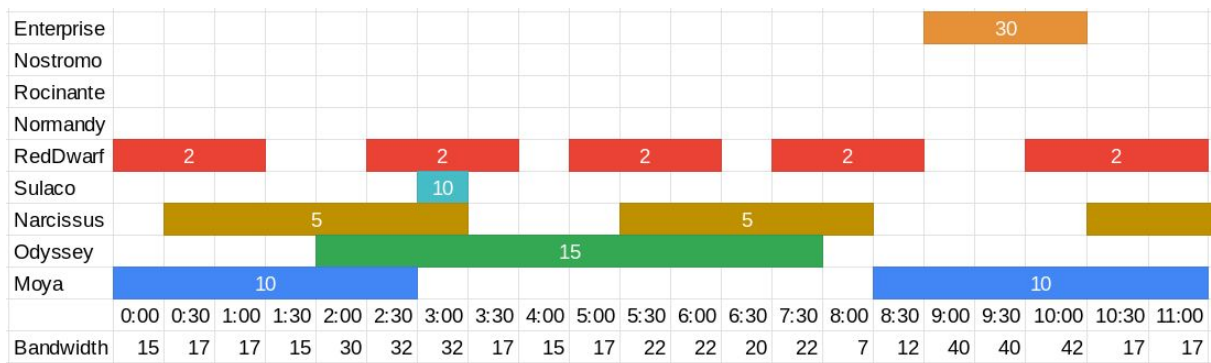


Figure 3. Timeline chart of the pass schedule

Now, it is clearer to calculate the total bandwidth consumed by the satellites during a given time.

Design of the solution

Approach 1:

In OOP, we can define the following class diagram (Figure 4) to represent our problem

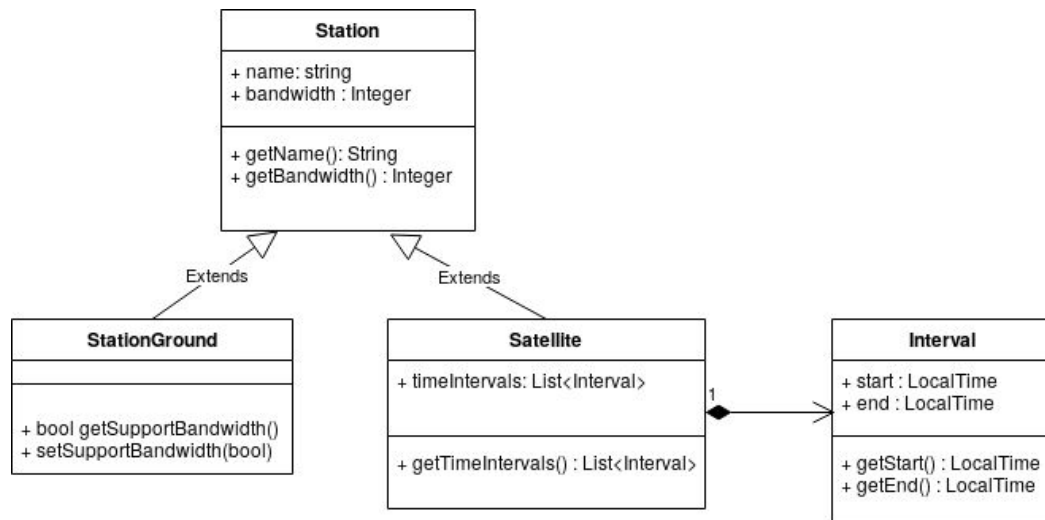


Figure 4. Class diagram of the problem WYSpace

The class **Satellite** extends **Station** and it has a List of time intervals.

The class **StationGround** also extends **Station**. In this case, I consider the bandwidth attribute as the maximum bandwidth.

The trivial solution is to visit each time and test the satellites that intersect the given time within its interval of communication, which requires $O(n*m)$. Where n is the number of time intervals during a day, and m is the number of satellites.

A first prototype of the algorithm can be as follow:

```
// declare a list of intervals as empty
periodTotalDownMax = {}

// maximum total bandwidth
maxTotalBandwidth = 0

// for each possible time in the interval
for time 00:00 to 00:00 (24:00) increased by 30:

    // calculate the total bandwidth occupied by the satellites in the given time
    totalBandwidth = 0
    for each satellite in pass schedule:
        if satellite.overlap(time) :
            totalBandwidth+= satellite.getBandwidth()
```

```

// whether the total bandwidth is higher than the support by the ground station
if totalBandwidth <= stationGround.getBandwidth():

    // keep the maximum total bandwidth
    if totalBandwidth > maxTotalBandwidth :
        periodTotalDownMax.clear()
    if totalBandwidth >= maxTotalBandwidth :
        maxTotalBandwidth = totalBandwidth
        periodTotalDownMax = {time}

else:
    set support bandwidth to false in stationGround

```

Approach 2:

From this point, I have a first prototype that runs in linear time complexity. Note that sometimes its time complexity can be quadratic, e.g., when $m = n$.

I think I can improve the solution with a better data structure. Thus, doing research on the Internet I found the Interval Tree data structure [1], which is an extension of a red-black tree. The Interval Tree maintains a dynamic set of elements, with each element i containing an interval $[i.start, i.end]$.

We say that intervals i and j overlap if the intersection between i and j is different from null. That is, if $i.start \leq j.end$ and $j.start \leq i.end$. Figure 5 shows all the cases where the two intervals satisfy the overlap.

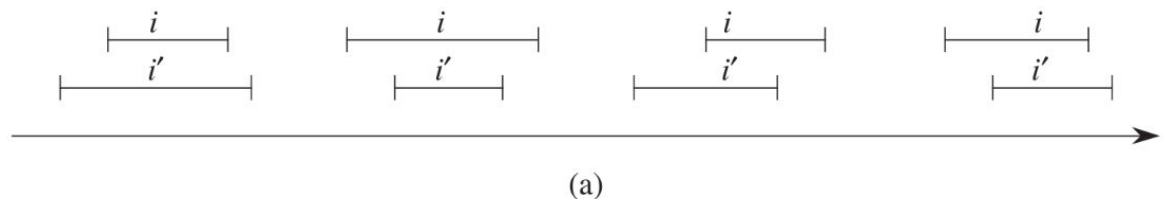


Figure 5. Cases where the two intervals overlap. Figure from [1].

With all this in mind, let's see the next proof of concept.

Let us assume the time intervals of the satellites ReDwarf, Sulaco, Narcissus, and Moya, shown in Figure 6.

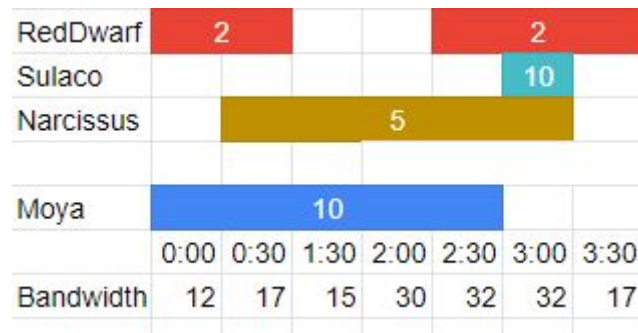


Figure 6. Time intervals proof of concept

A possible Interval tree that can be generated is shown in Figure 7.

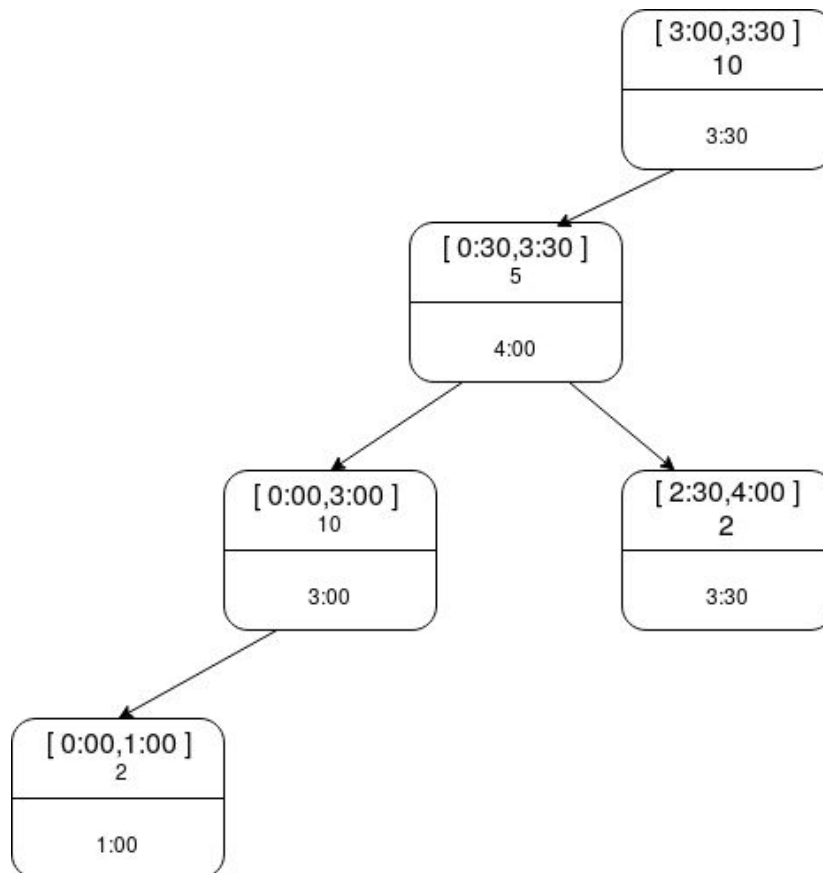


Figure 7. Interval Tree of the proof of concept

We this Interval tree, we calculate easily the total bandwidth occupied in a given time. For instance, given the time 0:30 and starting from the root of the node, we traverse the tree adding the bandwidth of any node which intervals fit 0:30. In this case, it would be : $5 + 10 + 2 = 17$.

With this approach, we reduce the time complexity of our algorithm to linear logarithm, $O(n \cdot \log m)$.

In regards with the classes, we will need the following classes:

IntervalNode

- interval : Interval
- left : IntervalNode
- right : IntervalNode
- max : integer
- bandwidth : integer

Note: From now, the class Satellite created in approach 1, is not necessary. So, we can remove it if we want.

Development

25/09/2020 14:00

With all the design clear, I start to develop the prototype in Java. I will develop the approaches 1 and 2 discussed in the previous section.

Approach 1

GitLab version :

During the development, I perform an improvement to the class diagram of Figure 4.

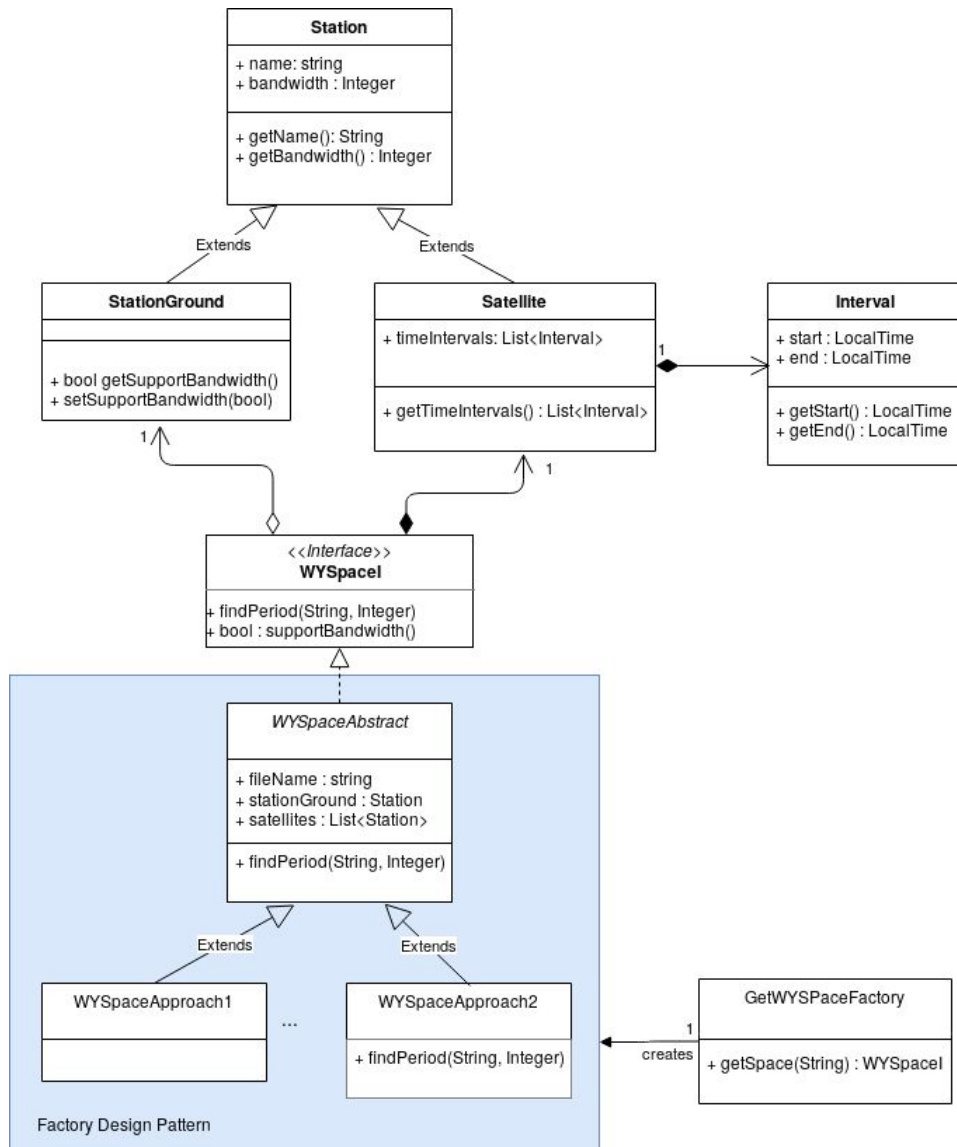


Figure 8. New Class Diagram of the proposed solution

Figure 8 shows the new class diagram. It has three new classes (WYSpaceApproach1, WYSpaceApproach2, GetWYSpaceFactory), an Interface (WYSpaceI) and an Abstract class (WYSpaceAbstract). All of them developed with the Factory design pattern.

The main reason to use the factory design pattern is that I am going to implement two approaches to solve the problem. With this pattern, I can easily not only change between the two approaches but add a new one without affecting all the code. I only need to add the new class and insert the creation in the class GetWYSpaceFactory.

Approach 2

27/09/2020 11:00

GitLab version :

Test

References

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms*. MIT press.