Task 2a (Coding Exercises)

https://github.com/jeanbou/Coding_Interview_Preparation/tree/master/SES/task2a

(1) **You may assume that there is only one of such character.**
In the provided solution above; I don't assume that there is only one, but I take the very first with MaxFrequency (including empty string case)

(2) **Select a data structure to store the input.**
HashMap<String,Integer> where the key is strings of given array

(3) **Explain why you have used that particular data structure.**
Because the check of presence of key is O(1) and retrieval an element (frequency) by the key in HashMap is also O(1)

(4) **Analyze the time and space complexity of your algorithm.**
Algorithm complexity is O(N) and the lower bond is also O(N) and can't be reduced it: the given array is random.
The space complexity of HashMap is O(N), however it's not the lowest possible bound. In the case when the given array full of empty strings or you have less than 3 elements in it, HashMap won't be used at all, however, they are low probable marginal cases.

(5) **Use any language you prefer (even pseudo-code).**
Java was used, see the listing above.

Task 2b (Pseudo Golang-like-code analysis)

1. The code obtains two list *satellites* and *gateways* via *db* requests. Using those two lists, a procedure forms two variable (two maps) inside the cycles *satMap* & *g*, but nobody uses it inside the procedure lately. I don't know nothing about those "mystical" sub procedure LongLastingOperation(), but probably it uses global variables *satellites* and *gateways*. I assume *satellites* and *gateways* they are global variables.
   The variable *db*, well I assume that *db* is a global variable defined somewhere above.
2. *s* variable should be initialized in the line 5 as *satMap* and instead of *g* one should use *gatwMap*, such naming will help to understand the code. I have the same remark for naming variable sp (*satelliteParams*) that what I prefer.
3. For me, *ephemerisType* that contains of mentioned in description fileType is a useless parameter. The incoming parameter should be removed, because for both types *A & B* we do the same processing, why should we do it if there is no difference at the end?
4. If we are sure that we need to initialize both 3 global lists (satellites, gateway and sp) before running mystical procedure LongLastingOperation() that use them in inexplicit way, let's retrieve them first in the first lines view lines of procedure ComputeVisHandler and do some checks of this piece of data retrieved. What about to close connection to db?
5. I'm worried that list of satellites and list of params of satellites are completely different lists, however, they should not be disconnected (each satellites have params, params are the part of satellite). I would prefer to have some preprocessing procedures that after retrieve data from DB, forms the two list of objects *Satellites* and *Gateways*, then there should be some procedure that check the validity of those list and objects inside: no null fields(!). Only after that I prefer to pass them in explicit way to *ComputeVisHandler(…* procedure to work on them. Again, here we talk

about the naming convention problem, because you do not compute here in this part. I would like to have the complicated object Satellites with all needed data-params incapsulated and not 3 lists.

6. I assume *for _, ss:=range satellites {* and not *…range s {* . Similarly *for _, gg:=range gateways {*
   I assume to *for _, vv := range sp {* ; *range* was missing instruction and I don't recommend to use *ss* for indexing; because it was already used. If it's possible to do so for your GoLand Pseudo Language using the variable visibility zone specificity (i.e. inside if variable *ss* and outside if *ss* are not the same). I'm against of such naming economizing approach, plus making the name longer, such as: *velocityInd* or whatever the naming convention established, will help to understand code easily. As a result of proposed changes you should call *computeVis(vv…* and not with *ss*
7. // zzzz…. LongLastingOperation() investigate, debug, why it is long, change the name after, add incoming parameters. It looks like it closes all connections to DBs, but why such strange name. If it is not related to the business logic, rename it. Without parameter passed I cannot say anything about this call.
8. *v := make(*… you should pass after correction *satMap* otherwise why to pass index of velocity *ss*? What for? *satMap!* Again, to be clear, first retrieve 3 lists from DB (the 1st group of operations), fill the local objects nicely in 3 cycles after (the 2nd group of operations), give a right naming *satMap* instead of s, *gatwMap* instead of g and *satparam**Array*** instead of v. However, I prefer decouple the 1st group of operation and the 2nd one and the 3rd group and do not have them in one procedure. I prefer in *Compute…* have strictly computing business logic.
9. Note for a mystical sub procedure LongLastingOperation() that works by the mean of modifying global variable. If during the debugging I will find out that it is not modifying something that we use for *v=* array construction in the loop, in this case I don't need to retrieve *satellites* & *gateways*. I need just to retrieve the list of *sp* (satellite params)
10. Because of the mess up with *s* and *satMap*, I assume that in line 5 you should rename s to satMap with 99.9% probability and therefor in such case the mystery of *s SateliteRouter* is clear it is an additional param for the call of *v:=computeVis(* that requires sateliteParams . It makes sense to use there a satellite router.
11. We return *v* array, but in the head of the function there is no definition of returned type. Well it is not specified and probably it's Okay for your GoLangPseudoLanguage, but in this case you should rename the function to specify the type of data that it returns.