Instructions for the challenge (**non-negotiable**):

1. Solution format:

```
<full-name>_solution.tar.gz
|
-- 1/
|  |
|  -- README.md
|  -- <solution-entrypoint-executable>
|  -- inputs/
|  |  |
|  |  -- sample-input
|  |  -- ...
|  -- <optional-sidecar-dir>/
|  |  |
|  |  -- ...
-- 2/
|  |
|  -- ... (see above)
...
```

2. Once extracted, solutions shall be evaluated as follows:

```
$ cd 1/ && \
> cat inputs/sample-input | ./<solution-entrypoint-executable>
```

3. Input is to be consumed via stdin, the output is expected on stdout, and error logging is expected on stderr.
4. Include the given sample input and any other test cases along with submission, as shown above.
5. The single tarball generated is to be attached with the reply to the challenge email to the concerned recruiter before the deadline.
6. Your solutions shall be judged not only by their correctness, but also by pragmatic choices of toolchain/languages, security, reliability, debuggability, maintainability, and general mindfulness that is expected of a team player.
7. Questions to be solved - The candidate can solve [Q1 or Q2] + Q3 (Q3 is mandatory)

I. Repetitions, everywhere, everywhere.

- You are given a text file, which contains English words. However, as a result of the typist's mental fatigue, there are many duplicate words in the file.
- Your task is to delete all *consecutive duplicate occurrences* of any word, which are separated by a single space.
- Three or more consecutive occurrences of any word will not be there in input, there are no punctuation marks.
- All words are lowercase only.
- You're expected to write a bash single-line command which expects input from a file and outputs the modified text to stdout. You're permitted to use generally available Linux utilities, such as awk, sed, grep, etc.
- *Note: You must use GNU/Linux versions of the tools. If you're using a mac, be careful, a lot of commands won't work without modifications. If you're using a Mac, you're encouraged to either install the gnu version or use a docker container.*
- Include a README.md file with your solution, which explains how your solution works.
- References
    - https://regex101.com/
    - https://www.gnu.org/software/grep/manual/grep.html
    - https://www.grymoire.com/Unix/Sed.html
    - https://www.gnu.org/software/gawk/manual/gawk.html

Sample input:

```
double double toil and trouble
fire burn and cauldron bubble bubble
tomorrow and tomorrow and tomorrow
creeps in this this petty pace from day toto day
to the last syllable of recorded time time
```

Sample output:

```
double toil and trouble
fire burn and cauldron bubble
tomorrow and tomorrow and tomorrow
creeps in this petty pace from day toto day
to the last syllable of recorded time
```

II. Analyzing shard data

- You have an elasticsearch cluster, whose state you need to analyze.
- Your script will be given the output of /_cat/shards api from elasticsearch.
- The api outputs a table of all the shards in your elasticsearch cluster.

Sample input

```
web1 0 p STARTED    3014 31.1mb 192.168.56.10 H5dfFeA
web2 0 r UNASSIGNED
web3 1 r STARTED    3014 31.1gb 192.168.56.20 I8hydUG
```

Columns are in this sequence: (index-name) (shard number) (primary ‘p’ or replica ‘r’) (state) (document count) (store size) (node IP) (node name)
Possible states of the shard: `STARTED, UNASSIGNED`

Write a script which will read the above api outputs from a file, and analyse the cluster. Your script needs to output:

- Number of primary shards and replica shards, separately.
- Total size occupied by the primary shards and replica shards, separately. Use mb, gb, kb etc. measurement units depending on the magnitude (size magnitude must be > 0). For example, if the total storage is 0.5gb, mention it as 500.0mb to improve readability. Similarly, to represent 19824.0 mb, you'll use 19.4 gb. Output upto one decimal place, the smallest shard size unit is kb, largest is tb.
- Name of the elasticsearch node where the disk usage is maximum out of all nodes.
- Assume a 128gb disk for each node; list the nodes where the 80% disk watermark has been crossed.

Sample output:

```
count: [primary: 7, replica: 15]
size: [primary: 718.0mb, replica: 1.3gb]
disk-max-node: H5dfFeA
watermark-breached: [H5dfFeA, I8hydUG]
```

III Search autocomplete

- You have to implement a system which autocompletes a word query.
- There will be 2 components of the system: a database and an api server
- You are required to use the redis in-memory database for storing words
- You are required to implement the api in a programming language/framework of your choice (python is preferable, you can use flask for the api). There should be two endpoints of this api:
    - `/add_word` - adds a word to the dictionary
    - `/autocomplete` - returns a json list of words which could be formed from this prefix
- For auto-completion, use the inbuilt functionalities provided by redis *(you do not have to implement the algorithm itself)*. Solutions which iterate over all the dictionary words to get the output will be penalized accordingly.
- You are expected to organize the code for your api in a directory structure, with a Dockerfile (see structure below).
- Create a README.md file in your solution, which will contain instructions on how to replicate your setup. This will typically include building your api-container docker image, what command(s) to run such that the api is ready-to-use, sample usage of the api's, and anything else you think necessary.
- Suggested reading:
    - http://oldblog.antirez.com/post/autocomplete-with-redis.html
    - https://redislabs.com/ebook/part-2-core-concepts/chapter-6-application-components-in-redis/6-1-autocomplete/

Sample input:

```
/add_word/word=foo
/add_word/word=example
/add_word/word=foobar
/add_word/word=exam
```

Sample output:

```
/autocomplete/query=fo
[
       "foo",
       "foobar"
]
```

Expected solution directory structure (you may have more files besides these):

```
III
├── README.md
└── api
    ├── Dockerfile
    ├── README.md
    └── app.py
```