☆ Hosts and the Total Number of Requests

In this challenge, write a program to analyze a log file and summarize the results. Given a text file of an http requests log, list the number of requests from each host. Output should be directed to a file as described in the Program Description below.

The format of the log file, a text file with a *.txt* extension, follows. Each line contains a single log record with the following columns (in order):

1. The *hostname* of the *host* making the request.
2. This column's values are missing and were replaced by a hyphen.
3. This column's values are missing and were replaced by a hyphen.
4. A timestamp enclosed in square brackets following the format *[DD/mmm/YYYY:HH:MM:SS -0400]*, where *DD* is the day of the month, *mmm* is the name of the month, *YYYY* is the year, *HH:MM:SS* is the time in *24*-hour format, and *-0400* is the time zone.
5. The *request*, enclosed in quotes (e.g., *"GET /images/NASA-logosmall.gif HTTP/1.0"*).
6. The *HTTP response code*.
7. The total number of *bytes* sent in the response.

▶ Example log file entry

Function Description

Your function must create a unique list of hostnames with their number of requests and output to a file named *records_filename* where *filename* is replaced with the input *filename*. Each hostname should be followed by a space then the number of requests and a newline. Order doesn't matter.

Constraints

- The log file has a maximum of 2×10^5 lines of records.

▶ Input Format

▼ Sample Case 0

Sample Input 0

```
hosts_access_log_00.txt
```

Sample Output 0

Given *filename* = *"hosts_access_log_00.txt"*, process the records in *hosts_access_log_00.txt* and create an output file named *records_hosts_access_log_00.txt* which contains the following rows:

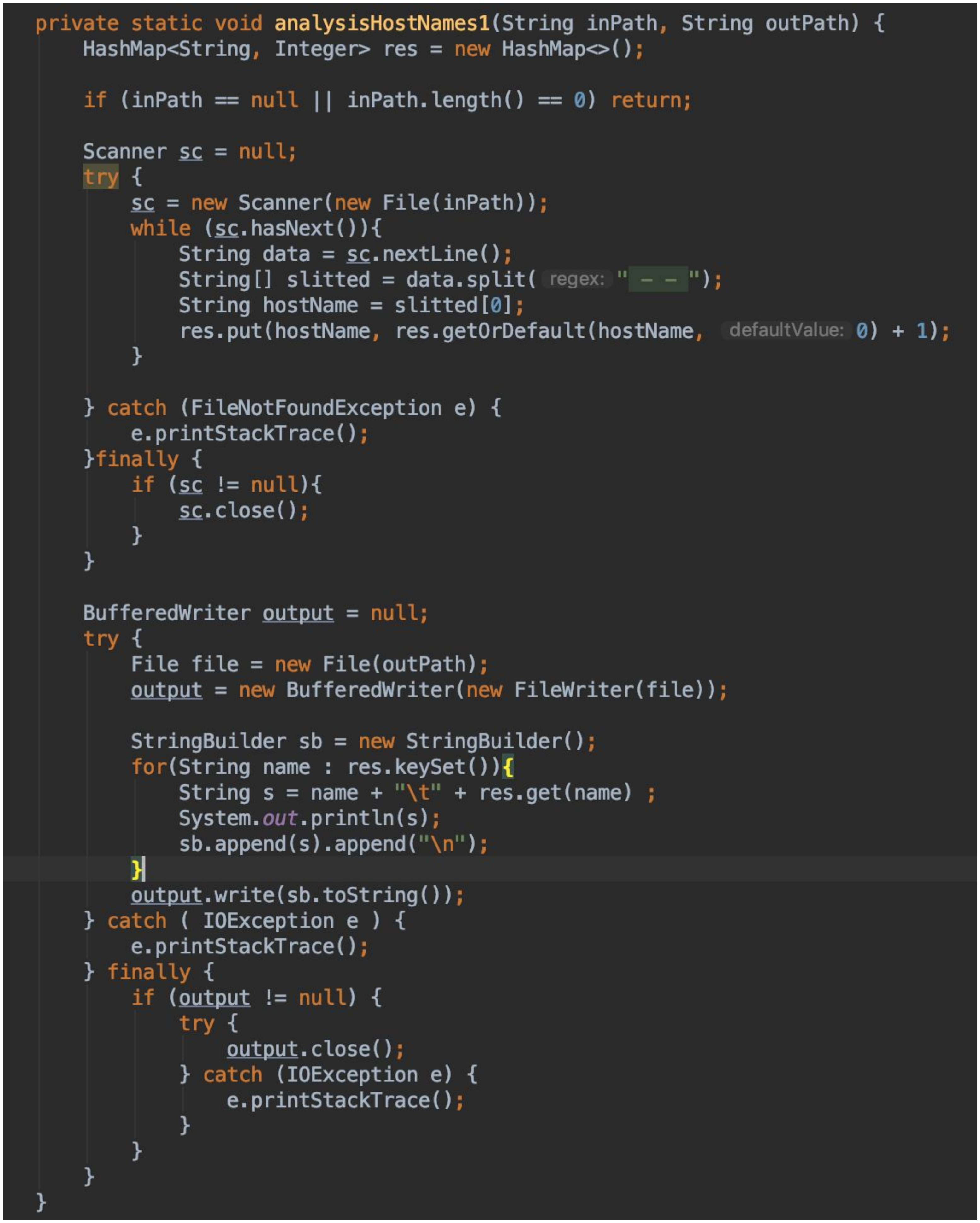
```
burger.letters.com 3
d104.aa.net 3
unicomp6.unicomp.net 4
```

Explanation 0

The log file *hosts_access_log_00.txt* contains the following log records:

```
unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
burger.letters.com - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/countdown/liftoff.html HTTP/1.0" 304 0
burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 304 0
burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /shuttle/countdown/video/livevideo.gif HTTP/1.0" 200 0
d104.aa.net - - [01/Jul/1995:00:00:13 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400] "GET /shuttle/countdown/count.gif HTTP/1.0" 200 40310
unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 200 786
unicomp6.unicomp.net - - [01/Jul/1995:00:00:14 -0400] "GET /images/KSC-logosmall.gif HTTP/1.0" 200 1204
d104.aa.net - - [01/Jul/1995:00:00:15 -0400] "GET /shuttle/countdown/count.gif HTTP/1.0" 200 40310
d104.aa.net - - [01/Jul/1995:00:00:15 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 200 786
```

When the data is consolidated, it confirms the following:



```java
private static void analysisHostNames1(String inPath, String outPath) {
    HashMap<String, Integer> res = new HashMap<>();

    if (inPath == null || inPath.length() == 0) return;

    Scanner sc = null;
    try {
        sc = new Scanner(new File(inPath));
        while (sc.hasNext()){
            String data = sc.nextLine();
            String[] slitted = data.split( regex: " - - ");
            String hostName = slitted[0];
            res.put(hostName, res.getOrDefault(hostName,  defaultValue: 0) + 1);
        }

    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }finally {
        if (sc != null){
            sc.close();
        }
    }

    BufferedWriter output = null;
    try {
        File file = new File(outPath);
        output = new BufferedWriter(new FileWriter(file));

        StringBuilder sb = new StringBuilder();
        for(String name : res.keySet()){
            String s = name + "\t" + res.get(name) ;
            System.out.println(s);
            sb.append(s).append("\n");
        }
        output.write(sb.toString());
    } catch ( IOException e ) {
        e.printStackTrace();
    } finally {
        if (output != null) {
            try {
                output.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

☆ Missing Words

Given two strings, one is a subsequence if all of the elements of the first string occur in the same order within the second string. They do not have to be contiguous in the second string, but order must be maintained. For example, given the string "I like cheese", the words "I" and "cheese" are one possible subsequence of that string.

In this challenge, you will be given two strings, *s* and *t*, where *t* is a subsequence of *s*, report the words of *s*, *missing in t (case sensitive),* in the order they are missing. Revisiting the earlier example, if *s = I like cheese* and *t = like*, then *like* is the longest subsequence, and *[I, cheese]* is the list of missing words in order.

Function Description

Complete the function *missingWords* in the editor below. It must return an array of strings containing any words in *s* that are missing from *t* in the order they occur within *s*.

missingWords has the following parameter(s):

s: a sentence of space-separated words

t: a sentence of space-separated words

Constraints

- Strings *s* and *t* consist of English alphabetic letters (i.e., *a−z* and *A−Z*) and spaces only.
- $1 \le |t| \le |s| \le 10^6$
- $1 \le$ length of any word in *s* or *t* ≤ 15
- It is guaranteed that string *t* is a subsequence of string *s*.

▶ Input Format for Custom Testing

▼ Sample Case 0

Sample Input 0

```
I am using HackerRank to improve programming
am HackerRank to improve
```

Sample Output 0

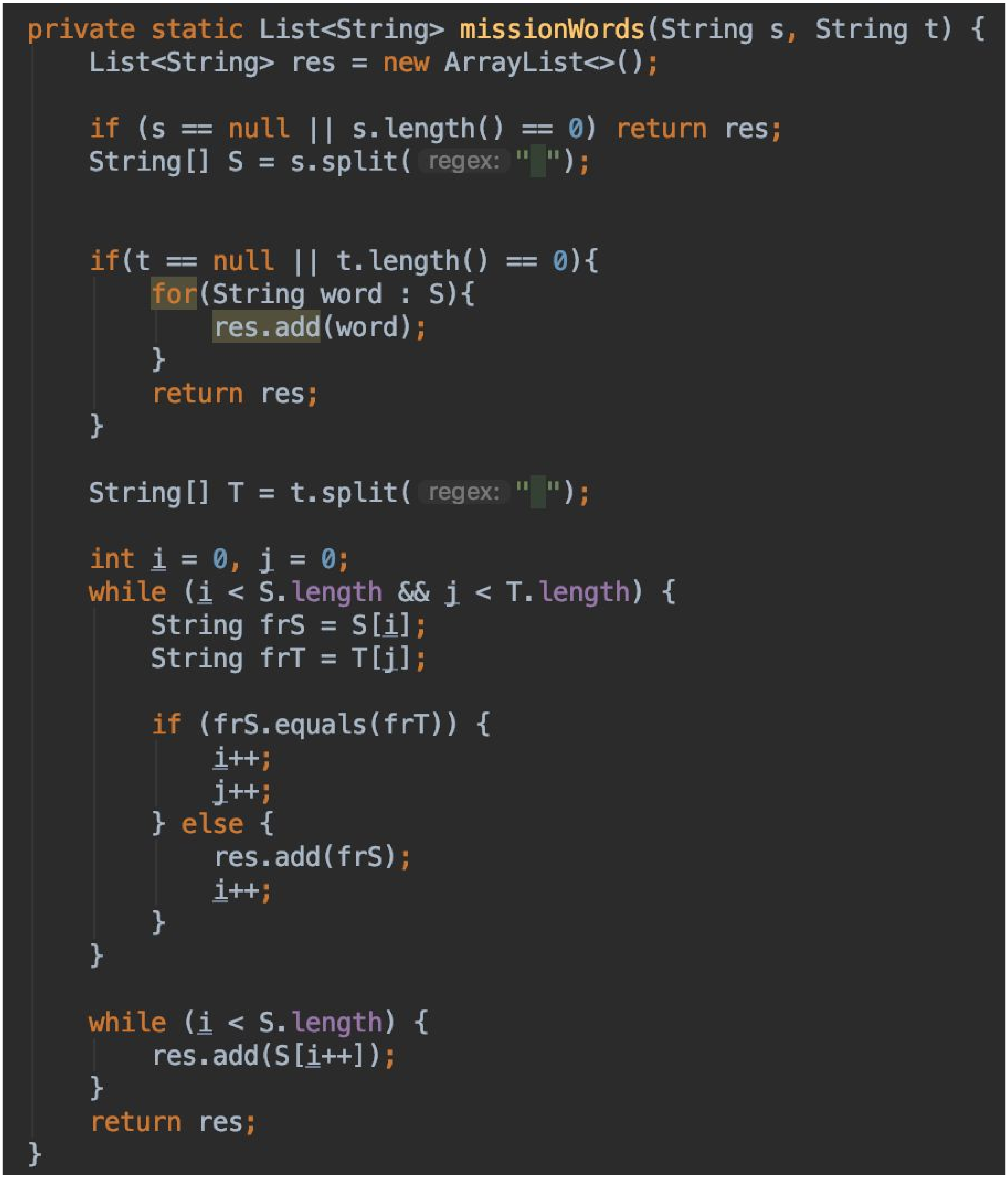
```
I
using
programming
```

Explanation 0

The missing words are:

1. I
2. using
3. programming

We add these words *in order* to the array *["I", "using", "programming"]*, then return this array as our answer.



```java
private static List<String> missionWords(String s, String t) {
    List<String> res = new ArrayList<>();

    if (s == null || s.length() == 0) return res;
    String[] S = s.split( regex: " ");


    if(t == null || t.length() == 0){
        for(String word : S){
            res.add(word);
        }
        return res;
    }

    String[] T = t.split( regex: " ");

    int i = 0, j = 0;
    while (i < S.length && j < T.length) {
        String frS = S[i];
        String frT = T[j];

        if (frS.equals(frT)) {
            i++;
            j++;
        } else {
            res.add(frS);
            i++;
        }
    }

    while (i < S.length) {
        res.add(S[i++]);
    }
    return res;
}
```