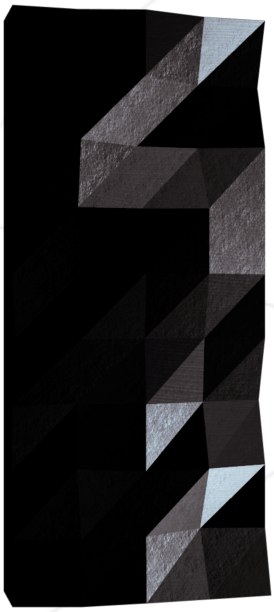

Starfleet

Interview

Staff 42 pedago@42.fr

Summary: This document is an interview question for the Starfleet Piscine.

Contents

Chapter I

General rules

- The interview should last between `45 minutes`.
- Both the interviewer and the interviewed student must be present.
- The interviewed student should write his code using a `whiteboard`, with the language of her/his choice.
- At the end of the interview, the interviewer evaluates the student based on the provided criteria.

Read carefully the interview question and solutions, and make sure you `understand` them before the interview. You can't share this document with other students, as they might be interviewed on the same question. Giving them the answer would prevent them from having to solve an unknown question during an interview.

I.1 During the interview

During the interview, we ask you to :

- Make sure the interviewed student `understands` the question.
- Give her/him any `clarification` on the subject that she/he might need.
- Let her/him come up with a solution before you guide her/him to the best solution given the constraints (time and space).
- Ask the student what is the `complexity` of her/his algorithm ? Can it be improved and how ?
- `Guide` her/him to the best solution without giving the answer. You may refer to the `hints` for that.
- You want to evaluate how the interviewed student thinks, so ask her/him to `explain everything` that she/he thinks or writes (there should be no silences).
- If you see a mistake in the code, wait untill the end and give her/him a chance to correct it by her/himself.
- Ask the student to show how the algorithm works on an `example`.
- Ask the student to explain how `limit cases` are handled.
- Bring out to the student any mistake she/he might have done.
- Give `feedback` on her/his performances after the interview.
- Be `fair` in your evaluation.

As always, stay mannerly, polite, respectful and constructive during the interview. If the interview is carried out smoothly, you will both benefit from it !

Chapter II

Decode

II.1 Interview question

If a = 1, b = 2, c = 3, ... and z = 26.

Given a string, find all the possible codes that string can generate. Give a count as well as print the strings.

Example :

Generate all valid alphabet codes from string "1123".

```
Input : "1123"

Output :

aabc // a = 1, a = 1, b = 2, c = 3
aaw  // a = 1, a = 1, w = 23
alc  // a = 1, l = 12, c = 3
kbc  // k = 11, b = 2, c = 3
kw   // k = 11, w = 23

count = 5
```

II.2 Acceptable answers (no constraint)

II.2.1 With recursion

Generate all valid alphabet codes from string using recursion.

```
O(2^n) time , O(n) space
where n is the length of input code
```

code:

```c
int decode(char *prefix, char *code, int i) {
        char c;
        int count = 0;

        // if the end of the code string has been reached, print prefix
        if (code[0] == '\0') {
                prefix[i] = '\0';
                printf("%s\n", prefix);
                return (1);
        }

        // decode for 1 digit
        c = (code[0] - '1') + 'a';
        prefix[i] = c;
        count += decode(prefix, code + 1, i + 1);

        // decode for 2 digits if valid alphabet code
        if (code[1]) {
                c = (code[0] - '0') * 10 + (code[1] - '1') + 'a';
                prefix[i] = c;
                if (c >= 'a' && c <= 'z')
                        count += decode(prefix, code + 2, i + 1);
        }

        // return the count
        return (count);
}

int decodeAll(char *code) {
        int len = strlen(code);
        char *prefix = malloc(sizeof(char) * (len + 1));
        return decode(prefix, code, 0);
}
```

`Note` : It is possible to optimize this algorithm with memoization (use of a hash table to store previous result so they can be retrive instead of doing the computation over again). But since we have to print all valid codes, it will take the same time.

II.3 Follow up question

Use `dynamic programming` to only count all valid codes in `O(n) time`, where n is the length of the input string.

II.3.1 Hints

- Why not `cache` the results and use them later?
- Maybe with an hash table.

II.4 Best solutions

II.4.1 Dynamic programming

Let's go through an example:

For input code "1123" :

```
len = 1, "1"     -> "1"

len = 2, "11"    -> "1" + "1"
                    "11"

len = 3, "112"   -> "1" + "1" + "2"
                    "11"      + "2"
                    "1"       + "12"

len = 4, "1123"  -> "1" + "1" + "2"  + "3"
                    "11"      + "2"  + "3"
                    "1"       " "12" + "3"
                    "1" + "1"        + "23"
                    "11"             + "23"
```

Everytime a digit is added to the input code :

- If it cannot form a valid alphabet code with the previous digit, the number of valid output codes is unchanged.
- If it can, the number of valid outputs increases by the number of valid output when the previous digit is removed.

```
O(n) time , O(n) space
where n is the length of input code
```

code:

```
int countAll(char *code, int len) {
        int count[len + 1];

        count[0] = 1; // for the base case where all characters are made of single digits
        for (int i = 1; i <= len; i++) {
                // if the 2 digits are a valid alphabet code
                if (i > 1 && (code[i - 2] - '0') * 10 + (code[i - 1] - '0') <= 26) {
                        count[i] = count[i - 1] + count[i - 2];
                } else { // otherwise
                        count[i] = count[i - 1];
                }
        }
        return (count[len]);
}
```

`Note` : If you really think about how this works, you only use `count[i]` for `count[i+1]` and `count[i+2]`. You don't need it after that. Therefore, we can get rid of the `count` table and just store a few variables.

```
O(n) time , O(1) space
where n is the length of input code
```

code:

```
int countAll(char *code, int len) {
        int a = 0;
        int b = 1;
        int c;

        for (int i = 1; i <= len; i++) {
                // if the 2 digits are a valid alphabet code
                if (i > 1 && (code[i - 2] - '0') * 10 + (code[i - 1] - '0') <= 26) {
                        c = a + b;
                } else { // otherwise
                        c = b;
                }
                a = b;
                b = c;
        }
        return (c);
}
```