Question 1.5 from CTCI: Replace Spaces

July 25, 2017

Question

Write a method to replace all spaces in a string with "%20". Use the following method definition:
public static char[] replaceSpaces(char[] str)
An alternate version of this problem involves resizing the existing char array to accommodate the new length. The solution for that is included as well. Use the following method definition:
public static void replaceSpaces(char[] str, int length)

Explanation and Algorithm

The solutions for both the original and alternate problem are included:

For the original problem, the solution is fairly simple. It involves counting the number of spaces in the original string, creating a new string that is resized using this information, and copying the original string while replacing spaces. First, the original string is looped through and the number of spaces noted. If there are no spaces counted, then the original string is returned. Otherwise, the new length of the string is calculated and a char array is created and sized accordingly. The original string is looped through and each character is copied with "%20" replacing the spaces. After this is done, the updated string is returned.

For the alternate problem, no new char array will be created. Instead, the original char array will be resized and the characters from the original string added in reverse with the spaces replaced. First, the original string is looped through and the number of spaces counted. The new length of the string is then calculated and the string is resized. Starting from the end of the string, each character is added in reverse with the spaces replaced when encountered. See the solution for further clarification.

Hints

1. Determine what information about the string should be tracked. (*The number of spaces*).
2. Consider how to calculate the new length of the string with spaces replaced.
3. Consider how to index through each character of the array to replace the spaces, keeping in mind the new length. How would you track the position of the character after a space?

Code

```
/*Solution 1 */

public static char[] replaceSpaces(char[] str){

   int numSpaces = 0;
   int i, j;
   int length = str.length;

   for(i = 0; i < length; i++){

      if(str[i] == ' '){
         numSpaces++;
      }

   }

   if(numSpaces == 0){

      return str;

   }

   int newLength = length + 2*numSpaces;

   char[] updatedStr = new char[newLength];
   i = 0;

   for(j = 0; j < length; j++){

      if(str[j] != ' '){
         updatedStr[i] = str[j];
         i++;
      } else {
         updatedStr[i] = '%';
         updatedStr[i + 1] = '2';
         updatedStr[i + 2] = '0';

         i += 3;

      }

   }

   return updatedStr;


}
```

```
/* Solution 2 (Alternate) */

public static void replaceSpaces(char[] str, int length){

   int spaceCount = 0, newLength, i = 0;
   for(i = 0; i < length; i++){
      if(str[i] == ' '){
         spaceCount++;
      }
   }

   newLength = length + spaceCount*2;
   str[newLength] = '\0';

   for(i = length - 1; i >= 0; i--){

      if(str[i] == ' '){

         str[newLength - 1] = '0';
         str[newLength - 2] = '2';
         str[newLength - 3] = '%';
         newLength = newLength - 3;

      } else {
         str[newLength - 1] = str[i];
         newLength--;
      }

   }
}
```

Big O analysis

1. Solution 1: The running time is $O(n)$.
2. Solution 2: The running time is $O(n)$.

Interviewer Considerations

Here are a few things to ask yourself as interviewer when judging the performance of your interviewee:

- Did the interviewee ask questions to clarify ambiguous portions of the problem statement? Things like input, output, data types, return type etc. Did they run into issues later because they did not have the foresight to ask?
- Were they able to classify and generalize the type of problem they were looking at? Were they able to do the same with potential solution algorithms? Did they compare the pros and cons of potential solutions *before* coding?
- Were they able to code a fully working solution with no logic errors and minimal syntax errors? If applicable, did they code the problem in a way that showed mastery of their chosen coding language including sensible library calls, programming constructs and coding conventions. Is the code easily readable?
- Did they run through their solutions and code with proper examples? Did they cover all corner cases?
- Did they do Big O analysis? Was it accurate? Did they consider both time and space? Did they distinguish between the Big O of multiple solutions if applicable?
- Through out the interview, did they clearly communicate their thought process through out all of the above? Did they observe, at least, basic social norms?

Interviewee rating

The basis of effective practicing is honest, constructive feedback. Please rate your interviewee honestly based on the following criteria. The ratings are meant to give an objective guideline to distinguish between the performance of interviewee's for a particular question, a list of things to look out for as an interviewer and a springboard for constructive criticism. It's highly encouraged to briefly talk about the mock interview afterward to highlight strengths and weaknesses so everyone what they can rely on and what ought to be improved. Furthermore, many companies use a similar system across interviews to decide who gets to advance to the next stages of interviews and to decide who gets the job. Being aware and becoming comfortable with this sort of performance analysis will also help you analyze your own performance in real interviews and improve.

1 star- Poor performance.

- Did not ask many or any clarifying questions and committed errors that could have been avoided by asking.
- Was not able to correctly classify the type of problem or the category of algorithms that might solve it.
- Was not able to get the solution despite getting many or all hints.
- Wrote no code or conceptually and syntactically flawed code.
- Did not work through any examples.
- Did not communicate in any way their thought process in any meaningful way or didn't talk at all.
- No or completely wrong Big O analysis.
- No insight into thought process through out.

2 stars - Below average performance.

- Asked no or non pertinent clarifying questions about the problem.
- Was not able to classify the question or algorithm or show any insights into how to generalize the problem or was able to do so in only the most superficial sense.
- Was able to get an almost working solution or just brute force after being given many hints.
- Wrote some code that worked only in minimal cases and/or was riddled with flaws.
- Worked through no examples or only the most straight forward type not considering corner cases.
- Attempted Big O analysis, but over simplified it or was wrong in more complex cases.
- Minimal insight into thought process.

3 stars - Average performance.

- Asked a proper amount of clarifying questions.

- Was able to recognize the class of problem the question came from and possible solutions.
- Was able to get at least one working solution with a few hints.
- Code got to working or near working state after a few revisions.
- Ran through code with an example at least once.
- Proper big O analysis for simple, generally known cases.
- Adequate thought process displayed for the question.

4 stars - Good performance.

- Asked important pertinent questions for the problem.
- Was able to code or conceptualize multiple working solutions with minimal hints and indicated thought process.
- Worked competently through examples including all or most corner cases.
- Ran through code with all important examples or proofs of working.
- Proper big O analysis for all, but the most complex algorithms.
- Displayed thought in a way that the average computer scientist would understand.

5 stars - Stellar performance.

- Asked questions to clarify key aspects of the questions if applicable.
- Was able to quickly and correctly identify the sort of problem, the approaches that may work and communicated them and analyze the quality of potential approaches before coding.
- Got the full solution(s) to the problem with no or minimal hints.
- Wrote fully working code with minimal syntax errors, no major conceptual errors and used proper coding conventions such as modularity and readability.
- Worked through pertinent examples which included both general cases and important corner cases.
- Communicated their thought process through out the entire process with enough clarity a layman could understand the logic (with exception of special knowledge, of course).
- In depth Big O analysis for solutions and proper comparisons between solutions.