Strings_1

@kbbhatt04

@July 5, 2023

Sort Characters By Frequency

- Approach
    - Better
        - Count frequencies of each char and store it along with the char in a list
        - Sort the list in descending order
        - Join the list into a string
        - Time Complexity: $O(n + klogk)$ where k <= 62 (include uppercase characters, lowercase characters and digits)
        - Space Complexity: $O(n)$
    - Optimal
        - Since freq values are in range [0...n], so we can use Bucket Sort to achieve $O(N)$ in Time Complexity
        - Count frequency of each char
        - Put char into frequency bucket
        - Iterate all buckets from last and append all chars to string (for `freq` times)
        - Time Complexity: $O(n)$
        - Space Complexity: $O(n)$

```
# Python3
# Better Solution
class Solution:
    def frequencySort(self, s: str) -> str:
        cnt = Counter(s)
        arr = [[freq, c] for c, freq in cnt.items()]
        arr.sort(key=lambda x:-x[0])

        ans = ""
        for freq, c in arr:
            for i in range(freq):
                ans += c
        return ans
```

```
# Python3
# Optimal Solution
class Solution:
    def frequencySort(self, s: str) -> str:
        cnt = Counter(s)
        n = len(s)
        bucket = [[] for _ in range(n+1)]
        for c, freq in cnt.items():
            bucket[freq].append(c)

        ans = []
        for freq in range(n, -1, -1):
            for c in bucket[freq]:
                ans.append(c * freq)
        return "".join(ans)
```

```
// C++
// Optimal Solution
class Solution {
public:
    string frequencySort(string s) {
        unordered_map<char,int> freq;
        vector<string> bucket(s.size()+1, "");
        string res;

        //count frequency of each character
        for(char c:s) freq[c]++;
        //put character into frequency bucket
        for(auto& it:freq) {
            int n = it.second;
            char c = it.first;
            bucket[n].append(n, c);
        }
        //form descending sorted string
        for(int i=s.size(); i>0; i--) {
            if(!bucket[i].empty())
                res.append(bucket[i]);
        }
        return res;
    }
};
```

Maximum Nesting Depth of the Parentheses

- Approach
    - Optimal
        - Maintain count of currently open parenthesis and max_ans
        - Iterate over all chars of string and if `char[i] == “(”` then increment open parenthesis count and set max_ans to `max(max_ans, open_count)`
        - Else if `char[i] == “)”`, decrement open_count
        - Time Complexity: $O(n)$
        - Space Complexity: $O(1)$

```
# Python3
# Optimal Solution
class Solution:
    def maxDepth(self, s: str) -> int:
        opn = 0
        ans = 0
        for i in s:
            if i == "(":
                opn += 1
                ans = max(ans, opn)
            elif i == ")":
                opn -= 1
        return ans
```

```
// C++
// Optimal Solution
class Solution {
public:
    int maxDepth(string s) {
        int maxi=0,curr=0;
        for(int i=0;i<s.size();i++){
            if(s[i]=='('){
                maxi=max(maxi,++curr);
            }else if(s[i]==')'){
                curr--;
            }
        }
        return maxi;
    }
};
```

Roman to Integer

- Approach
  - Optimal
    - `if roman[s[i]] < roman[s[i+1]]`, then subtract `roman[s[i]]` from ans else add it to ans
    - Time Complexity: $O(n)$
    - Space Complexity: $O(1)$

```
# Python3
# Optimal Solution
class Solution:
    def romanToInt(self, s: str) -> int:
        roman = {
            "I": 1,
            "V": 5,
            "X": 10,
            "L": 50,
            "C": 100,
            "D": 500,
            "M": 1000
        }

        ans = 0
        for i in range(len(s) - 1):
            if roman[s[i]] < roman[s[i+1]]:
                ans -= roman[s[i]]
            else:
                ans += roman[s[i]]
        ans += roman[s[-1]]
        return ans
```

```
// C++
// Optimal Solution
int romanToInt(string s)
{
    unordered_map<char, int> T = { { 'I' , 1 },
                                   { 'V' , 5 },
                                   { 'X' , 10 },
                                   { 'L' , 50 },
                                   { 'C' , 100 },
                                   { 'D' , 500 },
                                   { 'M' , 1000 } };

   int sum = T[s.back()];
   for (int i = s.length() - 2; i >= 0; --i)
   {
       if (T[s[i]] < T[s[i + 1]])
       {
           sum -= T[s[i]];
       }
       else
       {
           sum += T[s[i]];
       }
   }

   return sum;
}
```

Implement atoi()

- Approach
    - Optimal
        - Skip all whitespace chars
        - Check for negative/positive sign of first non-whitespace char
        - Iterate over remaining chars and add the digit to ans
        - If any non-digit char is encountered, break from loop
        - If negative sign was encountered, make ans = -ans
        - If the resultant ans is out of bound (32-bit) then clamp the ans else return ans
        - Time Complexity: $O(n)$
        - Space Complexity: $O(1)$

```
# Python3
# Optimal Solution
class Solution:
    def myAtoi(self, s: str) -> int:
        negative = False
        n = len(s)
        i = 0

        # skip all whitespaces
        while i < n:
            if s[i] == " ":
                i += 1
            else:
                break

        # check if end of string is reached (blank string or all whitespaces)
        if i == n:
            return 0

        # check if first non-whitespace char is "-", then ans will be negativ else positive
        if s[i] == "-":
            negative = True
            i += 1
        elif s[i] == "+":
            i += 1

        ans = 0
        while i < n:
            temp = ord(s[i]) - ord("0")

            # if any non-digit char is encountered, break from loop
            if temp >= 10 or temp < 0:
                break

            ans *= 10
            ans += temp
            i += 1

        if negative:
            ans = -ans
        if ans < -pow(2, 31):
            return -pow(2, 31)
        if ans > pow(2, 31) - 1:
            return pow(2, 31) - 1
        return ans
```

```
// C++
// Optimal Solution
class Solution {
public:
  int myAtoi(string s) {

    const int len = s.size();

    if(len == 0){
        return 0;
    }

    int index = 0;

    // skipping white spaces
    while(index < len && s[index] == ' '){
        ++index;
    }

    // to handle sign cases
    bool isNegative = false;

    if(index < len){

      if(s[index] == '-'){
        isNegative = true;
        ++index;
      } else if (s[index] == '+'){
          ++index;
      }

    }

    int result = 0;

    // converting digit(in character form) to integer form
    // iterate until non-digit character is not found or we can say iterate till found character is a digit
    while(index < len && isDigit(s[index])){

      /* s[index] - '0' is to convert the char digit into int digit eg: '5' - '0' --> 5
      or else it will store the ASCII value of 5 i.e. 53,
      so we do 53(ASCII of 5) - 48(ASCII of 0(zero)) to get 5 as int*/
      int digit = s[index] - '0';

      // to avoid integer overflow
      if(result > (INT_MAX / 10) || (result == (INT_MAX / 10) && digit > 7)){
          return isNegative ? INT_MIN : INT_MAX;
      }

      result = (result * 10) + digit; // adding digits at their desired place-value

      ++index;
    }

    return isNegative ? -result : result;
  }

private:
  bool isDigit(char ch){
    return ch >= '0' && ch <= '9';
  }
};
```

Reverse Words in a String

- Approach
    - Better
        - Push all the words in a list/stack
        - Pop from stack until the stack is empty and join words in a string
        - Time Complexity: $O(n)$
        - Space Complexity: $O(n)$
    - Optimal
        - Reverse the given string and then reverse individual words
        - Can only be done in programming languages that have mutable strings
        - Time Complexity: $O(n)$
        - Space Complexity: $O(1)$

```
# Python3
# Better Solution
class Solution:
    def reverseWords(self, s: str) -> str:
        res = []
        temp = ""
        for c in s:
            if c != " ":
                temp += c
            elif temp != "":
                res.append(temp)
                temp = ""
        if temp != "":
            res.append(temp)
        return " ".join(res[::-1])
```

```
// C++
// Better Solution
```

```
class Solution {
public:
    string reverseWords(string s) {
        vector<string> v;
        string temp = "";
        for (int i = 0; i < s.size(); i++) {
            if (s[i] == ' ') {
                if (temp.size() != 0){
                    v.push_back(temp);
                }
                temp = "";
            }
            else {
                temp += s[i];
            }
        }
        if (temp.size() != 0) {
            v.push_back(temp);
        }
        string ans = "";
        for (int i = v.size() - 1; i >= 0; i--) {
            ans += v[i] + " ";
        }
        ans.pop_back();
        return ans;
    }
};
```

```
// C++
// Optimal Solution
class Solution {
public:
    void rev_word(string &s, int i, int j) {
        int n = j - i;
        for (int a = 0; a <= int(n / 2); a++) {
            int temp = s[i+a];
            s[i+a] = s[n-a+i];
            s[n-a+i] = temp;
        }
    }

    string reverseWords(string s) {
        if (s.size() == 1) {
            if (s[0] == ' ') {return "";}
            return s;
        }

        int len = 0;
        string ns = "";

        for (auto i: s) {
            if (len == 0 && i != ' ') {
                ns += i;
                len++;
            }
            else if (len == 0 && i == ' ') {
                continue;
            }
            else if (len != 0 && ns[len-1] == ' ' && i == ' ') {
                continue;
            }
            else {
                ns += i;
                len++;
            }
        }
        if (ns[len-1] == ' ') {ns.pop_back();len--;}
        rev_word(ns, 0, len-1);

        int i = 0, j = 1;
        while (j < len) {
            if (ns[j] == ' ') {
                rev_word(ns, i, j-1);
                i = j+1;
                j = i+1;
            }
            else {j++;}
        }
        rev_word(ns, i, j-1);
        return ns;
    }
};
```

Sun of Beauty of all Substrings

- Approach
  - Better
    - Generate all substrings
    - Generate frequency array for all substring
    - Get max_freq and min_freq, calculate beauty and add it to ans variable
    - Time Complexity: $O(n^3)$
    - Space Complexity: $O(26)$
  - Optimal
    - Iterate over all substrings incrementally
    - Maintain frequency array incrementally
    - Get max_freq and min_freq, calculate beauty and add it to ans variable
    - Time Complexity: $O(26 * n^2)$
    - Space Complexity: $O(26)$

```
# Python3
# Better Solution
import math
class Solution:
    def beautySum(self, s: str) -> int:
        ans = 0

        for i in range(len(s)):
            for j in range(i+1, len(s)):
                freq = [0 for _ in range(26)]
                max_freq = -math.inf
                min_freq = math.inf
                for k in range(i, j+1):
                    freq[ord(s[k]) - ord("a")] += 1
                for l in range(26):
                    if freq[l] != 0:
                        max_freq = max(max_freq, freq[l])
                        min_freq = min(min_freq, freq[l])
                ans += (max_freq - min_freq)
        return ans
```

```
# Python3
# Optimal Solution
import math
class Solution:
    def beautySum(self, s: str) -> int:
        ans = 0

        for i in range(len(s)):
            freq = [0 for _ in range(26)]
            for j in range(i, len(s)):
                freq[ord(s[j]) - ord("a")] += 1
                max_freq = -math.inf
                min_freq = math.inf
                for l in range(26):
                    if freq[l] != 0:
                        max_freq = max(max_freq, freq[l])
                        min_freq = min(min_freq, freq[l])
                ans += (max_freq - min_freq)
        return ans
```

```
// C++
// Optimal Solution
class Solution {
public:
    int beautySum(string s) {
        int ans = 0;
        for (int i = 0; i < s.size(); i++) {
            int freq[26] = {0};
            for (int j = i; j < s.size(); j++) {
                int maxi = INT_MIN;
                int mini = INT_MAX;
                freq[s[j] - 'a']++;
                for (int k = 0; k < 26; k++) {
                    if (freq[k] != 0) {
                        maxi = max(maxi, freq[k]);
                        mini = min(mini, freq[k]);
                    }
                }
                ans += (maxi - mini);
            }
        }
        return ans;
    }
};
```

Template

- Approach
    - Brute-force
        -
        - Time Complexity: $O(n^3)$
        - Space Complexity: $O(1)$
    - Better
        -

        - Time Complexity: $O(n^3)$
        - Space Complexity: $O(1)$
    - Optimal
        -
        - Time Complexity: $O(n^3)$
        - Space Complexity: $O(1)$

```
# Python3
# Brute-force Solution
```

```
# Python3
# Better Solution
```

```
# Python3
# Optimal Solution
```

```
// C++
// Optimal Solution
```