Strings: Introduction

Authored by aparna_sridhar

Overview

Traditionally, a *string* is a sequence of characters, either a literal constant or a variable. A string variable might allow its elements to be mutated and its length changed, or it may be fixed (after creation).

Some languages provide strings as a built-in datatype (like C++, Java, C#) whereas others implement strings as an array of characters (Like C).

Substring

A *substring* can be defined as a contiguous slice of another string. A substring of a string S is another string S' that occurs "in" S.

Formally, a substring is the part of a string $S[i \ldots j]$ such that $i \leq j$.

For example: String $S = \texttt{"abc"}$ contains following substrings:

$$S = \texttt{"abc"}$$

$$\textbf{Substrings : } \texttt{"a"}, \texttt{"b"}, \texttt{"c"}, \texttt{"ab"}, \texttt{"bc"}, \texttt{"abc"}$$

Subsequence

A *sequence* in mathematics is an ordered collection of objects. Unlike sets, they are ordered and can have repeated elements. For example,

$$\langle A, B, C, D, E, F, A \rangle \textbf{ is a sequence}$$

A *subsequence* in mathematics is a sequence that can be derived from another sequence by deleting some elements without changing the order of the remaining elements. For example,

$$\langle A, B, D \rangle \textbf{ is a subsequence of } \langle A, B, C, D, E, F \rangle$$

- MCQS

Question 1

How many nonempty substrings and subsequences, respectively, of a string of length n are there? (Assume that 2 equal substrings/subsequences occurring at different positions are different).

- ○ $n^2, 2^n$
- ○ $\frac{n \times (n-1)}{2}, 2^n - 1$
- ✔ $\frac{n \times (n+1)}{2}, 2^n - 1$
- ✖ $\frac{n \times (n+1)}{2}, 2^n$

Go to Top

Palindromes

If a certain string reads the same when written backwards as it does reading forwards then it is called a *palindromic* string.

More formally, for a string $S[0, 1, \cdots, n-1]$, if $S[i] = S[n-i-1]$ for all i from 0 to $n-1$, then S is a palindrome or a palindromic string.

Example: **"malayalam"**
Not a palindrome: **"malaylam"**

Anagrams

Two strings are said to be *anagrams* of each other if the letters of one can be rearranged such that it reads exactly like the first one.

Example: **"TOMMARVOLORIDDLE"** and **"IAMLORDVOLDEMORT"**

- MCQS

Question 1

How many distinct palindromes of length n can you form if your alphabet has a characters? ($\lfloor x/y \rfloor$ denotes *floor division*.)

- ○ $a^{\lfloor n/2 \rfloor}$
- ○ a^n
- ✔ $a^{\lfloor (n+1)/2 \rfloor}$
- ✖ $a^{\lfloor (n-1)/2 \rfloor}$

Ordering in Strings

One of the key concepts involved in comparing two strings is the *Lexicographic Order* of strings. Basically, given 2 strings: S and T, S is said to be lexicographically smaller than T if S would appear earlier in a dictionary containing both S and T.

From here on, smaller and larger would mean lexicographically smaller and larger, respectively. Hence, **apple** is lexicographically smaller than **boy** because it appears earlier in the dictionary.

Formally, S is smaller than T if one of the following holds:

1. S is a prefix of T.
2. There is a position k such that $S[k] < T[k]$ and $S[i] = T[i]$ *for all* $i < k$ and $k \leq \min(\text{length}(S), \text{length}(T))$

Also, comparison of characters is done on the basis of their appearance in the alphabet.

For example: '**a**' is the smallest since it appears first in the dictionary. In most programming languages, the ASCII values of the characters are taken for their comparison. Hence, '**Z**' is smaller than '**a**'. An empty string (or a subsequence of length 0) is the smallest string in the lexiographic order.

- MCQS

Question 1

What is the correct lexicographical ordering of these strings: **arm**, **arts**, **armor**?

Go to Top

- arm, arts, armor
- ✔ arm, armor, arts
- arts, armor, arm
- arts, arm, armor
- armor, arm, arts
- armor, arts, arm

Implementation of Strings

Strings in C

Strings are not available in **C**. Instead, we use a char array to read strings, and the end of string is marked with the special character `\0` which is often called a null character.

When we have a char arr[] in C and want to iterate over the characters with a loop like this:

```
for (int i = 0; i < strlen(arr); i++) {
    printf("%c", arr[i]);
}
```

The complexity of a normal *for* loop that goes through a string of length N is $O(N)$. The loop above has a termination condition of `i < strlen(arr)`, where the `strlen` operation has a complexity of $O(N)$. Since the termination condition of the *for* loop is evaluated N times, the complexity of overall loop increases to $O(N^2)$. A better implementation would be as follows were `strlen` operation is performed only one time before the loop starts.

```
int len = strlen(arr);
for (int i = 0; i < len; i++){
    printf("%c", arr[i]);
}
```

However, for the particular task of printing a string, we recommend using the `%s` format specifier for printf, like this:

```
printf("%s", arr);
```

Strings in C++

Strings in C++ can be easily declared, initialized and received as input as:

```
string s, t;
s = "something predefined";
cin >> t;
```

To access or modify the character at the i^{th} position of the string, one can simply write something like this:

```
#include<bits/stdc++.h>
using namespace std;

int main() {
    string s = "string";
    cout << s[2] << "\n";
    s[3] = '9';
    cout << s << "\n";
    return 0;
}
```

Table

Overv

Substi

Subse

Palind

Anagr

Order

Imple

Practi

Go to Top

Output

Run

To get the length, one might use the `length()` function which returns an integer: the length of the string.
One can also sort and reverse strings like vectors.

```cpp
#include <bits/stdc++.h>
using namespace std;

int main() {
    string s = "string";
    cout << s.length() << "\n";
    reverse(s.begin(), s.end());
    cout << s << "\n";
    sort(s.begin(), s.end());
    cout << s << "\n";
    return 0;
}
```

Output

Run

Strings in Python

There is a very big difference between C++ strings and Python strings.
Python strings are *immutable*, meaning they cannot be modified once assigned.
However, it's still possible to append new characters at the end using the " += " operator.

```python
s = "string"
print len(s), s[2]
s += "abc"
print s
```

Output

Run

But the below code will produce an error:

```
s = "string"
s[3] = 'z'
```

Hence, strings in Python are immutable while strings in C++ are mutable.

Strings in Java

String in Java are also *immutable* meaning they cannot be modified once assigned.

When you're modifying a string a lot, it can get pretty costly in time and space to repeatedly recreate minor variants of the same string over and over. So what do you do when you have a real use case for a mutable string? Some languages have additional classes that allow you to manipulate strings. For example, Java's *String* class is immutable, but you can use StringBuffer or StringBuilder for scenarios where you truly need a mutable sequence of characters.

Strings in Ruby

1. Strings in Ruby are mutable, and they can be modified once assigned.

Go to Top

2. It is possible to edit a character at a particular location in the string.

3. It is also possible to append new characters at the end of the string using the " += " operator.

```
str = "Hello"
# Changing a character at a particular location in the string
str[2] = 'a'
puts str
# Appending new characters at the end of the string
str += " Good Morning"
puts str
#Trying to look for a regular expression match
puts str.match(/G.*?d/)
#This will split the string, on space characters by default
p str.split
```

Output

Run

Practice

Solve this easy challenge to verify your understanding of strings.