BCS 370 Non-formal Recursive Algorithm Writing Illustrated

10/23/19
gz

Let's take a look at the recursive function designs of reversing a string as an example to illustrate how to write the (non formal) base case and (non formal) general case of a recursive algorithm.

The problem/task: Using recursive function to reverse a string.

Solution 1: To display the string backward, in other words, to display the last character in the string first, then the second from the last…the first character in the string will displayed the last.

Base case: F(-1) => return

General case: F(n) => display string[n], F(n-1),

where n is the index number of the string

```
//tail recursion, no auxiliary space overhead
//T(n) = 3n = O(n)
//S(n) = n = O(n)
void reverse(string str, int index)  //this approach simply print out the reversed string one char at a time.
{
   if(index == -1) //…T(n) = n (comparison)
   {
      return;
   }
   cout << str[index];  //… T(n) = n (operation)
   reverse(str,index-1); //… T(n) = n (function call)

}
```

Solution 2: To display the string backward, in other words, to display the last character in the string first, then the second from the last…the first character in the string will be displayed the last

Base case: F(-1) => return reversed string

General case: F(n) => add string[n] to the revered string, F(n-1),

where n is the index number of the string

```
//tail recursion, no auxiliary space overhead
//function uses additional string to hold the reversed chars
//T(n) = 3n = O(n)
```

//S(n) = 2n = O(n) //two strings

```
string reverse(string str, int index, string newStr) // this approach
actually construct a new string. Has more space overhead, but solution
is more modular, its return can be reused.
{
    if(index < 0) //… T(n) = n (comparison operation)
      return newStr;
    else
    {
      newStr += str[index]; ////… T(n) = n (arithmetic operation)
      return reverse(str,index-1,newStr);// //… T(n) = n (function call)
    }

}
```

Solution 3: Swap two characters with the corresponding symmetric indices (one from the beginning, the other from the end)

Base case: F(n/2) => return reversed string

General case: F(n) => add string[n] to the revered string, F(n+1),

where n is the index number

//again //tailed recursion, no auxiliary space overhead
//T(n) = 5n = O(n)
//S(n) = 2n +1 = O(n) //two strings + temp char

```
string reverse(string str, int size, int index) {
   if(index >= size/2) //…T(n) = n (comparison)
      return str;
   else {
      char temp = str[index];  //…T(n) = n (assignment)
      str[index] = str[size-1-index]; //…T(n) = n (assignment)
      str[size-1-index] = temp; //…T(n) = n (assignment)
      return reverse(str,size,index+1); //.. T(n) = n (function call)

   }

}
```

The complete code: https://onlinegdb.com/ByIiQoAtr

***Auxiliary Space** is the extra space or temporary space used by an algorithm. Space Complexity of an algorithm is total space taken by the algorithm with respect to the input size. Space complexity includes both Auxiliary space and space used by input.