Recursion

Monday, September 28, 2020 12:31 PM

Palindrome string = a string that has the same characters from both ends

input: a string: "civic" (or noon, reviver)
output: true if if is is a palindrome, false otherwise

Algorithm:

c == c	compare first and last letter, if same, increment front, decrement back
i == i	compare first and last letter, if same, increment front, decrement back
V check	Return true

… until there is one or none letters in between front and back

Recursive algorithm:

Base case - simplest case
If size remaining characters is less than or equal to 1
    Return true
Else
    Recursive case - solve a small part, call function again on the remaining part
    compare first and last letter,
    if same,
        increment front, decrement back
        Call function on the remaining string
        Return value from the call
    Else
        Return false

Maximum value of an array

```
array = [3,5,8,2,3,8], n = 6

max(array,n)

if n == 1
    return array[0]
if n > 1
    max_temp = max(array,n-1)
    if array[n-1] > max_temp
        return array[n-1]
    else
        return max_temp
```

max(array,6)	n= 6	3,5,8,2,3,8	n-1 = 5	array[5] = 8	max_temp = max(8,8)	return 8
max(array,5)	n = 5	3,5,8,2,3	n-1 = 4	array[4] = 3	max_temp= max(3,8)	return 8
max(array,4)	n= 4	3,5,8,2	n-1 = 3	array[3] = 2	max_temp = max(2,8)	return 8
max(array,3)	n=3	3,5,8,	n-1 = 2	array[2] = 8	max_temp = max(8,5)	return 8

max(array,2)	n = 2	3,5	n-1 = 1	array[1] = 5	max_temp = max(5,3)	return 5
max(array,1)	n=1	3	n-1 = 0	array[0] = 3		return 3

Erase a character from a string:

Input: string:"Original string", erase character (erase_ch), 'i', index i - which character is next
Output: string: "Orgnal strng"

Algorithm: 1 check character at index i,
2. if equal to erase_ch, don't copy in the result string,
if not copy it in result string, i++
Repeat 1,2 until no more characters in the string

Recursive algorithm: str, erase_ch, i

Base case:
If i == length of str
Return ""
Recursive case:
Else: string result ="";
check character at index I
If str[i] != erase_ch
result += str[i]
Result += recursive call (str, erase_ch, i++)
Return result

str = "Origin" // al string"
erase_ch = 'i'

Replace(str, 'i', i=0)	Recursive case	str[0] = 'O'	result = 'O' + replace(str,'i', 1)	'Orgn'
replace(str,'i', i= 1)	Recursive case	str[1] = 'r'	Result = 'r'+ replace(str,'i', 2)	'rgn'
replace(str,'i', i=2)	Recursive case	str[2] = 'i'	Result = replace(str,'i',3)	'gn'
replace(str,'i',i =3)	Recursive case	str[3] = 'g'	Result = 'g'+ replace(str,'i',4)	'gn'
replace(str,'i',i = 4)	Recursive case	str[4] = i	Result = replace(str,'i',5)	'n'
replace(str,'i',i = 3)	Recursive case	str[5] = n	Result = n + replace(str,'i',6)	'n'
replace(str,'i',i = 6)	Base case		Return ""	

Binary search recursively

search(array, start, end, value) - look in the middle, then either stop if you found the value or search in one half ->
search(array, start, middle-1, value) or search(array, middle+1, end, value)

base case:
if array has zero items left return -1 - not found
check for zero items: compare start and end values: if start> end then stop searching, value not found.

```
binary_search_rec(int array[], int start, int end, int value)
{
    if (start > end)
        return -1; // item not found
    else
    {
        int middle = (start+end)/2;
        if (array[middle] == value)
            return middle;
        else if (array[middle] > value) // search in the first half
            return binary_search(array, start, middle-1);
        else // search in the second half
            return binary_search(array,middle+1, end);
    }
}
```

Fibonacci sequence = not efficient recursion

0,1,1,2,3,5,8, ...

F(n) = F(n-1) + F(n-2)

F(0) = 0, F(1) = 1

```
void fibonacci(int n) // return the nth item in the Fibonacci sequence
{
    if (n == 0 || n == 1)
        return n;
    else
    {
        return fibonacci(n-1) + fibonacci(n-2);
    }
}
```

n = 5

fib(5)	fib(4) = 3	fib(3) = 2	return 5
fib(4)	fib(3) = 2	fib(2) = 1	return 3
fib(3)	fib(2) = 1	fib(1) = 1	return 2
fib(2)	fib(1) = 1	fib(0) = 0	return 1
fib(2)	fib(1)		return 1
fib(3)	fib(2)	fib(1) = 1	return 2
fib(2)	fib(1)	fib(0)	return 1