CS208 Lab1 Practice

12312110 李轩然

DDL: Mar.2 23:59

Practice 1

Description

Trace the computation for the tower of Hanoi when `n = 3`.

Analysis

Firstly, when `n = 1`, move the disk from A to C directly.
When `n >= 2`, we consider the smaller $n-1$ disks as a whole , thus we just need to:

1. move them from A to B
2. move the largest disk from A to C
3. move the $n-1$ disks from B to C.

And when we consider the way to move $n-1$ disks, the problem is degraded to move the smaller $n-2$ disks, where recursion occurs.

C++ code

```cpp
#include <iostream>
using namespace std;

void hanoi(int n, char from_rod, char to_rod, char aux_rod) {
    if (n == 1) {
        cout << "Move disk 1 from " << from_rod << " to " << to_rod << endl;
        return;
    }
    hanoi(n - 1, from_rod, aux_rod, to_rod);
    cout << "Move disk " << n << " from " << from_rod << " to " << to_rod << 
endl;
    hanoi(n - 1, aux_rod, to_rod, from_rod);
}

int main()
{
```

```
    int n = 3;
    hanoi(n, 'A', 'C', 'B');
    return 0;
}
```

Sample

Output:

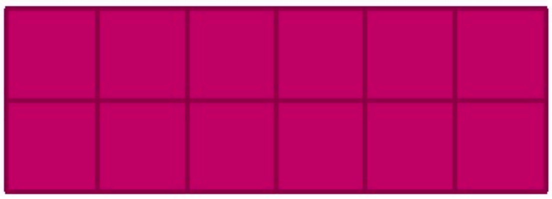
```
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
```

Practice 2

Description

Find the number of ways a `2 * n` rectangle can be tiled with rectangular tiles of size `2 * 1`.

2*1



2*6

Analysis

Firstly, when `n = 1`, we only have one way to tile the `2 * 1` rectangle, so $count(1) = 1$.
When `n = 2`, we have two ways which are horizontal and vertical, so $count(2) = 2$.
Now we consider the ways to fill when `n >= 3`.We have two options:

1. fill the `2 * (n - 2)` rectangle then tile two horizontal `2 * 1` rectangle
2. fill the `2 * (n - 1)` rectangle then tile a vertical `2 * 1` rectangle

So the number of ways to fill the `2 * n` rectangle will be:

$$count(n) = \begin{cases} 1, & n = 1 \\ 2, & n = 2 \\ count(n-1) + count(n-2), & n >= 3 \end{cases}$$

and it is smiliar with the **Fibonacci sequence**.

C++ code:

```
#include <iostream>
using namespace std;

int count(int n) {
    if (n <= 2) return n;
    return count(n - 1) + count (n - 2);
}

int main()
{
    int n;
    cin >> n;

    if (n < 1) cout << "Invalid input!";
    else {
        int cnt = count(n);
        cout << cnt;
    }

    return 0;
}
```

Sample

Input:

```
6
```

Output:

```
13
```

Practice 3 (optional)

Description

Enter a string and print out all permutations of the characters in the string.

```
Example:
Input: abc
Output: abc, acb, bac, bca, cab, cba
```

Analysis

We use **backtrack method**. Suppose $nums$ is the original array, $path$ is the arrangement being built, $used$ is a boolean array of which elements have been used.
We call the recursive function and iterate over each element in $nums$: If it has not been used (`used[i] == false`), add it to $path$ and let `used[i] = true`, then keep iterating until the length of $path$ equals to the length of $nums$, thus we have a complete permutation $path$.
After that, we return the recursive function, undo the selection (remove the element from $path$, and `used[i] = false`), and back to the last situation.

No code needed

Practice 4 (optional)

Description

Enter a string and print out all combinations of the characters in the string.

```
Example:
Input: abc
Output: a, b, c, ab, bc, ac, abc
```

Analysis

Suppose the length of the string is n. For each character of the string, a binary 0 is used to indicate no selection and a 1 is used to indicate selection, so that **each binary number whose number of bits does not exceed n represents a different combination**. Therefore, all combinations can be obtained by looping from 1 to $2^n - 1$ and performing a bitwise operation on each number to determine whether the character is selected or not.

No code needed