【回溯】-第N个排列题目描

述与示例

给定参数 `n` ，从 `1` 到 `n` 会有 `n` 个整数 `1，2，3，...，n` 。这 `n` 个数字共有 `n!` 种排列，按大小顺序升序列出所有排列情况，并一一标记。当 `n = 3` 时，所有排列如下： `"123","132","213","231","312","321"` 。

给定 `n` 和 `k` 返回第 `k` 个排列。

输入

第一行为 `n`
第二行为 `k`
n 的范围是 `1 ~ 9`
k 的范围是 `1 ~ n!`

输出

输出排列第 `k` 位置的数字

示例一

输入

```
3
3
```

输出

```
213
```

示例二

输入

```
2
2
```

输出

```
21
```

解题思路

这道题本质上是一道**排列类型**的回溯问题。具体过程和LC46. 全排列几乎完全一致。

注意本题可以进行剪枝操作，即无需计算所有排列，只需要计算前 `k` 个排列即可。

代码

Python

```python
# 题目：2024E-第N个排列
# 分值：200
# 作者：许老师-闭着眼睛学数理化
# 算法：回溯
# 代码看不懂的地方，请直接在群上提问


# 输入n和k
n = int(input())
k = int(input())

# 用于记录当前已经得到了几个全排列的变量cnt
cnt = 0
ans = ""

usedList = [False] * (n+1)

def dfs(path, usedList, k, n):
```

```python
    # cnt和ans均需要声明为全局变量
    global cnt, ans
    # 如果path的长度等于n，那么得到了一个全排列
    if len(path) == n:
        # 已获得的全排列的数目+1
        cnt += 1
        # 如果已经获得了第k个排列，那么得到了答案
        if cnt == k:
            ans = "".join([str(num) for num in path])
        return
    # 如果path的小于n，进行递归
    else:
        for i in range(1, n+1):
            # 如果i尚未使用过，那么可以进行递归
            if usedList[i] == False:
                # 状态更新：usedList[i]改为True，将i加入当前path
                usedList[i] = True
                path.append(i)
                # 回溯
                dfs(path, usedList, k, n)
                # 回滚：usedList[i]改为False，将i从当前path中删除
                usedList[i] = False
                path.pop()
                # 这里可以进行剪枝，虽然可以不加，但是尽量加上
                # 即如果已经找到了第k个排列，那么无需进行后续的回溯
                if cnt == k:
                    return

# 调用递归函数的入口，最开始path为空列表
dfs([], usedList, k, n)
print(ans)
```

Java

```java
import java.util.Scanner;

public class Main {
    // 用于记录当前已经得到了几个全排列的变量cnt
    static int cnt = 0;
    static String ans = "";

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```java

        // 输入n和k
        int n = scanner.nextInt();
        int k = scanner.nextInt();

        // 初始化usedList数组，用于记录哪些数字已使用过
        boolean[] usedList = new boolean[n + 1];

        // 调用递归函数的入口，最开始path为空列表
        dfs(new StringBuilder(), usedList, k, n);
        System.out.println(ans);
    }

    // 定义递归回溯函数
    public static void dfs(StringBuilder path, boolean[] usedList, int k, int n) {
        // 如果path的长度等于n，那么得到了一个全排列
        if (path.length() == n) {
            // 已获得的全排列的数目+1
            cnt++;

            // 如果已经获得了第k个排列，那么得到了答案
            if (cnt == k) {
                ans = path.toString();
            }
            return;
        }

        // 如果path小于n，进行递归
        for (int i = 1; i <= n; i++) {
            // 如果i尚未使用过，那么可以进行递归
            if (!usedList[i]) {
                // 状态更新：usedList[i]改为True，将i加入当前path
                usedList[i] = true;
                path.append(i);

                // 递归调用
                dfs(path, usedList, k, n);

                // 回滚：usedList[i]改为False，将i从当前path中删除
                usedList[i] = false;
                path.deleteCharAt(path.length() - 1);

                // 剪枝：如果已经找到了第k个排列，无需进行后续的回溯
                if (cnt == k) {
                    return;
                }
```

```
            }
        }
    }
}

```

C++

```cpp
#include <iostream>
#include <vector>
#include <string>

using namespace std;

// 用于记录当前已经得到了几个全排列的变量cnt
int cnt = 0;
string ans = "";

// 定义递归回溯函数
void dfs(vector<int>& path, vector<bool>& usedList, int k, int n) {
    // 如果path的长度等于n，那么得到了一个全排列
    if (path.size() == n) {
        // 已获得的全排列的数目+1
        cnt++;

        // 如果已经获得了第k个排列，那么得到了答案
        if (cnt == k) {
            for (int num : path) {
                ans += to_string(num);
            }
        }
        return;
    }

    // 如果path小于n，进行递归
    for (int i = 1; i <= n; i++) {
        // 如果i尚未使用过，那么可以进行递归
        if (!usedList[i]) {
            // 状态更新：usedList[i]改为True，将i加入当前path
            usedList[i] = true;
            path.push_back(i);

            // 递归调用
```

```
            dfs(path, usedList, k, n);

            // 回滚：usedList[i]改为False，将i从当前path中删除
            usedList[i] = false;
            path.pop_back();

            // 剪枝：如果已经找到了第k个排列，无需进行后续的回溯
            if (cnt == k) {
                return;
            }
        }
    }
}

int main() {
    // 输入n和k
    int n, k;
    cin >> n >> k;

    // 初始化usedList数组，用于记录哪些数字已使用过
    vector<bool> usedList(n + 1, false);

    // 初始化path列表
    vector<int> path;

    // 调用递归函数的入口，最开始path为空列表
    dfs(path, usedList, k, n);

    // 输出答案
    cout << ans << endl;

    return 0;
}

```

时空复杂度

时间复杂度：`O(k)`。需要计算得到前`k`种排列。

空间复杂度：`O(N)`。`usedList`所占空间。