

【回溯】-第N个排列题目描述

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

```
1 3
2 3
```

输出

```
1 213
```

示例二

输入

```
1 2
2 2
```

输出

```
1 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

```
1 # 题目：2024E-第N个排列
2 # 分值：200
3 # 作者：许老师-闭着眼睛学数理化
4 # 算法：回溯
5 # 代码看不懂的地方，请直接在群上提问
6
7
8 # 输入n和k
9 n = int(input())
10 k = int(input())
11
12 # 用于记录当前已经得到了几个全排列的变量cnt
13 cnt = 0
14 ans = ""
15
16 usedList = [False] * (n+1)
17
18 def dfs(path, usedList, k, n):
```

```

19     # cnt和ans均需要声明为全局变量
20     global cnt, ans
21     # 如果path的长度等于n, 那么得到了一个全排列
22     if len(path) == n:
23         # 已获得的全排列的数目+1
24         cnt += 1
25         # 如果已经获得了第k个排列, 那么得到了答案
26         if cnt == k:
27             ans = "".join([str(num) for num in path])
28         return
29     # 如果path的小于n, 进行递归
30     else:
31         for i in range(1, n+1):
32             # 如果i尚未使用过, 那么可以进行递归
33             if usedList[i] == False:
34                 # 状态更新: usedList[i]改为True, 将i加入当前path
35                 usedList[i] = True
36                 path.append(i)
37                 # 回溯
38                 dfs(path, usedList, k, n)
39                 # 回滚: usedList[i]改为False, 将i从当前path中删除
40                 usedList[i] = False
41                 path.pop()
42                 # 这里可以进行剪枝, 虽然可以不加, 但是尽量加上
43                 # 即如果已经找到了第k个排列, 那么无需进行后续的回溯
44                 if cnt == k:
45                     return
46
47 # 调用递归函数的入口, 最开始path为空列表
48 dfs([], usedList, k, n)
49 print(ans)

```

Java

```

1 import java.util.Scanner;
2
3 public class Main {
4     // 用于记录当前已经得到了几个全排列的变量cnt
5     static int cnt = 0;
6     static String ans = "";
7
8     public static void main(String[] args) {
9         Scanner scanner = new Scanner(System.in);

```

```
10
11     // 输入n和k
12     int n = scanner.nextInt();
13     int k = scanner.nextInt();
14
15     // 初始化usedList数组, 用于记录哪些数字已使用过
16     boolean[] usedList = new boolean[n + 1];
17
18     // 调用递归函数的入口, 最开始path为空列表
19     dfs(new StringBuilder(), usedList, k, n);
20     System.out.println(ans);
21 }
22
23 // 定义递归回溯函数
24 public static void dfs(StringBuilder path, boolean[] usedList, int k, int
n) {
25     // 如果path的长度等于n, 那么得到了一个全排列
26     if (path.length() == n) {
27         // 已获得的全排列的数目+1
28         cnt++;
29
30         // 如果已经获得了第k个排列, 那么得到了答案
31         if (cnt == k) {
32             ans = path.toString();
33         }
34         return;
35     }
36
37     // 如果path小于n, 进行递归
38     for (int i = 1; i <= n; i++) {
39         // 如果i尚未使用过, 那么可以进行递归
40         if (!usedList[i]) {
41             // 状态更新: usedList[i]改为True, 将i加入当前path
42             usedList[i] = true;
43             path.append(i);
44
45             // 递归调用
46             dfs(path, usedList, k, n);
47
48             // 回滚: usedList[i]改为False, 将i从当前path中删除
49             usedList[i] = false;
50             path.deleteCharAt(path.length() - 1);
51
52             // 剪枝: 如果已经找到了第k个排列, 无需进行后续的回溯
53             if (cnt == k) {
54                 return;
55             }
56         }
57     }
58 }
```

```
56         }
57     }
58 }
59 }
60
```

C++

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4
5  using namespace std;
6
7  // 用于记录当前已经得到了几个全排列的变量cnt
8  int cnt = 0;
9  string ans = "";
10
11 // 定义递归回溯函数
12 void dfs(vector<int>& path, vector<bool>& usedList, int k, int n) {
13     // 如果path的长度等于n, 那么得到了一个全排列
14     if (path.size() == n) {
15         // 已获得的全排列的数目+1
16         cnt++;
17
18         // 如果已经获得了第k个排列, 那么得到了答案
19         if (cnt == k) {
20             for (int num : path) {
21                 ans += to_string(num);
22             }
23             return;
24         }
25     }
26
27     // 如果path小于n, 进行递归
28     for (int i = 1; i <= n; i++) {
29         // 如果i尚未使用过, 那么可以进行递归
30         if (!usedList[i]) {
31             // 状态更新: usedList[i]改为True, 将i加入当前path
32             usedList[i] = true;
33             path.push_back(i);
34
35             // 递归调用
```

```

36         dfs(path, usedList, k, n);
37
38         // 回滚: usedList[i]改为False, 将i从当前path中删除
39         usedList[i] = false;
40         path.pop_back();
41
42         // 剪枝: 如果已经找到了第k个排列, 无需进行后续的回溯
43         if (cnt == k) {
44             return;
45         }
46     }
47 }
48 }
49
50 int main() {
51     // 输入n和k
52     int n, k;
53     cin >> n >> k;
54
55     // 初始化usedList数组, 用于记录哪些数字已使用过
56     vector<bool> usedList(n + 1, false);
57
58     // 初始化path列表
59     vector<int> path;
60
61     // 调用递归函数的入口, 最开始path为空列表
62     dfs(path, usedList, k, n);
63
64     // 输出答案
65     cout << ans << endl;
66
67     return 0;
68 }
69

```

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。