题目描述

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的情况，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。

输入描述

输入两行，第一行为n，第二行为k，

给定n的范围是[1,9],给定k的范围是[1,n!]。

输出描述

输出排在第k位置的数字。

用例

输入	3 3
输出	213
说明	3的排列有123,132,213…,那么第三位置就是213

输入	2 2
输出	21
说明	2的排列有12,21，那么第二位置的为21。

题目解析

通过上面n=3的全排列可以分析，以1开头、以2开头、以3开头的排列个数各有两个，因为固定开头为1的，则其排列情况就是n=2的排列情况，即有两个23、32。

因此以1开头的排列个数有 2!个，以2，3开头的排列个数求解同理。

因此我们要求n=3的第k个排列，完全可以推断出第k个排列的开头数字是几。

比如n=3的第1个和第2个排列的开头数字prefix就是1，第3、4个排列的开头数字prefix就是2，第5、6个排列的开头数字prefix就是3。

推导一下，可得公式：Math.ceil(k / (n-1)!)

但是这里我不想根据k、n来直接推导出排列的开头数字prefix，因为这个逻辑不适用于后面的递归，我们会将组成排列的数字按大小升序存入一个数组 arr中，比如n=3的排列元素为 arr = [1,2,3]，此时我们要求n=3的第k个排列的开头数字，公式为：arr [Math.floor((k-1) / (n-1)!)]

知道了第k个排列的开头数字prefix后，我们就可以缩小排列的查找范围，比如n=3，k=3的排列查找就可以在以下范围中查找：

- 213
- 231

而此时k=3值就不适用了，因为k=3是相对于下面情况的

- 123
- 132
- 213
- 231
- 312
- 321

我们需要将k值转换为1，转换公式如下：

newK = k % (n-1)!

但是这个公式不普适，比如n=3,k=4时，经过上面公式转换newK就变为0，而实际上newK应该为2,因此我们需要附加判断

newK === 0 ? (n-1)! : newK

此时就完成了大问题

> n=3, k=3, arr=[1,2,3]

的简化，简化为了

> n=2, k=1 ,arr=[1,3] 且 prefix=2

因此我们可以使用递归来解决此题，而当k=1时，表示当前arr=[1,3]的排列就是需要的排列，因此最终要找的排列就是 prefix + arr.join('') = 213。

因此 k =1就是递归的出口条件。

Java算法源码

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main {
  static int[] fact;

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

    int n = sc.nextInt();
    int k = sc.nextInt();

    fact = new int[n + 1];
    fact[1] = 1;
    for (int i = 2; i <= n; i++) {
      fact[i] = fact[i - 1] * i;
    }

    int[] arr = new int[n];
    for (int i = 0; i < n; i++) arr[i] = i + 1;

    System.out.println(getNK(n, k, arr));
  }

  public static String getNK(int n, int k, int[] arr) {
    if (n == 1) return "1";

    int f = fact[n - 1];
    int prefix = arr[(k - 1) / f];
    k %= f;
    k = k == 0 ? f : k;

    arr = Arrays.stream(arr).filter(ele -> ele != prefix).toArray();

    if (k == 1) {
      StringBuilder sb = new StringBuilder();
      for (int v : arr) sb.append(v);
      return prefix + sb.toString();
    } else {
      return prefix + getNK(n - 1, k, arr);
    }
  }
}
```

JS算法源码

```
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
let factorial = [0, 1];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    let [n, k] = lines.map((ele) => parseInt(ele));

    for (let i = 2; i <= n; i++) {
      factorial[i] = factorial[i - 1] * i;
    }

    let arr = new Array(n).fill(0).map((_, index) => index + 1);
    console.log(getNK(n, k, arr));

    lines.length = 0;
  }
});

```

```
/* 算法 */
function getNK(n, k, arr) {
  if (n === 1) {
    return "1";
  }

  let f = factorial[n - 1];
  let prefix = arr[Math.floor((k - 1) / f)];
  k = k % f;
  k = k === 0 ? f : k;

  arr = arr.filter((ele) => ele !== prefix);

  if (k === 1) {
    return prefix + arr.join("");
  } else {
    return prefix + getNK(n - 1, k, arr);
  }
}
```

Python算法源码

```python
# 输入获取
n = int(input())
k = int(input())
arr = [i + 1 for i in range(n)]

fact = [0] * (n + 1)
fact[1] = 1
for i in range(2, n + 1):
    fact[i] = fact[i - 1] * i


# 算法入口
def getResult(n, k, arr):
    if n == 1:
        return "1"

    f = fact[n - 1]
    prefix = arr[(k - 1) // f]
    k %= f
    k = f if k == 0 else k

    arr = list(filter(lambda x: x != prefix, arr))

    if k == 1:
        return str(prefix) + "".join(map(str, arr))
    else:
        return str(prefix) + getResult(n - 1, k, arr)


# 算法调用
print(getResult(n, k, arr))
```

解法二：不用递归

Java算法源码

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main {
  static int[] fact;

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

    int n = sc.nextInt();
    int k = sc.nextInt();

    fact = new int[n + 1];
    fact[1] = 1;
    for (int i = 2; i <= n; i++) {
      fact[i] = fact[i - 1] * i;
    }

    int[] arr = new int[n];
    for (int i = 0; i < n; i++) arr[i] = i + 1;

    System.out.println(getResult(n, k, arr));
  }

  public static String getResult(int n, int k, int[] arr) {
    if (n == 1) return "1";

    StringBuilder sb = new StringBuilder();

    while (true) {
      int f = fact[n - 1];
      int prefix = arr[(k - 1) / f];
      sb.append(prefix);

      k = k % f;
      if (k == 0) k = f;

      arr = Arrays.stream(arr).filter(ele -> ele != prefix).toArray();
      n--;
      if (k == 1) {
        for (int v : arr) sb.append(v);
        break;
      }
    }

    return sb.toString();
  }
}
```

JS算法源码

```
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    let [n, k] = lines.map((ele) => parseInt(ele));

    console.log(getPermutation(n, k));

    lines.length = 0;
  }
});

function getPermutation(n, k) {
  if (n === 1) return "1";

  // 记录n!排列组合需要的成员
  let arr = new Array(n).fill(0).map((_, idx) => idx + 1);

  // 记录阶乘值
  let factorial = [0, 1];
  for (let i = 2; i <= n; i++) {
    factorial[i] = factorial[i - 1] * i;
  }

```

```
  let res = "";

  while (true) {
    // 第k个排列的开头数字prefix
    let prefix = arr[Math.floor((k - 1) / factorial[n - 1])];
    res += prefix;
    // 对应排列在开头为prefix的排列中的序号
    k = k % factorial[n - 1];
    if (k === 0) {
      k = factorial[n - 1];
    }

    // 开头为prefix的排列所属的成员
    arr = arr.filter((ele) => ele !== prefix);
    n--;
    if (k === 1) {
      res += arr.join("");
      break;
    }
  }
  return res;
}
```

Python算法源码

```python
# 输入获取
n = int(input())
k = int(input())
arr = [i + 1 for i in range(n)]

fact = [0] * (n + 1)
fact[1] = 1
for i in range(2, n + 1):
    fact[i] = fact[i - 1] * i


# 算法入口
def getResult(n, k, arr):
    if n == 1:
        return "1"

    res = []

    while True:
        prefix = arr[(k - 1) // fact[n-1]]
        res.append(str(prefix))

        k = k % fact[n-1]
        if k == 0:
            k = fact[n-1]

        arr = list(filter(lambda x: x != prefix, arr))
        n -= 1
        if k == 1:
            res.append("".join(map(str, arr)))
            break

    return "".join(res)


# 算法调用
print(getResult(n, k, arr))
```