265 AI处理器组合

题目描述

某公司研发了一款高性能AI处理器，每台物理设备具备8颗AI处理器，编号分别为0、1、2、3、4、5、6、7。

编号0~3的处理器处于同一链路中，编号4~7的处理器处于另外一个链路中，不同链路中的处理器不能通信，如下图所示。

现给定服务器可用的处理器编号数组`array`，以及任务申请的处理器数量`num`，找出符合下列亲和性调度原则的芯片组合，如果不存在符合要求的组合，则返回空列表。

亲和性调度原则：

- 如果申请处理器个数为1，则选择同一链路，剩余可用的处理器数量为1个的最佳，其次是剩余3个的为次佳，然后是剩余2个，最后是剩余4个。
- 如果申请处理器个数为2，则选择同一链路剩余可用的处理器2个的为最佳，其余是剩余4个，最后是剩余3个。
- 如果申请处理器个数为4，则必须选择同一链路剩余可用的处理器数量为4个。
- 如果申请处理器个数为8，则申请节点所有8个处理器。

提示：

1. 任务申请的处理器数量只能是1、2、4、8。
2. 编号0~3的处理器处于一个链路，编号4~7的处理器处于另外一个链路。
3. 处理器编号唯一，且不存在相同编号处理器。

输入描述

输入包含可用的处理器编号数组array，以及任务申请的处理器数量num两个部分。
第一行为array，第二行为num。例如：

```
[0, 1, 4, 5, 6, 7]
1
```

表示当前编号为0、1、4、5、6、7的处理器可用。任务申请1个处理器

数据范围：

```
0 <= array.length <= 8
0 <= array[i] <= 7
num in [1, 2, 4, 8]
```

输出描述

输出为组合列表，当array=[0,1,4,5,6,7]、num=1时，输出为[[0], [1]]

示例一

输入：

```
[0, 1, 4, 5, 6, 7]
1
```

输出：

```
[[0], [1]]
```

说明：
根据第一条亲和性调度原则，在剩余两个处理器的链路(0,1,2,3)中选择处理器。由于只有0和1可用，则返回任意一颗处理器即可。

示例二

输入：

```
[0, 1, 4, 5, 6, 7]
4
```

输出：

```
[[4, 5, 6, 7]]
```

说明：
根据第三条亲和性调度原则，必须选择同一链路剩余可用的处理器数量为4个的环。

解题思路

1. 将输入的数组分成两组，第一组数字小于4，第二组数字大于4
2. 列出相关逻辑
    - 当num为1时，根据题意传入优先级[1,3,2,4]，遍历优先级，返回子序列列表
    - 当num为2时，根据题意传入优先级[2, 4, 3]，考虑到排列组合，使用python内置的itertools包中的combinations函数，获取子序列之后，整理成list格式返回结果列表
    - 当num为4时，根据题意如果有一组数字满足条件，返回该组的所有处理器编号
    - 当num为8时，根据题意如果两组数字都满足条件，返回所有处理器编号

解题代码

```
from itertools import combinations


def solve_method(arr, num):
    result = []

    first = [n for n in arr if n < 4]
    second = [n for n in arr if n >= 4]
```

```
    first_nums = len(first)
    second_nums = len(second)

    if num == 1:
        result.extend(cpu1(first, first_nums, second, second_nums))
    elif num == 2:
        result.extend(cpu2(first, first_nums, second, second_nums))
    elif num == 4:
        if first_nums == 4:
            result.append([n for n in range(4)])
        if second_nums == 4:
            result.append([n for n in range(4, 8)])
    elif num == 8:
        if first_nums == 4 and second_nums == 4:
            result.append([n for n in range(8)])

    return result


def cpuN(first, first_nums, second, second_nums, priority, k) -> list:
    cpus = []

    is_fit = False
    for p in priority:
        if p == first_nums:
            cpus.extend([list(n) for n in combinations(first, k)])
            is_fit = True
        if p == second_nums:
            cpus.extend([list(n) for n in combinations(second, k)])
            is_fit = True
        if is_fit:
            break

    return cpus


def cpu1(first, first_nums, second, second_nums) -> list:
    return cpuN(first, first_nums, second, second_nums, [1, 3, 2, 4], 1)


def cpu2(first, first_nums, second, second_nums) -> list:
    return cpuN(first, first_nums, second, second_nums, [2, 4, 3], 2)


if __name__ == '__main__':
    assert solve_method([0, 1, 4, 5, 6, 7], 4) == [[4, 5, 6, 7]]
    assert solve_method([0, 1, 4, 5, 6, 7], 1) == [[0], [1]]
    assert solve_method([0, 1, 2, 4, 5], 2) == [[0, 1], [0, 2], [1, 2], [4, 5], [4,
6], [5, 6]]
```