Algorithm 06 순열

```java
public static ArrayList<Integer[]> newChickenRecipe(int[] stuffArr, int choiceNum) {
    // TODO:
    // 사용이 가능한 재료만 넣을 변수를 선언한다.
    ArrayList<Integer> freshArr = new ArrayList<>();
    // stuffArr를 순회하며 사용 가능한 재료만 arr 리스트에 추가한다.
    for (int i = 0; i < stuffArr.length; i++) {
        // suffArr[i]를 String 타입으로 변환한다.
        String str = String.valueOf(stuffArr[i]);
        // 값을 char 타입의 배열로 바꾸고, 0이 들어간 갯수만큼 element 배열에 추가한다.
        int[] element = str.chars().filter(c -> c == '0').toArray();
        // element 배열의 숫자가 2 이하인 경우, freshArr에 해당 재료를 넣는다.
        if (element.length <= 2) freshArr.add(stuffArr[i]);
    }
    // 재료가 들어간 list를 오름차순 정렬한다.
    Collections.sort(freshArr);
    // 사용할 수 있는 재료가 없거나, 재료의 양보다 사용해야 할 갯수가 많은 경우 null을 반환한다.
    if (freshArr.size() == 0 || freshArr.size() < choiceNum) return null;
    // 결과를 담을 리스트를 선언한다.
    ArrayList<Integer[]> result = new ArrayList<>();
    // 해당 재료의 사용 여부를 확인할 배열을 선언한다.
    boolean[] visited = new boolean[freshArr.size()];
    // 순열 메서드를 사용해 모든 경우의 수를 구하고 해당 값을 반환한다.
    return permutation(choiceNum, new Integer[]{}, result, freshArr, visited, depth: 0);
}
```

[1, 10, 11000, 1111]

[0,0,0]

1 → [1] → [ ]
10 → [1, 0] → [0]
11000 → [1, 1, 0, 0, 0] → [0, 0, 0]

0이 2개 이상이면 상한 재료

예외 Case

[false, false, false]

재료를 몇 개까지 선택했는지

2

```java
public static ArrayList<Integer[]> permutation(int choiceNum, Integer[] bucket, ArrayList<Integer[]> result, ArrayList<Integer> freshArr, boolean[] visited, int depth) {
    // 사용한 재료의 수가 num에 도달하면, (재귀 종료)
    if (depth == choiceNum) {
        // result에 재료가 저장된 bucket 배열을 넣고 반환한다.
        result.add(bucket);
        return result;
    }
    // 사용할 수 있는 재료의 수 만큼 반복한다.
    for (int i = 0; i < freshArr.size(); i++) {
        // 해당 재료를 사용하지 않았다면,
        if (!visited[i]) {
            // 해당 재료의 사용 여부를 체크한다.
            visited[i] = true;
            // bucket에 사용한 재료를 넣을 새로운 배열을 선언한다.
            Integer[] concatArray = Arrays.copyOf(bucket, newLength: bucket.length+1);
            concatArray[concatArray.length - 1] = freshArr.get(i);

            // 재귀를 사용한다.
            result = permutation(choiceNum, concatArray, result, freshArr, visited, depth: depth+1);
            // 한 번 순회하고, 반복문을 다시 시작하기 위해 첫 시작 재료의 사용여부를 false로 변경한다..
            visited[i] = false;
        }
    }
    return result;
}
```

base case

사용 가능한 재료
[1, 10, 1111]
0 1 2

choiceNum = 2

freshArr	visited[i]	concatArray
[f, f, f]		
[t, f, f]	0	[1]
	1 (1≠2)	
	0	
[t, t, f]	1	[1, 10]
[t, t, t]	2	<[1, 10]> result
⋮	⋮	⋮

다시 재귀로 돌아가므로 0부터 시작