# 题目

给定两个整数 n 和 k，返回范围 [1, n] 中所有可能的 k 个数的组合。

你可以按 任何顺序 返回答案。

示例 1：

输入：n = 4, k = 2

输出：

[

[2,4],

[3,4],

[2,3],

[1,2],

[1,3],

[1,4],

]

示例 2：

输入：n = 1, k = 1

输出：[[1]]

提示：

1 <= n <= 20

1 <= k <= n

# 分析

## 方法一：回溯法

思路：

具体步骤如下：

1、创建一个数组 path 用来存储当前组合的数字。

2、创建一个函数 dfs，在该函数中进行递归回溯：

如果 path 的长度等于 k，将 path 加入结果数组中。

从 start 开始循环，将数字加入 path 中，并递归调用 dfs。

在递归结束后，将最后一个数字从 path 中移除，以便尝试下一个数字。

最终，将结果数组返回即可。

代码：

class Solution {

public:

vector<vector<int>> combine(int n, int k) {

vector<vector<int>> result;

vector<int> path;

dfs(result, path, n, k, 1);

return result;

}

private:

void dfs(vector<vector<int>>& result, vector<int>& path, int n, int k, int start) {

if (path.size() == k) {

result.push\_back(path);

return;

}

for (int i = start; i <= n; ++i) {

path.push\_back(i);

dfs(result, path, n, k, i + 1);

path.pop\_back();

}

}

};

## 方法二：递归法

class Solution {

public:

vector<vector<int>> combine(int n, int k) {

vector<vector<int>> result; // 存放最终结果的二维数组

vector<int> path; // 存放每个组合的一维数组

// 调用递归函数

combineRecursive(result, path, n, k, 1);

return result;

}

private:

void combineRecursive(vector<vector<int>>& result, vector<int>& path, int n, int k, int start) {

// 当k为0时，表示当前组合已经选够k个数字，将当前组合加入结果数组中

if (k == 0) {

result.push\_back(path);

return;

}

// 从start开始循环，选择当前数字，并递归调用下一层选择剩余数字

for (int i = start; i <= n - k + 1; ++i) {

path.push\_back(i); // 选择当前数字

combineRecursive(result, path, n, k - 1, i + 1); // 递归调用下一层，选取剩余数字

path.pop\_back(); // 撤销选择，回溯到上一状态

}

}

};