回溯算法的第一道题，就不简单呀

第77题. 组合

题目链接：https://leetcode-cn.com/problems/combinations/

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

示例:

输入: n = 4, k = 2

输出:

[

[2,4],

[3,4],

[2,3],

[1,2],

[1,3],

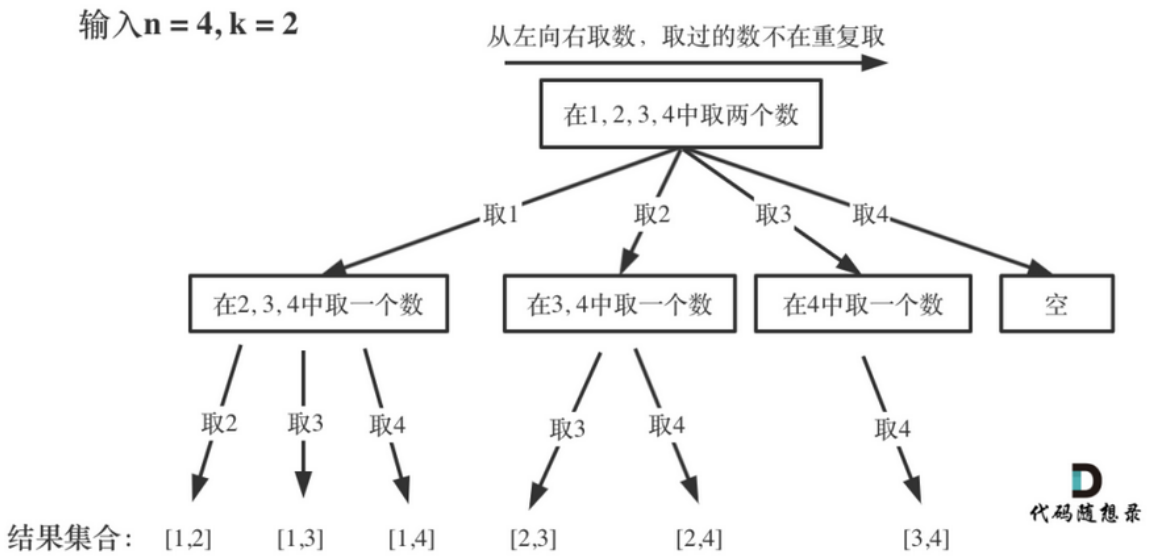
[1,4],

]

> 对于该问题如果用for循环，那么k个数则需要用k个for循环才能有答案
> 而用**回溯法**则用递归来做**层叠嵌套，每一层的递归中嵌套一个for循环，那么递归就可以用来解决多层嵌套循环的问题**

此时递归的层数大家应该知道了，例如：n为100，k为50的情况下，就是递归50层。

> 回溯算法可以用一棵树来表示如下所示：



> 这道题是一个**组合问题，无序的**，排列问题是**一个有序问题**。注意之间的区别哦
> 发现1:每次从集合中选取元素，可选择的范围随着选择的进行而收缩，调整可选择的范围

汇总分析题目

1、递归函数的返回值以及参数

> 定义两个全局变量，一个用来存放符合条件的单一结果，一个用来存放符合条件结果的集合
> 代码如下：
> vector<vector<int>> result;//存放符合条件结果的**集合**
> vector<int> path;//用来存放符合条件的结果

注意：

其实不定义这两个全局遍历也是可以的，把这两个变量放进递归函数的参数里，但函数里参数太多影响可读性，所以我定义全局变量了。

> **然后的函数参数还要有两个参数n和k均为int型参数，然后还需要一个参数，为int型变量startIndex，这个参数用来记录本层递归，集合从哪里开始遍历(集合就是[1,...,n])**

```
vector<vector<int>> result;
vector<int> path;
void backtracking(int n,int k,int startIndex);
```

2、递归终止条件

> 什么时候到达所谓的叶子结点?
> path这个数组的大小如果达到k，说明我们找到了一个**子集大小为k**的组合

```
if(path.size()==k){
 result.push_back(path);
 return ;
}
```

3、单层逻辑

```
for(int i=startIndex;i<=n;i++){//控制树的横向遍历
 path.push_back(i)//处理结点
```

```
 backtracking(n,k,i+1);//递归：控制树的纵向遍历，注意下一层搜索要从i+1开始
 path.pop_back();//回溯，撤销处理的结点
}
```

综上所述完整代码如下：

```
class Solution {
public:
 vector<vector<int>> ans;
 vector<int> path;
 void backtracking(int n,int k,int startIndex){
 if(path.size()==k){
 ans.push_back(path);
 return ;
 }
  for(int i=startIndex;i<=n;i++){//n表示可以取到的最大值
  path.push_back(i);
  backtracking(n,k,i+1);//想想组合的方法，此时只能取i+1后面的元素，其他元素可能会造成重复
  path.pop_back();
  }
  }
     vector<vector<int>> combine(int n, int k) {
  backtracking(n,k,1);
  return ans;
     }
};
```

通过剪枝优化减少时间复杂度

> 我们可以知道
> **如果for循环选择的起始位置之后的元素个数已经不足我们需要的元素个数了，那么就没有必要搜索了。**
> 剪枝的地方:一般是**放在for循环**里面

数学公式

必须满足(**放置的位置在递归前一个语法单元,此时path还没放入i**)

path.size()+n-i+1>=k

则i的取值满足**i<=n+path.size()+1-k;**

所以剪枝后的代码如下：

```
class Solution {
public:
    vector<vector<int>> ans;
    vector<int> path;
```

```
5      void backtracking(int n,int k,int startIndex){
6          if(path.size()==k){
7              ans.push_back(path);
8              return ;
9          }
10          for(int i=startIndex;i<=n+1+path.size()-k;i++){//n表示可以取到的最大值
11              path.push_back(i);
12              backtracking(n,k,i+1);//想想组合的方法，此时只能取i+1后面的元素，其他元素可能会造成重复
13              path.pop_back();
14          }
15      }
16      vector<vector<int>> combine(int n, int k) {
17          backtracking(n,k,1);
18          return ans;
19      }
20  };
21  };
```

与上面那道题相比该题目难度刚刚好

第216题.组合总和III

链接：https://leetcode-cn.com/problems/combination-sum-iii/

找出所有相加之和为 n 的 k 个数的组合。组合中只允许含有 1 - 9 的正整数，并且每种组合中不存在重复的数字。

说明：

- 所有数字都是正整数。
- 解集不能包含重复的组合。

示例 1:

输入: k = 3, n = 7

输出: [[1,2,4]]

示例 2:

输入: k = 3, n = 9

输出: [[1,2,6], [1,3,5], [2,3,4]]

这道题我们可以理解为上面道题中从1-9数里面取出k个值(不重复)，k个值相加和为target，记录满足条件组合
在回溯算法中累加可以转换为累减最后结果为0

```
class Solution {
public:
 vector<vector<int>> result;
 vector<int> path;
 void backtracking(int count,int k,int startIndex){//从1-9数里面取出k个数结果为n
 if(path.size()==k){
 if(count==0)result.push_back(path);
 return ;
 }
  for(int i=startIndex;i<=path.size()+10-k;i++){//根据第一道题的剪枝结果
  path.push_back(i);
  backtracking(count-i,k,i+1);//此时count实际上有回溯的痕迹
  path.pop_back();
 }
 }
    vector<vector<int>> combinationSum3(int k, int n) {
 backtracking(n,k,1);
 return result;
    }
};
```

电话号码的字母组合

17.电话号码的字母组合

给定一个仅包含数字 2-9 的字符串，返回所有它能表示的字母组合。

给出数字到字母的映射如下（与电话按键相同）。注意 1 不对应任何字母。

17.电话号码的字母组合

示例:

输入："23"

输出：["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].