第 8 课
递归排列问题

薛浩

xuehao0618@outlook.com

阅读

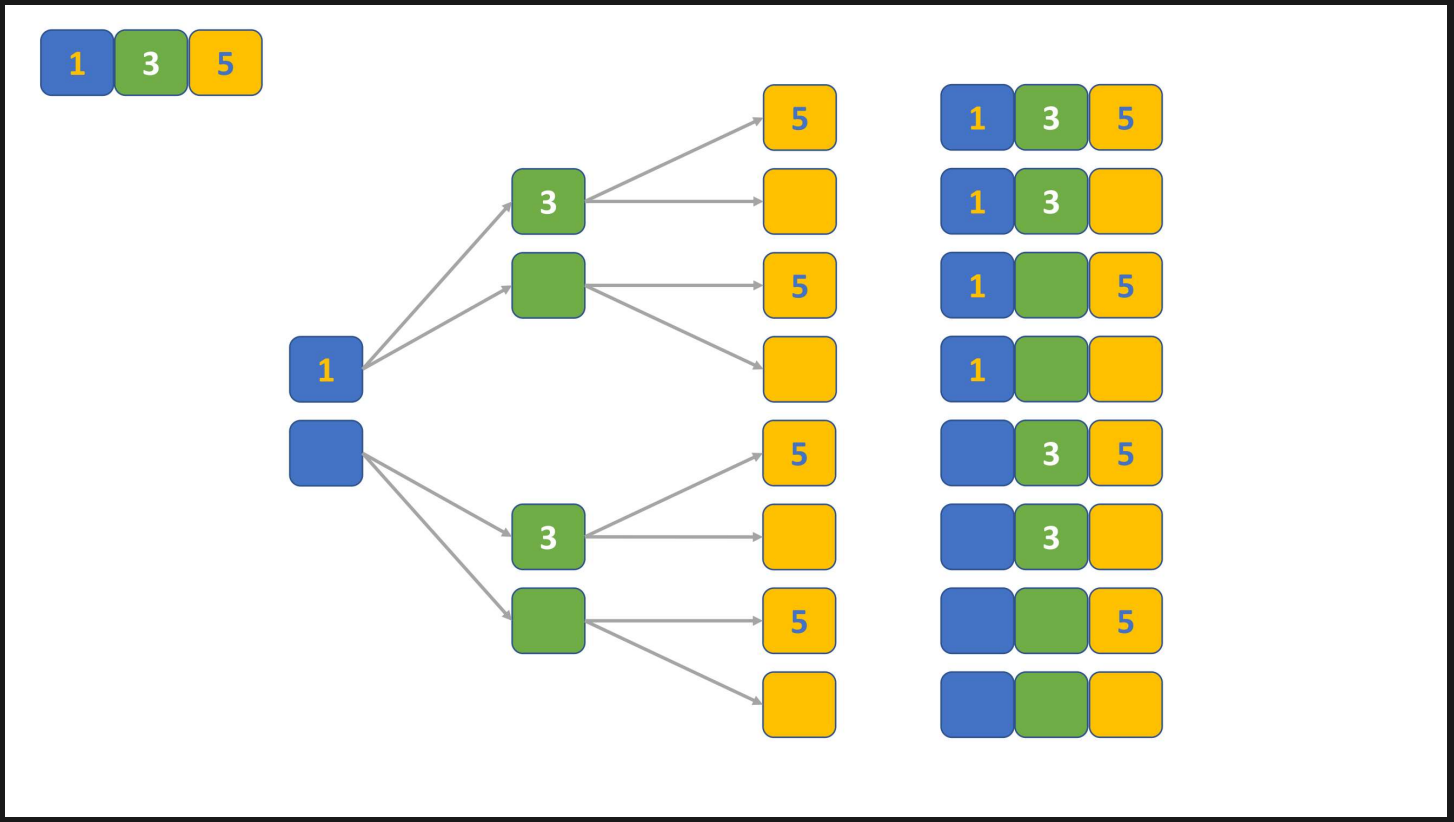
- Programming Abstraction in C++ *Chapter 8.3*

今日话题

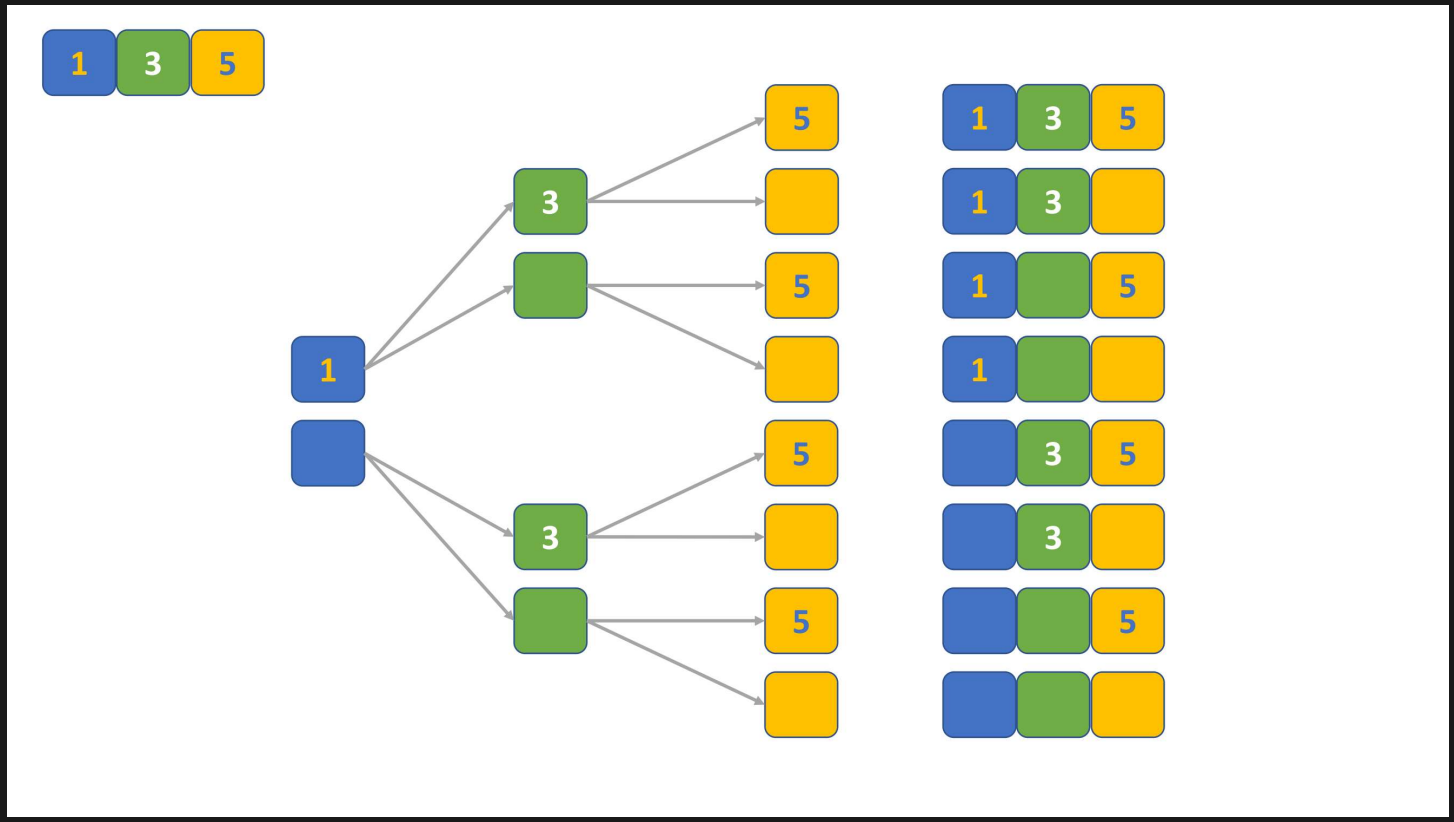
- 回顾
- 递归排列问题

回顧

递归子集问题



递归子集问题



决策树（Decision Tree）

包含/排除模式

```
if( 最简单的情况 ) {
    直接处理问题;
    返回处理结果。
} else {
    选择其中一个元素，降低问题规模;
    包含该元素的处理结果;                    // include
    不包含该元素的处理结果;                  // exclude
    返回最终结果。
}
```

递归函数优化

递归函数优化

- 中间变量

递归函数优化

- 中间变量
  - 包装函数

递归函数优化

- 中间变量
  - 包装函数
- 返回值处理

递归函数优化

- 中间变量
  - 包装函数
- 返回值处理
- 引用 vs 传值

递归函数优化

- 中间变量
  - 包装函数
- 返回值处理
- 引用 vs 传值
  - Side Effect

递归函数优化

- 中间变量
  - 包装函数
- 返回值处理
- 引用 vs 传值
  - Side Effect
- 接口复杂度

今日话题

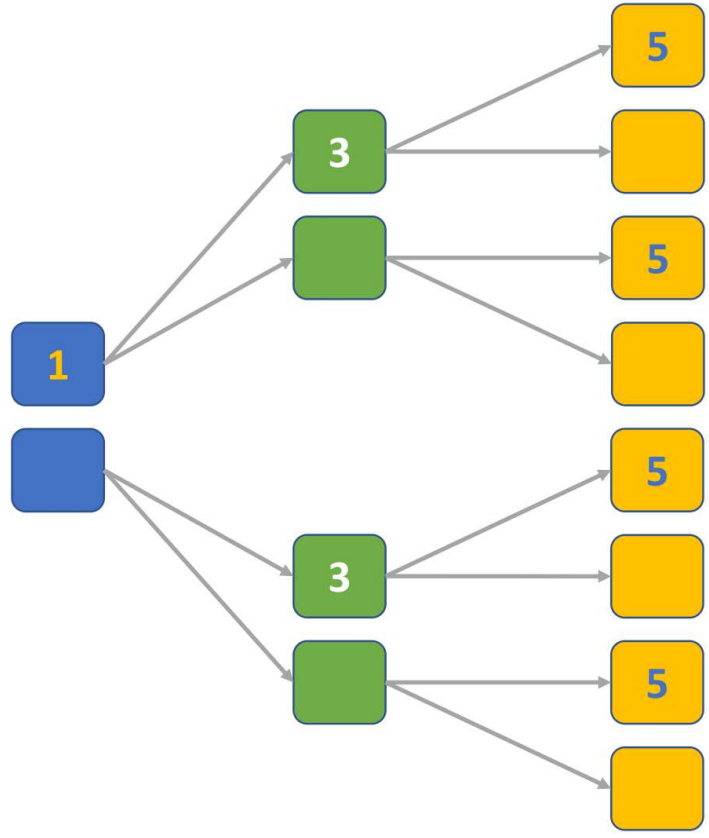
- ~~回顾~~
- 递归排列问题

递归排列问题

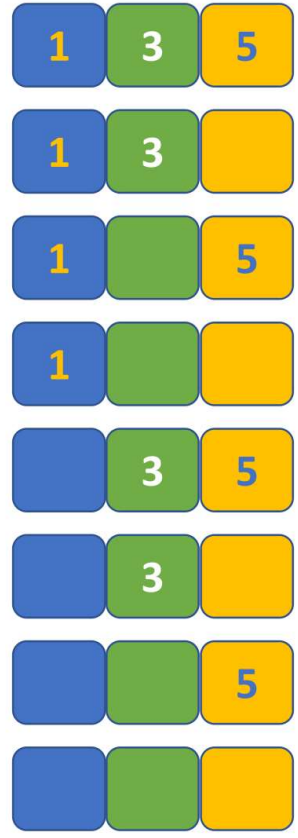
排列是指序列中各元素的重排列。


1
3
5

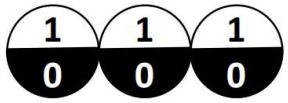

1
3
5

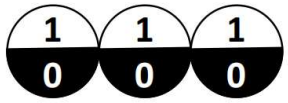

1
3
3
5
5
5
5
5


1
3
5

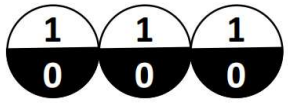

1
3
5
1
3
1
5
1
3
5
3
5


1
3
5


1 1 1
0 0 0

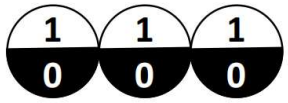

1 1 1
0 0 0

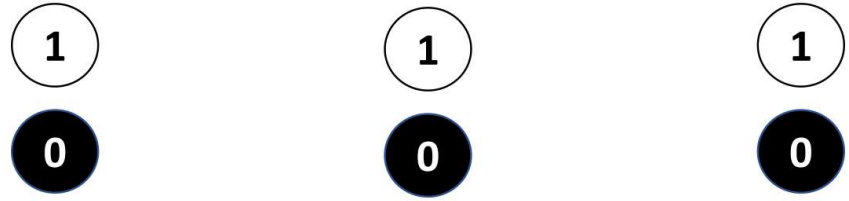

1
0

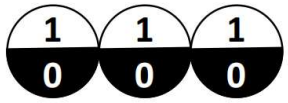

1
1
1
0
0
0

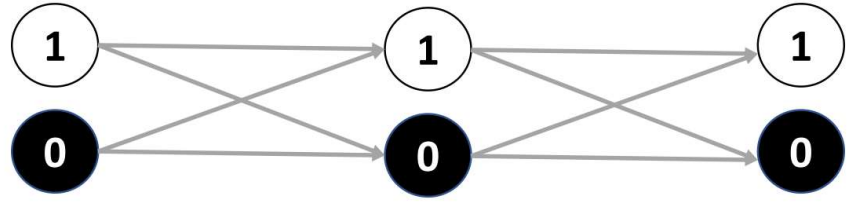

1
0

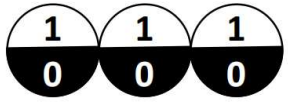

1
0

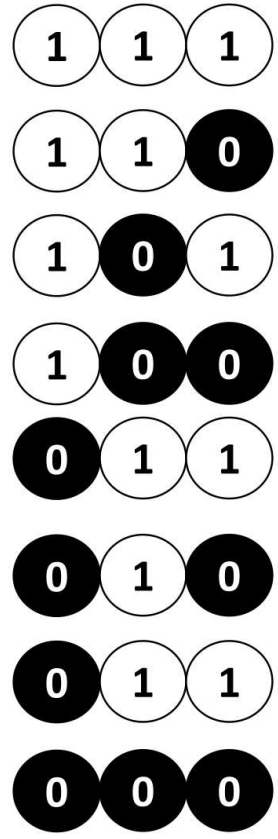

1
1
1
0
0
0


1
1
1
0
0
0


1
1
1
0
0
0


1
1
1
0
0
0


1 1 1
0 0 0


1 1 1
1 1 0
1 0 1
1 0 0
0 1 1
0 1 0
0 1 1
0 0 0

升级版决策树

```cpp
// 递归法求排列: Choose/Explore/Unchoose Pattern
void listPermutationsRec(const Vector<int> &elems,
                         Vector<int> &soFar,
                         int index,
                         Vector<Vector<int>> &result)
    // 1. Base Case: 索引达到结尾
    if (index == elems.size()) {
        result.add(soFar);
    } else {
        for (const int item : elems) {
            // 2.1 Choose
            soFar[index] = item;
            // 2.2 Explore
            listPermutationsRec(elems, soFar, index + 1, re
            // 2.3 Unchoose
```

```cpp
// 递归法求排列: Choose/Explore/Unchoose Pattern
void listPermutationsRec(const Vector<int> &elems,
                         Vector<int> &soFar,
                         int index,
                         Vector<Vector<int>> &result)
    // 1. Base Case: 索引达到结尾
    if (index == elems.size()) {
        result.add(soFar);
    } else {
        for (const int item : elems) {
            // 2.1 Choose
            soFar[index] = item;
            // 2.2 Explore
            listPermutationsRec(elems, soFar, index + 1, re
            // 2.3 Unchoose
```

```
 7      if (index == elems.size()) {
 8          result.add(soFar);
 9      } else {
10          for (const int item : elems) {
11              // 2.1 Choose
12              soFar[index] = item;
13              // 2.2 Explore
14              listPermutationsRec(elems, soFar, index + 1, re
15              // 2.3 Unchoose
16          }
17      }
18  }
19
20  void listPermutationsOf(const Vector<int> &elems) {
```

```
 5                                      Vector<Vector<int>> &result)
 6     // 1. Base Case: 索引达到结尾
 7     if (index == elems.size()) {
 8         result.add(soFar);
 9     } else {
10         for (const int item : elems) {
11             // 2.1 Choose
12             soFar[index] = item;
13             // 2.2 Explore
14             listPermutationsRec(elems, soFar, index + 1, re
15             // 2.3 Unchoose
16         }
17     }
18 }
19
```

```
 7      if (index == elems.size()) {
 8          result.add(soFar);
 9      } else {
10          for (const int item : elems) {
11              // 2.1 Choose
12              soFar[index] = item;
13              // 2.2 Explore
14              listPermutationsRec(elems, soFar, index + 1, re
15              // 2.3 Unchoose
16          }
17      }
18  }
19
20  void listPermutationsOf(const Vector<int> &elems) {
21      Vector<int> soFar = {0, 0, 0};
```

```
 9      } else {
10          for (const int item : elems) {
11              // 2.1 Choose
12              soFar[index] = item;
13              // 2.2 Explore
14              listPermutationsRec(elems, soFar, index + 1, re
15              // 2.3 Unchoose
16          }
17      }
18  }
19
20  void listPermutationsOf(const Vector<int> &elems) {
21      Vector<int> soFar = {0, 0, 0};
22      Vector<Vector<int>> result;
23      listPermutationsRec(elems, soFar, 0, result);
```

迭代  递归

递归和迭代并不是相互排斥的，而且经常组合

CHOOSE/EXPLORE/UNCHOOSE PATTERN

```
ResultType exploreRec(剩余选项, 已选择对象) {
    if (没有多余选项) {
        return 决策结果;
    } else {
        ResultType result; // 用于保存决策结果
        for (下一个可能选项) {
            result += exploreRec(剩余选项, 当前对象 + 可能选项);
        }
        return result; // 返回结果
    }
}
ResultType exploreAllTheThings(初始选项) {
    return exploreRec(初始选项, 无任何已选对象);
}
```

小试牛刀

迷宫问题

今日话题

- ~~回顾~~
- ~~递归排列问题~~

下一次课

- 递归组合问题
- 递归回溯

THE END