Assignment #8: 递归

Updated 1315 GMT+8 Oct 21, 2025

2025 fall, Complied by **韩旭 元培学院**

> **说明：**
>
> 1. **解题与记录：**
>
> 对于每一个题目，请提供其解题思路（可选），并附上使用Python或C++编写的源代码（确保已在OpenJudge， Codeforces，LeetCode等平台上获得Accepted）。请将这些信息连同显示“Accepted”的截图一起填写到下方的作业模板中。（推荐使用Typora https://typoraio.cn 进行编辑，当然你也可以选择Word。）无论题目是否已通过，请标明每个题目大致花费的时间。
>
> 2. 提交安排：**提交时，请首先上传PDF格式的文件，并将.md或.doc格式的文件作为附件上传至右侧的“作业评论”区。确保你的Canvas账户有一个清晰可见的本人头像，提交的文件为PDF格式，并且“作业评论”区包含上传的.md或.doc附件。
> 3. **延迟提交：**如果你预计无法在截止日期前提交作业，请提前告知具体原因。这有助于我们了解情况并可能为你提供适当的延期或其他帮助。
>
> 请按照上述指导认真准备和提交作业，以保证顺利完成课程要求。

1. 题目

M04147汉诺塔问题(Tower of Hanoi)

dfs, http://cs101.openjudge.cn/pctbook/M04147

用时：5min

思路

汉诺塔问题是经典的递归问题：我们可以把移动N个圆盘的大问题拆成移动N-1个圆盘+移动1个圆盘+移动N-1个圆盘的子问题，这样就可以在移动N-1个圆盘的问题上继续进行递归，直到还原为移动1个圆盘的问题。注意这里需要输出完整的移动流程，所以移动M个圆盘的函数需要起始、途径和结束三个杆子的编号。

代码

```python
def move(M,x,y,z): #Move M plates from x to z through y
    if M==1:
        print(f"{M}:{x}->{z}")
    else:
        move(M-1,x,z,y)
        print(f"{M}:{x}->{z}")
        move(M-1,y,x,z)

N,a,b,c=input().split()
N=int(N)
move(N,a,b,c)
```

代码运行截图

M05585: 晶矿的个数

matrices, dfs similar, http://cs101.openjudge.cn/pctbook/M05585

用时：20min

思路

本题每遇到一个晶矿点，都需要递归地查找其上下左右相通的同类型晶矿点，并且在地图上将这一片晶矿标记成`'#'`，避免重复计数。具体来说，递归函数就是将晶矿点本身先设置成`'#'`防止递归无法停止，然后对该位置上下左右的地点进行检查，如果是同类晶矿点，就继续调用函数将其设置为`'#'`并且查找其上下左右的地点点，直到这些地点是不同类的晶矿点或者不是晶矿点，则函数回退。注意需要对地图加保护圈。

代码

```python
def find(Map,i,j,x):
```

```
    if x=="#":
        return
    if Map[i][j]==x:
        Map[i][j]='#'
        find(Map,i-1,j,x)
        find(Map,i+1,j,x)
        find(Map,i,j+1,x)
        find(Map,i,j-1,x)
    return

k=int(input())
for _ in range(k):
    n=int(input())
    Map = []
    r_count=0
    b_count=0
    Map.append(['#']*(n+2))
    for _ in range(n):
        Map.append(['#']+list(input())+['#'])
    Map.append(['#']*(n+2))
    for i in range(n+2):
        for j in range(n+2):
            if Map[i][j]=="r":
                r_count+=1
            elif Map[i][j]=="b":
                b_count+=1
            find(Map,i,j,Map[i][j])
    print(r_count,b_count)
```

代码运行截图

#50673093提交状态 查看 提交 统计 提问

状态: Accepted

基本信息

#: 50673093
题目: M05585
提交人: 22n2200017737
内存: 3672kB
时间: 19ms
语言: Python3
提交时间: 2025-11-02 18:31:08

源代码

```
def find(Map,i,j,x):
    if x=="#":
        return
    if Map[i][j]==x:
        Map[i][j]='#'
        find(Map,i-1,j,x)
        find(Map,i+1,j,x)
        find(Map,i,j+1,x)
        find(Map,i,j-1,x)
    return

k=int(input())
for _ in range(k):
    n=int(input())
    Map = []
    r_count=0
    b_count=0
    Map.append(['#']*(n+2))
    for _ in range(n):
        Map.append(['#']+list(input())+['#'])
    Map.append(['#']*(n+2))
    for i in range(n+2):
        for j in range(n+2):
            if Map[i][j]=="r":
                r_count+=1
            elif Map[i][j]=="b":
                b_count+=1
            find(Map,i,j,Map[i][j])
    print(r_count,b_count)
```


M02786: Pell数列

dfs, dp, http://cs101.openjudge.cn/pctbook/M02786/

用时：25min

思路

这道题逻辑很简单，但是如果用斐波那契数列类似的递归会超时，原因在于递推公式有两项，会造成多次重复运算；此外，当`k`特别大的时候，还可能超过python的递归层数限制。因此，考虑动态规划，用两个变量记录`n-1`和`n-2`两项，然后计算第`n`项并且滚动更新前面两项，这样就不会存在重复运算的问题。但这可能导致最后数字非常大，因此利用取模的线性性质，只需要对各项模32767的余数按递推公式迭代即可。

代码

```
N=int(input())
for _ in range(N):
    a=1
    b=2
```

```
    ans=0
    k=int(input())
    for i in range(k):
        if i==0:
            ans=a
        elif i==1:
            ans=b
        else:
            ans=(a+2*b)%32767
            a,b=b,ans
    print(ans)
```

代码运行截图

#50673715提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
N=int(input())
for _ in range(N):
    a=1
    b=2
    ans=0
    k=int(input())
    for i in range(k):
        if i==0:
            ans=a
        elif i==1:
            ans=b
        else:
            ans=(a+2*b)%32767
            a,b=b,ans
    print(ans)
```

基本信息

#: 50673715
题目: M02786
提交人: 22n2200017737
内存: 3572kB
时间: 516ms
语言: Python3
提交时间: 2025-11-02 19:13:10

English 帮助 关于

M46.全排列

backtracking, https://leetcode.cn/problems/permutations/

用时：25min

思路

这道题我最开始的思路是第一种，即先用循环分别选定第一个数字，然后剩下的数字递归地进行全排列。Leetcode题解给出了一个更快的解法，使用的是**回溯**的策略，这第二个思路本质上和第一个一样，也是用循环分别选定第一个数字，然后递归地处理剩下的数字。比较巧妙的是，它在这里对选定的数字和目前第一个数字进行了交换，然后直接对从第二个数字开始的数组递归地执行回溯函数，直到第n个数字，然后再慢慢复位，尝试新的排列。这种思路比较像DFS，它不需要复制数组，因此时间复杂度更低。此外，如果新建一个临时数组存储答案，每次都循环完整的`nums`数组，但是在循环中加一个判断是否已经在临时数组中，也是可以的。

在细节上，本题要特别注意深拷贝的问题。不论是第一种思路在复制的时候浅拷贝，还是第二种思路在append进答案数组的时候浅拷贝，都会改变原数组或者使答案随原数组改变。所以，如果列表元素不是列表，使用`nums[:]`可以达到目的，如果列表元素也包括列表，那么使用`copy.deepcopy(nums)`是最保险的。

代码

思路一：

```
import copy
class Solution(object):
    def permute(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        ans=[]
        if len(nums)==1:
            ans.append(nums[:])
            return ans
        for i in range(len(nums)):
            tmp=copy.deepcopy(nums)
            del tmp[i]
            sub_ans=self.permute(tmp)
            for j in sub_ans:
                ans.append([nums[i]]+j)
        return ans
```

思路二：

```
class Solution(object):
    def permute(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        def backtracking(first=0):
            if first==n:
                ans.append(nums[:])
            for i in range(first,n):
                nums[first],nums[i]=nums[i],nums[first]
                backtracking(first+1)
                nums[first],nums[i]=nums[i],nums[first]
        n=len(nums)
        ans=[]
        backtracking()
        return ans
```

代码运行截图

通过 26 / 26 个通过的测试用例

韩旭 提交于 2025.11.02 22:18

官方题解 写题解

执行用时分布

11 ms | 击败 11.77%

复杂度分析

消耗内存分布

12.32 MB | 击败 88.95%

代码 | Python

```python
import copy
class Solution(object):
    def permute(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        ans=[]
        if len(nums)==1:
            ans.append(nums[:])
            return ans
        for i in range(len(nums)):
            tmp=copy.deepcopy(nums)
            del tmp[i]
            sub_ans=self.permute(tmp)
            for j in sub_ans:
                ans.append([nums[i]]+j)
        return ans
```

收起

通过 26 / 26 个通过的测试用例

韩旭 提交于 2025.11.02 22:02

官方题解 写题解

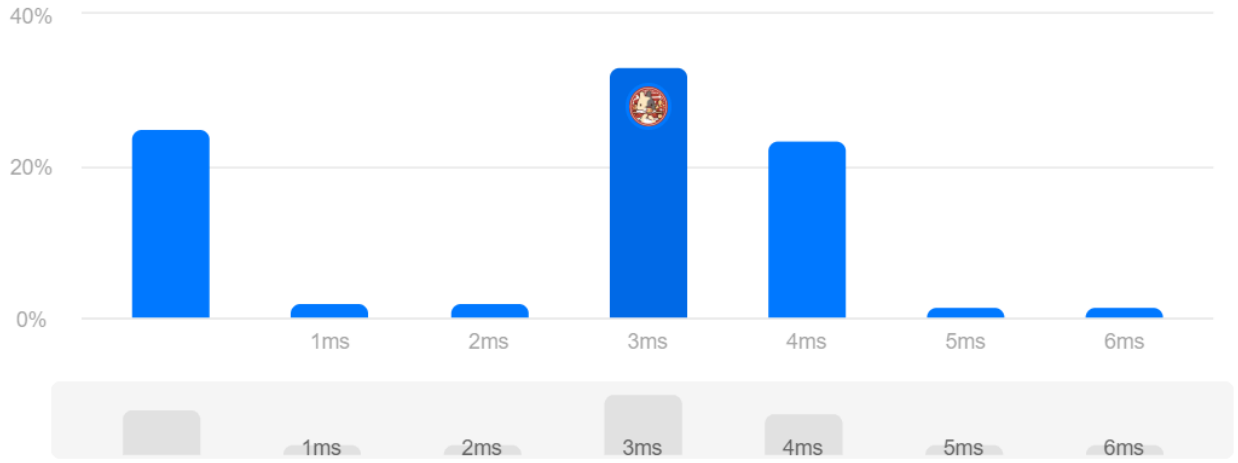
执行用时分布

3 ms | 击败 70.92%

复杂度分析

消耗内存分布

12.52 MB | 击败 35.37%



代码 | Python

```python
class Solution(object):
    def permute(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        def backtracking(first=0):
            if first==n:
                ans.append(nums[:])
            for i in range(first,n):
                nums[first],nums[i]=nums[i],nums[first]
                backtracking(first+1)
                nums[first],nums[i]=nums[i],nums[first]
        n=len(nums)
        ans=[]
        backtracking()
        return ans
```

收起

T02754: 八皇后

dfs and similar, http://cs101.openjudge.cn/pctbook/T02754

用时：15min

思路

这题仍然使用回溯，先在第一列试探性地放一个皇后，然后递归进入第二列，循环各行来判断第一个不和前面的皇后相冲突的位置，然后继续试探性地放在这里，以此类推直到第八列，记录对应的皇后串。然后进行回溯，把上一个试探性放入的皇后撤回，再尝试下一个可能得皇后位置即可。这样按顺序求得的皇后串恰好就是排序好的，所以在打表之后输出第`k`的皇后串即可。

代码

```
rows=[]
diag1=[]
diag2=[]
sol=[]
ans=[]
def queens(column=1):
    for i in range(1,9):
        if i not in rows and i+column not in diag1 and i-column not in diag2:
            sol.append(i)
            rows.append(i)
            diag1.append(i+column)
            diag2.append(i-column)
            if column==8:
                ans.append(sol[:])
            else:
                queens(column+1)
            sol.pop()
            rows.pop()
            diag1.pop()
            diag2.pop()
queens()
n=int(input())
for _ in range(n):
    k=int(input())
    print("".join(map(str,ans[k-1])))
```

代码运行截图

#50678291提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
rows=[]
diag1=[]
diag2=[]
sol=[]
ans=[]
def queens(column=1):
    for i in range(1,9):
        if i not in rows and i+column not in diag1 and i-column not in d
            sol.append(i)
            rows.append(i)
            diag1.append(i+column)
            diag2.append(i-column)
            if column==8:
                ans.append(sol[:])
            else:
                queens(column+1)
            sol.pop()
            rows.pop()
            diag1.pop()
            diag2.pop()
queens()
n=int(input())
for _ in range(n):
    k=int(input())
    print("".join(map(str,ans[k-1])))
```

基本信息

#: 50678291
题目: T02754
提交人: 22n2200017737
内存: 3656kB
时间: 23ms
语言: Python3
提交时间: 2025-11-02 22:53:07

English 帮助 关于

T01958 Strange Towers of Hanoi

http://cs101.openjudge.cn/practice/01958/

用时：20min

思路

按照题目所描述的解法，我们需要四个柱子移动的函数和三个柱子移动的函数。其中，三个柱子移动的函数就按照经典汉诺塔的写法来写即可。而四个柱子移动的函数，则需要遍历$k \in \{1, 2, \ldots, n\}$，分别先在四个柱子时移动n-k个，然后在三个柱子时移动k个，最后在四个柱子时移动n-k个。然后比较各个k所需要的移动次数，选出最小值即可。

代码

```
def move_three(n):
    if n==1:
        return 1
    ans=0
    ans+=move_three(n-1)
    ans+=1
    ans+=move_three(n-1)
    return ans
```

```python
def move_four(n):
    if n==0:
        return 0
    if n==1:
        return 1
    ans=10000
    for k in range(1,n+1):
        tmp=0
        tmp+=move_four(n-k)
        tmp+=move_three(k)
        tmp+=move_four(n-k)
        ans=min(tmp,ans)
    return ans

for n in range(1,13):
    print(move_four(n))
```

代码运行截图

#50678583提交状态

查看 提交 统计 提问

状态: Accepted

基本信息

#: 50678583
题目: 01958
提交人: 22n2200017737
内存: 3520kB
时间: 221ms
语言: Python3
提交时间: 2025-11-02 23:22:19

源代码

```python
def move_three(n):
    if n==1:
        return 1
    ans=0
    ans+=move_three(n-1)
    ans+=1
    ans+=move_three(n-1)
    return ans

def move_four(n):
    if n==0:
        return 0
    if n==1:
        return 1
    ans=10000
    for k in range(1,n+1):
        tmp=0
        tmp+=move_four(n-k)
        tmp+=move_three(k)
        tmp+=move_four(n-k)
        ans=min(tmp,ans)
    return ans

for n in range(1,13):
    print(move_four(n))
```

2. 学习总结和收获

本次作业主题是递归与回溯。递归问题主要需要分析两个部分，一个是如何正确拆解子问题（特别是递归函数需要哪些参数，有时候最大的递归不需要的参数，后面处理子问题可能需要，就需要对某些参数设置default值），另一个是如何回溯到原来的状态重新尝试。

汉诺塔和晶矿计数这两个题是比较简单的递归，每次只需要递归到底就可以完成任务。Pell 数列的问题说明经典递归的局限性，即当子问题之间存在重叠时，单纯的递归会造成大量重复计算甚至超出栈深，这就需要使用动态规划的思想，但本质和递归是类似的。通过将递归改写为迭代、使用滚动变量优化空间，事实上实现了“记忆化”与“自底向上”的过程。

全排列和八皇后问题引入了回溯，我在全排列这题学习了回溯的思路后，很快就能运用到八皇后问题里面。这里的核心思路是“递归搜索”，或者DFS，也就是通过递归和回溯来实现**试错**的过程。通过调试这两道题的细节，我也体会到**原地修改 + 状态恢复**相比复制数组的高效性，以及在修改和恢复过程中深拷贝的重要性。最后的奇怪汉诺塔问题是一种“递归嵌套递归”的结构：在外层递归中又调用另一种递归（move_three 与 move_four 的相互作用）。