Assignment #A: 递归、田忌赛马

Updated 2355 GMT+8 Nov 4, 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

M018160: 最大连通域面积

dfs similar, http://cs101.openjudge.cn/pctbook/M18160

用时：10min

思路

这道题和之前作业里面“晶矿的个数”思路基本一致，所以完成得很快。基本思路还是写一个递归的`find()`函数，从某一个坐标(i,j)出发，如果(i,j)上有棋子，就继续搜索其周围八个坐标，然后按照dfs的思路不断扩展，直到遇到没有棋子的情况就返回0停止搜索。要注意在搜索前要先把已经搜索过的位置更新成“.”，避免递归无法停止。

代码

```
Dx=[-1,0,1]
Dy=[-1,0,1]
def find(i,j,b):
    if b[i][j]==".":
        return 0
    else:
```

```
        ans=1
        b[i][j]="."
        for dx in Dx:
            for dy in Dy:
                ans+=find(i+dx,j+dy,b)
        return ans

K=int(input())
for _ in range(K):
    N,M=map(int,input().split())
    board=[]
    board.append(['.']*(M+2))
    for _ in range(N):
        board.append(['.']+list(input())+['.'])
    board.append(['.'] * (M+2))
    ans=0
    for i in range(1,N+1):
        for j in range(1,M+1):
            ans=max(find(i,j,board),ans)
    print(ans)
```

代码运行截图

#50879555提交状态

查看 提交 统计 提问

状态：Accepted

源代码

```
Dx=[-1,0,1]
Dy=[-1,0,1]
def find(i,j,b):
    if b[i][j]==".":
        return 0
    else:
        ans=1
        b[i][j]="."
        for dx in Dx:
            for dy in Dy:
                ans+=find(i+dx,j+dy,b)
        return ans

K=int(input())
for _ in range(K):
    N,M=map(int,input().split())
    board=[]
    board.append(['.']*(M+2))
    for _ in range(N):
        board.append(['.']+list(input())+['.'])
    board.append(['.'] * (M+2))
    ans=0
    for i in range(1,N+1):
        for j in range(1,M+1):
            ans=max(find(i,j,board),ans)
    print(ans)
```

基本信息

#:	50879555
题目:	M18160
提交人:	22n2200017737
内存:	3684kB
时间:	105ms
语言:	Python3
提交时间:	2025-11-17 16:49:15

sy134: 全排列III 中等

https://sunnywhy.com/sfbj/4/3/134

用时：15min

思路

这道题由于要按顺序输出所有排列，所以选择使用生成器来完成。基本的递归思路和正常全排列一致，先挑选数组里的某一个元素，然后对数组中剩余选择进行全排列，再把这个挑出来的元素加在每个排列前面。由于输入是升序排列的，直接从左到右遍历就可以实现字典序的输出。这道题的区别在于数字存在重复，对于重复的数字，我们挑出来任意一个即可，不需要在遍历的时候重复挑出，所以考虑加入一个`record`数组进行记录，在递归之前判断同一个数字之前有没有挑出来过即可。

代码

```
N=int(input())
nums=list(map(int,input().split()))

def arrange(nums):
    if len(nums)==1:
        yield nums
    else:
        record=[]
        for i in range(len(nums)):
            if nums[i] in record:
                continue
            else:
                tmp=nums[:]
                del tmp[i]
                for y in arrange(tmp):
                    yield [nums[i]]+y
                record.append(nums[i])

for ans in arrange(nums):
    print(" ".join(map(str,ans)))
```

代码运行截图

sy136: 组合II 中等

https://sunnywhy.com/sfbj/4/3/136

给定一个长度为的序列，其中有n个互不相同的正整数，再给定一个正整数k，求从序列中任选k个的所有可能结果。

用时：10min

思路

这道题沿用上一题的生成器解法似也可以，只需要给函数加一个目前子序列长度的参数即可。这里为了练习，采用了回溯的方法。具体来说，每次在一个临时数组`sol`上面进行dfs，直到长度达到目标K之后，就把`sol`数组拷贝到`Ans`数组当中，注意这里要使用深拷贝。由于组合没有顺序的区别，我们可以给`backtracking()`函数加入一个参数`start`，表示此后的递归从`start`之后（更大的）数字中挑选即可。

代码

```
N,K=map(int,input().split())
nums=list(map(int,input().split()))

Ans, sol=[],[]

def backtracking(start=0):
```

```
    if len(sol)==K:
        Ans.append(sol[:])
        return
    else:
        for i in range(start,N):
            if nums[i] not in sol:
                sol.append(nums[i])
                backtracking(start=i+1)
                sol.pop()
backtracking()
for ans in Ans:
    print(" ".join(map(str,ans)))
```

代码运行截图

sy137: 组合III 中等

https://sunnywhy.com/sfbj/4/3/137

用时：5min

思路

这道题从上一题的代码修改而来，只需要加入第二题处理可重复输入的方法即可。具体来说，对于重复的数字，一开始只需要挑出其中任意一个，然后对剩下的序列进行dfs即可，所以我们仍然维护一个record数组，在遍历过程中，对于已经遍历过的数字，如果再遇到重复的数字就跳过去即可。但要注意，这个record数组必须只在每一层递归内部有效，而不能是一个全局数组，因为我们只是在某一层递归的时候不挑出重复的数字，以避免选出来的子序列重复。但是进入下一层递归之后，即使是和之前重复的数字也是需要挑出来作为一个组合的。

代码

```
N,K=map(int,input().split())
nums=list(map(int,input().split()))

Ans, sol=[],[]
def backtracking(start=0):
    if len(sol)==K:
        Ans.append(sol[:])
        return
    else:
        record=[] #NEW
        for i in range(start,N):
            if nums[i] not in record:
                sol.append(nums[i])
                record.append(nums[i]) #NEW
                backtracking(start=i+1)
                sol.pop()
backtracking()
for ans in Ans:
    print(" ".join(map(str,ans)))
```

代码运行截图

M04123: 马走日

dfs, http://cs101.openjudge.cn/pctbook/M04123

用时：45min

思路

这道题的基本思路并不困难，就是按照dfs，每次遍历八种可能的走法，一层层搜索，直到走过的格点数等于$n \times m$。在这个过程中，要检查约束条件，在一个`board`矩阵里面检查每次走到的点是不是之前走过，不然递归将会爆栈。这里面要特别注意回溯，在**每一处return之前**都要检查是否完成了回溯。我一开始在检查走过的格点数等于$n \times m$的分支没有进行回溯，导致答案出现错误。

AI给出了一个孤立点的剪枝，也就是，进入遍历之前，先查看一下棋盘上有没有“孤立点”，也就是某个点周围的八个点都被走过了，并且下一步也没有办法走到它。如果存在这样的点，就可以直接`return 0`，不需要继续往下搜索了。这个剪枝可以降低一定的时间复杂度，但是写的时候需要逻辑比较严谨（在代码里有所呈现，截图中仍然是不含剪枝的版本）。

代码

```
K=int(input())
for _ in range(K):
    n,m,x,y=map(int,input().split())
```

```python
board=[[False]*m for _ in range(n)]
D=[(-1,2),(-1,-2),(-2,1),(-2,-1),(1,2),(1,-2),(2,1),(2,-1)]
count=0

def has_isolated(x,y):
    for i in range(n):
        for j in range(m):
            if not board[i][j]:
                ok = False
                for dx, dy in D:
                    ni, nj = i + dx, j + dy
                    if 0 <= ni < n and 0 <= nj < m and not board[ni][nj]:
                        ok = True
                        break
                if not ok:
                    can_be_last = False
                    for dx, dy in D:
                        if i == x + dx and j == y + dy:
                            can_be_last = True
                            break
                    if not can_be_last:
                        return True
    return False

def dfs(x,y):
    global count #注意设置全局变量
    if board[x][y]:
        return 0

    board[x][y]=True
    count+=1

    if count==n*m:
        board[x][y]=False #这里的回溯不要忘记
        count-=1# 这里的回溯不要忘记
        return 1

    if has_isolated(x,y):
        board[x][y]=False
        count-=1
        return 0

    ans=0
    for l in D:
        if 0<=x+l[0]<n and 0<=y+l[1]<m:
            ans+=dfs(x+l[0],y+l[1])
    board[x][y]=False
    count-=1
    return ans

print(dfs(x,y))
```

代码运行截图

#50883095提交状态

查看 提交 统计 提问

状态：Accepted

源代码

```
K=int(input())
for _ in range(K):
    n,m,x,y=map(int,input().split())
    board=[[False]*m for _ in range(n)]
    D=[(-1,2),(-1,-2),(-2,1),(-2,-1),(1,2),(1,-2),(2,1),(2,-1)]
    count=0

    def dfs(x,y):
        global count #注意设置全局变量
        if board[x][y]:
            return 0

        board[x][y]=True
        count+=1

        if count==n*m:
            board[x][y]=False #这里的回溯不要忘记
            count-=1# 这里的回溯不要忘记
            return 1

        ans=0
        for l in D:
            if 0<=x+l[0]<n and 0<=y+l[1]<m:
                ans+=dfs(x+l[0],y+l[1])
        board[x][y]=False
        count-=1
        return ans

    print(dfs(x,y))
```

基本信息

#: 50883095
题目: M04123
提交人: 22n2200017737
内存: 3560kB
时间: 3959ms
语言: Python3
提交时间: 2025-11-17 20:09:50

English 帮助 关于

T02287: Tian Ji -- The Horse Racing

greedy, dfs http://cs101.openjudge.cn/pctbook/T02287

用时：1h

思路

这个题确实不能被题干说的图算法干扰，一开始想用dfs搜索所有可能得matching方式然后找出收益最大的方法（加一些剪枝），但是时间复杂度太高（$O(n!)$）。

正确的思路就是按田忌本来的方法进行贪心。田忌本来是发现，他的每个等级的马都打不过齐王的每个等级的马，所以宁可直接用他最弱的马打齐王最强的马，这样性价比最高。然后田忌就发现，去掉这两个比完赛的马，再排序对齐之后，他的每个等级的马就可以打得过齐王每个等级的马了。

由于用最弱打最强的方案影响的只是一头一尾两个马（剩下的马后面还可以比），所以一开始是否要用这种最弱打最强的方法时，我们只需要判断自己的最弱和最强的马有没有可能直接打赢齐王最弱和最强的马。如果可以，那么消耗最弱或者最强的马就可以直接获得收益（各）200，所以我们就使用贪心，优先让这两匹马出战。如果不行，就采取最弱打最强的方法。这里要特别注意平局的情况，平局的时候，消耗最弱或者最强的马无法获得收益；但这个时候就

可以推出，田忌的最强一定能打得过齐王的第二强，田忌的第二弱一定打得过齐王的最弱，所以拿最弱打最强之后虽然损失200，但是后面一定可以收回来400，所以平局的情况也要使用最弱打最强的的方法。

具体来说，由于要分别对一头一尾进行贪心，后面还要重新对齐，我们对齐王和田忌各自维护双指针，然后先比最强的两个马，然后比最弱的两个马，如果田忌有能打赢的，就出战并且将指针往里推一步；如果两个都打不过或者平局，就用最弱打最强，然后把齐王的左指针往右推，田忌的右指针往左推，重新进行对齐。以此类推，直到所有马都出战过了。

代码

```
while True:
    n=int(input())
    if n==0:
        break
    tian=list(map(int,input().split()))
    king=list(map(int,input().split()))
    tian.sort(reverse=True)
    king.sort(reverse=True)
    tL,tR=0,n-1
    kL,kR=0,n-1
    ans=0
    while tL<=tR:
        if tian[tL]>king[kL]:
            ans+=200
            tL+=1
            kL+=1
        elif tian[tR]>king[kR]:
            ans+=200
            tR-=1
            kR-=1
        else:
            if tian[tR]<king[kL]:
                ans-=200
            tR-=1
            kL+=1
    print(ans)
```

代码运行截图

#50891409提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
while True:
    n=int(input())
    if n==0:
        break
    tian=list(map(int,input().split()))
    king=list(map(int,input().split()))
    tian.sort(reverse=True)
    king.sort(reverse=True)
    tL,tR=0,n-1
    kL,kR=0,n-1
    ans=0
    while tL<=tR:
        if tian[tL]>king[kL]:
            ans+=200
            tL+=1
            kL+=1
        elif tian[tR]>king[kR]:
            ans+=200
            tR-=1
            kR-=1
        else:
            if tian[tR]<king[kL]:
                ans-=200
            tR-=1
            kL+=1
    print(ans)
```

基本信息

#: 50891409
题目: T02287
提交人: 22n2200017737
内存: 3816kB
时间: 53ms
语言: Python3
提交时间: 2025-11-18 15:53:23

English 帮助 关于

2. 学习总结和收获

本次作业主要是 **递归（包括DFS、回溯、生成器、剪枝等具体策略）** 。通过地图搜索和排列组合的题目和各种变式，我更熟悉了递归的各种写法，包括状态恢复的重要性，递归函数参数的传递，以及如何用 `record` 数组处理重复元素以避免产生重复解。

在“最大连通域面积”和“马走日”两题中，我进一步巩固了 DFS 在矩阵地图中的应用。尤其是“马走日”里面，我的思路是在递归函数一开始就标记这一步，所以后面如果有分支语句，每一个分支就都要回溯。同时，这个题的优化（剪枝，判断孤立点）也很值得思考，优化的过程不能引入更复杂的算法（比如多次的遍历），也不能只是调整了搜索的顺序而没有影响到复杂度的数量级——一个比较常见的思路是，提前通过某些特征/关系式来判断不需要继续搜索，这个路径不可能成功。

在排列组合的各种变式中，我同时采用了课上讲的 **回溯与生成器** 的方法。回溯适合全局收集某个统计量或者满足条件的某些结果（挑选组合），生成器则能更优雅地流式输出所有结果（全排列）。

“田忌赛马”主要是一个贪心题。一开始尝试用 DFS 枚举所有匹配方式，但复杂度是 $O(n!)$，贪心其实是对DFS的一个降维的剪枝。这个题有非常好的排序性质，所以很适合使用贪心。我们需要从大家都知道的田忌的3种马策略，尝试推广到n匹马的情况，抓住其中关键的判断条件（什么情况下要用最弱打最强），这里面每一步都需要判断到底什么策略是“最贪心的”。另外，平局的情况确实比较难考虑，需要利用好排序特征，往后多想一步，考虑这一轮到底是打一个平局还是采用最弱打最强先输一轮。