1. 题目

18161: 矩阵运算

matrices, http://cs101.openjudge.cn/practice/18161

思路：分为乘法（矩阵 ABD）和加法（矩阵 CDE）两部分依次进行，先对输入进行检查，输入不对输出 Error；接着先进行乘法，同样先进行检查，之后初始化一个矩阵 D，遍历矩阵 A 的每一行和矩阵 B 的每一列，再遍历矩阵 A 的每一列，计算 A 的每行与 B 的每列对应元素的乘积之和，得到矩阵 D 中的每个元素。接着进行加法，先检查，然后初始化矩阵 E，遍历矩阵 C 和之前乘法中得到的矩阵 D 的每个元素，把对应位置的元素相加，得到矩阵 E 的元素。

代码：

```
A1,A2=map(int,input().split())
A=[]
for _ in range(A1):
    jzA=list(map(int, input().split()))
    A.append(jzA)
if len(A) != A1 or len(A[0]) != A2: # 检查矩阵的维度是否符合输入的行列数
    print("Error!")
    exit()

B1,B2=map(int,input().split())
B=[]
for _ in range(B1):
    jzB=list(map(int, input().split()))
    B.append(jzB)
if len(B) != B1 or len(B[0]) != B2: # 检查矩阵的维度是否符合输入的行列数
```

```
    print("Error!")
    exit()


C1,C2=map(int,input().split())
C=[]
for _ in range(C1):
    jzC=list(map(int, input().split()))
    C.append(jzC)
if len(C) != C1 or len(C[0]) != C2: # 检查矩阵的维度是否符合输入的行列数
    print("Error!")
    exit()
# print(A,B,C)
# ====================乘法======================
rows_A = len(A)
cols_A = len(A[0])
rows_B = len(B)
cols_B = len(B[0])
if cols_A != rows_B: # 矩阵 A 的列数必须等于矩阵 B 的行数才能进行矩阵乘法
    print("Error!")
    exit()
D=[[0]*cols_B for _ in range(rows_A)] # 用矩阵 A 的行数*矩阵 B 的列数创建一个全为 0 的矩阵（初始化）
for i in range(rows_A): # 遍历矩阵 A 的每一行
    for j in range(cols_B): # 遍历矩阵 B 的每一列
        for k in range(cols_A): # 遍历矩阵 A 的每一列
            D[i][j] += A[i][k] * B[k][j] # 计算 A 的每行与 B 的每列对应元素的乘积之和，得到矩阵 D 中的每个元素
```

```
# print(D)
# ==========加法==========
rows_C = len(C)
cols_C = len(C[0])
rows_D = len(D)
cols_D = len(D[0])

if rows_C != rows_D or cols_C != cols_D: # 矩阵 D 和矩阵 C 的行列数必须相等才能进行矩阵加法
    print("Error!")
    exit()
E= [[0] * cols_C for _ in range(rows_C)] # 同样初始化矩阵 E
for i in range(rows_C):
    for j in range(cols_C):
        E[i][j] = C[i][j] + D[i][j] # 遍历矩阵 C 和 D 的每个元素，把对应位置的元素相加，得到矩阵 E 的元素
for i in E:
    print(" ".join(map(str,i)))
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

#48406234提交状态

查看 提交 统计 提问

状态: Accepted

基本信息

#: 48406234
题目: 18161
提交人: 2400093012 苏倩仪
内存: 4456kB
时间: 125ms
语言: Python3
提交时间: 2025-03-01 21:04:58

源代码

```
A1,A2=map(int,input().split())
A=[]
for _ in range(A1):
    jzA=list(map(int, input().split()))
    A.append(jzA)
if len(A) != A1 or len(A[0]) != A2: # 检查矩阵的维度是否符合输入的行列数
    print("Error!")
    exit()

B1,B2=map(int,input().split())
B=[]
for _ in range(B1):
    jzB=list(map(int, input().split()))
    B.append(jzB)
if len(B) != B1 or len(B[0]) != B2: # 检查矩阵的维度是否符合输入的行列数
    print("Error!")
    exit()

C1,C2=map(int,input().split())
C=[]
for _ in range(C1):
    jzC=list(map(int, input().split()))
    C.append(jzC)
if len(C) != C1 or len(C[0]) != C2: # 检查矩阵的维度是否符合输入的行列数
    print("Error!")
    exit()
# print(A,B,C)
# ====================乘法======================
rows_A = len(A)
cols_A = len(A[0])
rows_B = len(B)
cols_B = len(B[0])
if cols_A != rows_B: # 矩阵A的列数必须等于矩阵B的行数才能进行矩阵乘法
    print("Error!")
    exit()
D=[[0]*cols_B for _ in range(rows_A)] # 用矩阵A的行数*矩阵B的列数创建一个全为
for i in range(rows_A): # 遍历矩阵A的每一行
    for j in range(cols_B): # 遍历矩阵B的每一列
        for k in range(cols_A): # 遍历矩阵A的每一列
            D[i][j] += A[i][k] * B[k][j] # 计算A的每行与B的每列对应元素的乘积

# print(D)
# ==========================加法=========================
rows_C = len(C)
cols_C = len(C[0])
rows_D = len(D)
cols_D = len(D[0])

if rows_C != rows_D or cols_C != cols_D: # 矩阵D和矩阵C的行列数必须相等才能进
    print("Error!")
    exit()
E= [[0] * cols_C for _ in range(rows_C)] # 同样初始化矩阵E
for i in range(rows_C):
    for j in range(cols_C):
        E[i][j] = C[i][j] + D[i][j] # 遍历矩阵C和D的每个元素，把对应位置的元素

for i in E:
    print(" ".join(map(str,i)))
```

大约用时：1 小时

19942: 二维矩阵上的卷积运算

matrices, http://cs101.openjudge.cn/practice/19942/

思路：定义列表 mat 为一个 m*n 的原矩阵和列表 ker 为一个 p*q 的卷积核，接着计算滑动窗口的大小，再遍历 mat 列表，在 mat 中取出 p*q 的子矩阵并将每个元素与 ker 对应位置的元素相乘并累加，最后存入 res 列表并打印。

代码：

```
m,n,p,q=list(map(int,input().split()))
mat=[] # 原矩阵
for _ in range(m):
    b=list(map(int,input().split()))
    mat.append(b)
ker=[] # 卷积核
for _ in range(p):
    s=list(map(int,input().split()))
    ker.append(s)

cols=n+1-q # 能移动的列
rows=m+1-p # 能移动的行

res=[] # 初始化
for i in range(rows):
    row = [] #当前行
    for j in range(cols):
        num = 0 # 记录每一次计算
        for a in range(p):
            for b in range(q):
                num+=mat[a+i][b+j] * ker[a][b] # 矩阵与卷积核对应位置元素相乘并累加
```

```
        row.append(num) # 保存结果
    res.append(row) # 保存计算后的整行结果

for i in res:
    print(" ".join(map(str,i)))
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

大约用时：1 小时

04140: 方程求解

牛顿迭代法，http://cs101.openjudge.cn/practice/04140/

请用<mark>牛顿迭代法</mark>实现。

因为大语言模型的训练过程中涉及到了梯度下降（或其变种，如 SGD、Adam 等），用于优化模型参数以最小化损失函数。两种方法都是通过迭代的方式逐步接近最优解。每一次迭代都基于当前点的局部信息调整参数，试图找到一个比当前点更优的新点。理解牛顿迭代法有助于深入理解基于梯度的优化算法的工作原理，特别是它们如何利用导数信息进行决策。

> **牛顿迭代法**

>

> - **目的**：主要用于寻找一个函数 $f(x)$ 的根，即找到满足 $f(x)=0$ 的 x 值。不过，通过适当变换目标函数，它也可以用于寻找函数的极值。

> - **方法基础**：利用泰勒级数的一阶和二阶项来近似目标函数，在每次迭代中使用目标函数及其导数的信息来计算下一步的方向和步长。

> - **迭代公式**：$ x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} $ 对于求极值问题，这可以转化为$ x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)} $，这里 $f'(x)$ 和 $f''(x)$ 分别是目标函数的一阶导数和二阶导数。

> - **特点**：牛顿法通常具有更快的收敛速度（尤其是对于二次可微函数），但是需要计算目标函数的二阶导数（Hessian 矩阵在多维情况下），并且对初始点的选择较为敏感。

>

> **梯度下降法**

>

> - **目的**：直接用于寻找函数的最小值（也可以通过取负寻找最大值），尤其在机器学习领域应用广泛。

> - **方法基础**：仅依赖于目标函数的一阶导数信息（即梯度），沿着梯度的反方向移动以达到减少函数值的目的。

> - **迭代公式**：$ x_{n+1} = x_n - \alpha \cdot \nabla f(x_n) $ 这里 α 是学习率，$\nabla f(x_n)$ 表示目标函数在 x_n 点的梯度。

> - **特点**：梯度下降不需要计算复杂的二阶导数，因此在高维空间中相对容易实现。然而，它的收敛速度通常较慢，特别是当目标函数的等高线呈现出椭圆而非圆形时（即存在条件数大的情况）。

>

> **相同与不同**

>

> - **相同点**：两者都可用于优化问题，试图找到函数的极小值点；都需要目标函数至少一阶可导。

> - **不同点**：

> - 牛顿法使用了更多的局部信息（即二阶导数），因此理论上收敛速度更快，但在实际应用中可能会遇到计算成本高、难以处理大规模数据集等问题。

> - 梯度下降则更为简单，易于实现，特别是在高维空间中，但由于只使用了一阶导数信息，其收敛速度可能较慢，尤其是在接近极值点时。

>

思路：定义函数及其导数，并根据公式 x_new = x - f(x)/df(x)不断更新 x，当迭代的误差小于设定的容忍度时则收敛，返回 x_new，否则当导数为 0 或超过最大迭代次数则退出或返回 None。

代码：

```
def f(x): # 原函数
    return (x)**3-5*(x**2)+(10*x)-80


def df(x): # 导数
    return 3*(x**2)-(10*x)+10


def newton(x0,tolerance=1e-7,max_iter=1000):
    x=x0 # 初始值
    for i in range(max_iter): # 最大迭代次数
        fx=f(x)
        dfx=df(x)
        x_new = x - (fx / dfx) # 牛顿迭代法公式
        if dfx == 0:
```

```
            break
        if abs(x_new - x) < tolerance: # 误差越来越小
            return x_new
        x=x_new
    return None # 达到最大迭代次数，没找到合适的x_new


x0=1.0
ans=newton(x0)
if ans is not None:
    print(f"{ans:.9f}")
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

大约用时：1 小时 30 分钟

06640: 倒排索引

data structures, http://cs101.openjudge.cn/practice/06640/

思路：将输入的句子存入 doc 列表中，查询的单词存入 list 列表中；接着初始化字典，遍历 doc 列表中的单词并从索引 1 开始，将单词分开去重，将不在 dict 字典中的单词及查找到的索引以列表形式存入该字典，如果查找的单词存在则输出，不存在则输出 NOT FOUND！。

代码：

```
n=int(input())
doc=[]
for _ in range(n):
    sentence=input()
    doc.append(sentence[1:])
m=int(input())
list=[]
for _ in range(m):
    word=input()
    list.append(word)
dict={} # 初始化字典
for i,j in enumerate(doc,1): # 索引号从 1 开始
    word=j.split() # 将单词一一分开
    words=set(word)
    for w in words:
        if w not in dict: # 不在字典里的单词存入字典
            dict[w]=[]
        dict[w].append(i)
for i in list:
```

```
    if i in dict:
        print(" ".join(map(str,dict[i])))
    else:
        print("NOT FOUND!")
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

大约用时：45 分钟

04093: 倒排索引查询

data structures, http://cs101.openjudge.cn/practice/04093/

思路：初始化字典用于存放每个词的文档编号，all 用于记录所有出现过的文档编号，然后获取单词的所有文档编号存入集合 docs，再将其对应的编号存入字典，用并集将 docs 和 all 合并，之后对每个条件进行查询，如果为 1 则保留包含该词的文档，-1 则

去除，最后如果 res 为空则没有符合条件的文档，输出 NOT FOUND，否则将 res 中的文档编号升序输出。

代码：

```
N=int(input())
dict={} # 初始化字典
all=set() # 保存所有出现的文档编号
for _ in range(1,N+1): # 从索引 1 开始记入字典
    data=list(map(int,input().split()))
    docs=set(data[1:])
    dict[_]=docs
    all |= docs # 并集，收集所有文档

M=int(input())
for _ in range(M):
    con=list(map(int,input().split()))
    res=all.copy() # 初始化为所有出现过的文档
    for i,j in enumerate(con,1): # 同样从索引 1 开始
        word=dict[i]
        if j == 1:
            res &= word # 交集，保留包含当前单词
        # if j == 0:
        #     pass
        elif j == -1:
            res -= word # 差集，去除包含当前单词
    if not res: # 没有符合条件的
        print("NOT FOUND")
```

```
else:
    print(" ".join(map(str,sorted(res))))
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

#48406752提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
N=int(input())
dict={} # 初始化字典
all=set() # 保存所有出现的文档编号
for _ in range(1,N+1): # 从索引1开始记入字典
    data=list(map(int,input().split()))
    docs=set(data[1:])
    dict[_]=docs
    all |= docs # 并集，收集所有文档

M=int(input())
for _ in range(M):
    con=list(map(int,input().split()))
    res=all.copy() # 初始化为所有出现过的文档
    for i,j in enumerate(con,1): # 同样从索引1开始
        word=dict[i]
        if j == 1:
            res &= word # 交集，保留包含当前单词
        # if j == 0:
        #     pass
        elif j == -1:
            res -= word # 差集，去除包含当前单词
    if not res: # 没有符合条件的
        print("NOT FOUND")
    else:
        print(" ".join(map(str,sorted(res))))
```

基本信息

#: 48406752
题目: 04093
提交人: 2400093012 苏倩仪
内存: 9308kB
时间: 59ms
语言: Python3
提交时间: 2025-03-01 22:07:19

©2002-2022 POJ 京ICP备20010980号-1 English 帮助 关于

大约用时：1 小时 30 分钟

2. 学习总结和个人收获

<mark>如果发现作业题目相对简单，有否寻找额外的练习题目，如“数算 2025spring 每日选做”、LeetCode、Codeforces、洛谷等网站上的题目。</mark>

通过这次作业，我复习了去年的矩阵运算，并且这次的作业在之前的基础上提高了一定的难度，尝试了乘法以及卷积运算，让我面对这类的矩阵运算有了一定的基础。此外在第三题中，我也尝试写了牛顿迭代法，明白了其收敛的原理；而且也尝试了倒排索引，明白了倒排索引的原理，也进一步学习了在某个单词是否“出现”、“不出现”及

“无所谓”的条件下进行倒排索引查询的思路，虽然对我来说有点困难，但是通过这次练习，我大致理解了 python 中集合的使用。