中山大学计算机学院算法设计与分析本科生实验报告

课程名称：算法设计与分析

教学班级	专业（方向）	学号	姓名
2班	计算机科学与技术	21307174	刘俊杰

一、实验内容

作业

对于给定的 N 本书和 M 个学生，每本书的页数已经按升序排列。我们的任务是分配这些书，使得分配给每个学生的最大阅读页数达到最小，并且每个学生需要阅读升序排列上连续的书。输出最小页数。

例子：

输入：N=4，页数[] = {12, 34, 67, 90}, M = 2

输出：113。

二、实现思想

实验原理

基本思想

- 可以将每一位学生可以读的书的页数看作一个桶或者一个包可容纳物品的个数，每本书可以看做一堆堆物品的个数，题目要求就变为按升序将

每一堆物品放入桶或背包内，求最小容量。因为最小容量是这些学生中的最大可读书页数，我们不妨设每个学生的读书页数都是最大可读书页数，这样在满足最小容量的基础上可以更容易完成任务。

- 这样我们的目标就是找到最小的每个学生的读书的页数。为了保证可读完所有书，所以最小容量至少要大于等于所有书页数的最大值；又为了求最小容量，最小容量要小于等于页数的总和，否则一个学生就可以读完所有书。
- 这样之后我们可以暴力遍历搜索空间[max(Pages),sum(Pages)]来搜索最小容量；在这里我们可以使用二分搜索来更快地搜索最小容量。

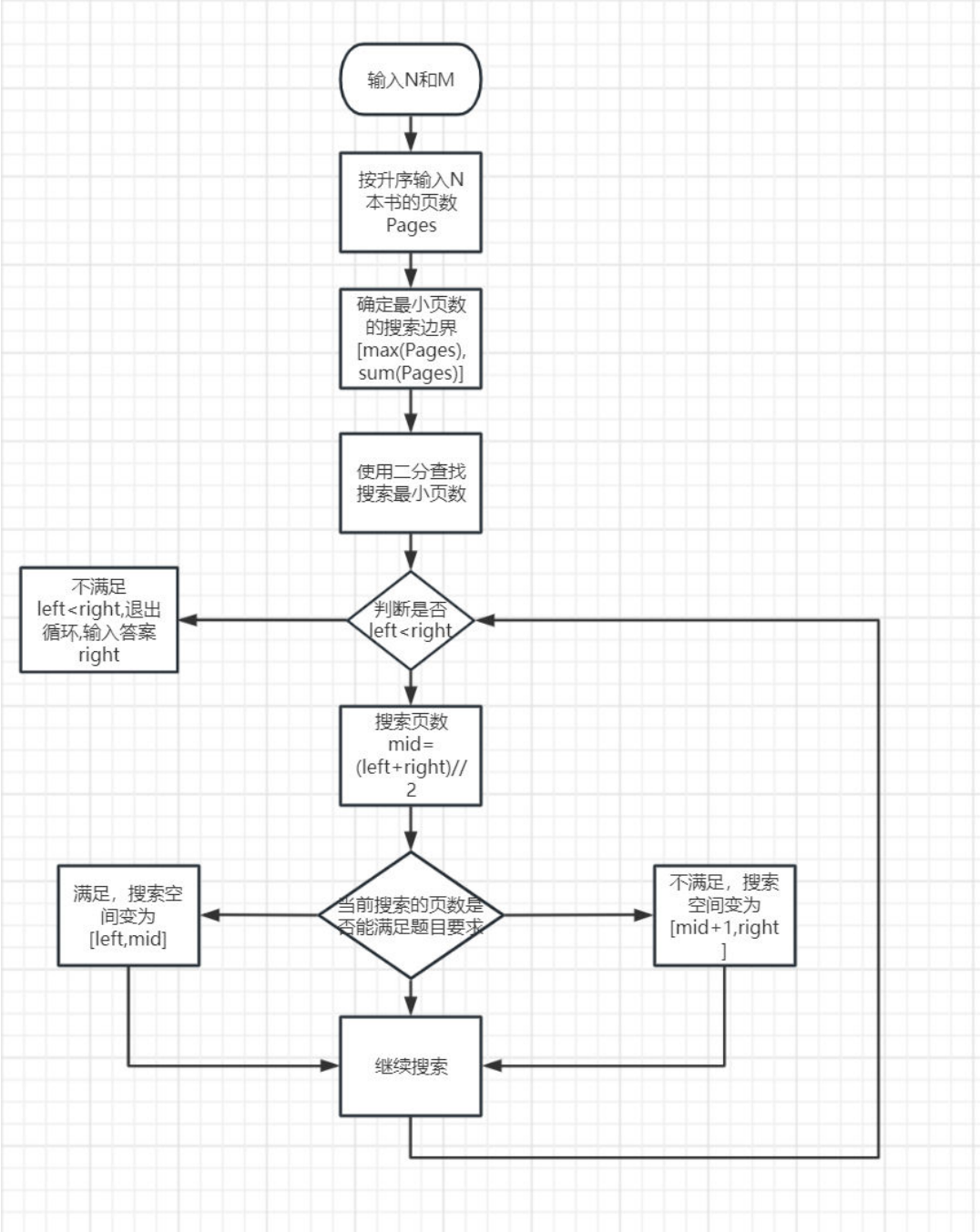
二分搜索实现思想

- 初始化搜索空间[left,right]，left=max(Pages),right=sum(Pages)。
- 进入循环，若left< right不成立则退出循环，输出right。mid = (left+right)//2，判断当前搜索容量mid能否满足题目要求,若可以则right=mid,搜索空间为[left,mid];否则left=mid+1,搜索空间为[mid+1，right]。

判断当前搜索容量能否满足题目要求

- 初始化已经在读书或读完书的学生students为1，当前学生还能读的书的页数count_pages为搜索容量。
- 遍历书的页数，当前遍历到的书的页数为page，若当前学生还能读的书的页数count_pages>=page，则这个学生可读完，则count_pages-=page;否则，说明这个学生读不完这本书，换下一个学生读书,students+=1，count_page=read_page，count_page-=page。
- 最后求出的student是在当前搜索容量下能读完书需要的学生数。若student<=M，则当前容量满足要否则不满足。

流程图


输入N和M
按升序输入N本书的页数Pages
确定最小页数的搜索边界[max(Pages),sum(Pages)]
使用二分查找搜索最小页数
判断是否left<right
不满足left<right,退出循环,输入答案right
搜索页数mid=(left+right)//2
当前搜索的页数是否能满足题目要求
满足，搜索空间变为[left,mid]
不满足，搜索空间变为[mid+1,right]
继续搜索

三、源代码(带有注释)

```
def isOK(Pages,M,read_page):#检测每个学生读read_page页书能否按照题目要求读完
    count_page = read_page#当前读书的学生还能读的页数
    students = 1#读完以及还在读书的学生数量
    for i in range(0,len(Pages)):
        page = Pages[i]
        if page <= count_page:#当前学生能读完这本书
            count_page -= page

        else:#当前学生读不完这本书，换下一个学生读
            count_page = read_page
            count_page -= page
            students += 1
    if students<=M:#学生人数能在不超过M的情况下读完
        return True
    return False

def find_min_page(Pages,M):#计算最小页数
    if len(Pages)==0 :#没有书可读 返回-
        return 0
    if M==0:#没有学生
        print("No solution!")
        return -1
    #min_page = max(Pages),max_page = sum(Pages)
    min_page,max_page=Pages[0],0
    for i in range(len(Pages)):
        min_page = max(min_page,Pages[i])
        max_page += Pages[i]

    #二分查找能满足要求的最小页数
    left , right = min_page,max_page
    while left<right:#停止循环条件
        mid = (left+right)//2
        #print(left,right,mid,isOK(Pages,M,mid))
        if isOK(Pages,M,mid):#当前容量能读完 搜索空间为[left,mid]
            right = mid
        else :#当前容量不能读完 搜索空间为[mid+1,right]
            left = mid + 1
    return right
```

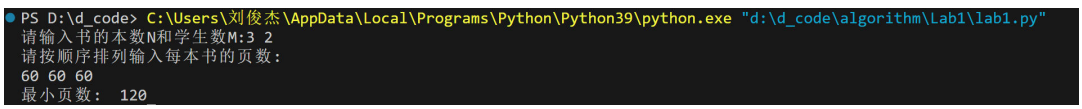
```
def main():
    #处理输入数据
    Pages = []
    input_NM = input("请输入书的本数N和学生数
M:").strip().split(' ')
    N,M = int(input_NM[0]),int(input_NM[1])
    print("请按顺序排列输入每本书的页数:")
    input_page = input().strip().split()
    for i in range(N):
        Pages .append(int(input_page[i]))
    #输出答案
    print("最小页数: ",find_min_page(Pages,M))

if __name__ == "__main__":
    main()
```

四、输出截图

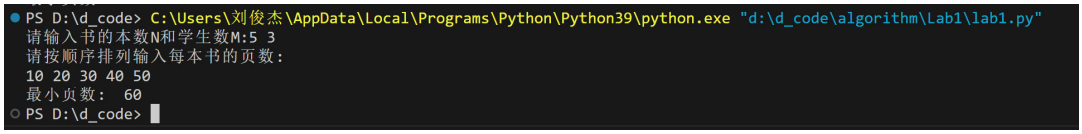
PPT上的示例:

测试1

测试2



测试3



通过计算分析可得，每次的输出都是正确的

五、实验分析

时间复杂度分析:

每次二分搜索，搜索范围为[max(Pages),sum(Pages)]，将sum(Pages)-max(Pages)设为P ，且每次搜索判断是否能读完时需要遍历书的页数数组，判断的时间复杂度为O(N)，因此总的时间复杂度为O(Nlog(P))。

空间复杂度分析:

空间主要消耗在存储书的页数的数组，所以空间复杂度为O(M)。