二分与贪心算法

郭炜 刘家瑛

POJ 1505 誊抄书籍

北京大学

问题描述

- 有 m 本书需要誊抄, 每本书的页数分别是 (p_1, p_2, …, p_m)
- 有 k (k <= m) 个抄写员负责誊抄这些书籍
- 任 务
  - 将这些书分成 k 份, 每本书必须只分给一个抄写员
  - 每个抄写员至少分到一本
  - 要求每个抄写员分到的书的编号是连续的
  - 即存在一个连续升序数列 $0=b_0 < b_1 < b_2 < \ldots < b_{k-1} < b_k=m$

  → 这样, 第 i 号抄写员得到的书稿是从 $b_{i-1}+1$ 到第 b_i 本书

问题描述

- 复制工作是同时开始进行的
- 且每个抄写员复制的速度都是一样的
- 复制完所有书稿所需时间

→ 取决于分配得到 **最多工作** 的那个抄写员的复制时间

- 现要求分到页数最多的抄写员需要抄写的页数尽可能少，求最优的分配方式

程序输入

- 输入包含N组数据
- 每组数据包含两行
  - 第一行是两个整数m和k ($1 \le k <= m \le 500$),
    分别表示书的数量和抄写员的数量
  - 第二行有m个数 $p_1, p_2, \ldots, p_m$, 分别表示各本书的页数,
    这些页数都是正整数且小于 10000000

程序输出

- 对于每组数据，输出一行
  - 输出让分配最多页数的抄写员的页数最小的方案
  - 在两个抄写员分配的书之间插入 ‘/’
  - 如果存在多种最优分配方案，则输出让第一个抄写员尽可能拷贝页数少，然后第二个尽可能少等
  - 但是每个人必须分到至少一本书

- 样例输入

2

9 3

100 200 300 400 500 600 700 800 900

5 4

100 100 100 100 100

- 样例输出

100 200 300 400 500 / 600 700 / 800 900

100 / 100 / 100 / 100 100

解题思路

- 题目要求—使得最大的一堆书的页数和最小
  - 这个问题看起来并不容易求, 尝试简化问题
- 问题转换: 最小值问题→判定性问题 (常用方法)
  - 判断最大的一堆书的页数小于等于X, 是否可行
- 目标就转化为: 将X从小到大枚举, 依次判断是否可行, 直到找到第一个可行的X, 就是答案
  - 问题: 效率太低！
  - 如何解决这个问题?

解题思路

- 由小到大枚举找到满足条件的值, 依次判断
- 可以用二分加快速度
  - 如果小于等于X可行, 那么小于等于Y(Y<=X)就都可行,
  - 如果小于等于X不可行, 那么小于等于Y(Y <= X)就都不可行
  - 满足单调性, 满足二分的条件!
- 经典思想: 二分+判定

解题思路

- 判定性问题:

  如何判断最大的一堆书的页数小于等于X是否可行?

  - 贪心策略:
  - 要想使最大页数尽可能小, 那么每一堆书应该尽量多放
  - 注意题干的条件: 每一堆书都是连续的
  - 分配的策略是连续地依次分配, 直到页数超过X,
    再分配下一堆

解题思路

- 判定规则为:
  - 可以从后往前依次判断
  - 如果加入当前这本书没有超过X, 就加入
  - 否则就另起一堆
  - 如果最后的堆数小于K, 则可行

解题思路

- 二分查找 搜索范围
  - 最小值:

    最多的一本书的页数（因为每个人至少分配一本书）
  - 最大值: 总页数

解题思路

- 用二分法求出满足条件最小的X之后

→ 最后的问题就是如何输出

- 要求分给前面的抄写员的页数尽可能少

→ 后面的抄写员尽可能多

- 所以和之前一样，从后往前依次判断，
- 能加入的就尽量放入当前堆
- 因为从后往前判断，输出又是从前往后
  - 栈的思想→所以可以使用 递归的算法输出

重点函数分析

- 判断函数

```
bool check(X){
    当前堆的页数 = 0; 总堆数 = 0;
    for (从最后一本书到第一本书) {
        if (当前堆的页数+当前书的页数 > X) {
            总堆数 +1;
            当前堆的页数=当前书的页数; //当前书进入下一堆
        }else 当前堆的页数 += 当前书的页数;
    }
    if (当前堆的页数 > 0) 总堆数 +1; //注意别忘了最后一堆
    //判断最后的总堆数是否小于等于k
    return 总堆数 <= 抄写员数
}
```

重点函数分析

- 二分查找

```
while (l <= r){
    mid = (l+r)/2;
    if (check(mid)) r = mid - 1;
    else l = mid + 1;
}
```

  - 结果取l,r,mid中哪一个
    - 退出条件是 l>r，此时
      - 若 mid 满足条件，则l=mid=x
      - 若 mid 不满足条件,则l=mid+1=x-1+1=x
    - 结果选取l

重点函数分析

- 递归输出

```
//当前处理到第book本书，第scriber堆，当前堆的页数为now，每堆页数不能超过ans
void print( int book, int ans, int scriber, long long now){
      bool sepa = false;
      if (book < 0) return ;
      //若当前书不能放入当前堆，则说明要加入分隔符
      if ( book == scriber-1 || now+a[book] > ans) {
          print( book-1, ans, scriber-1, a[book]);
           sepa = true;
      }else print( book-1, ans, scriber, now+a[book]);
      //输出当前书的页数和分隔符
      if ( book > 0 ) printf(" %d", a[book]);
      else printf("%d", a[book]);
      if (sepa) printf("/");
}
```

重点函数分析

- `book == scriber-1`
  - 每个抄写员至少分配1本，否则将这本放入scriber的话，
  - 还剩book=scriber-1本书，还有scriber个抄写员未分配
- `else printf("%d", a[book]);`
  - 第0本书前面没有空格

参考程序

```
#include <stdio.h>
#include <iostream>
using namespace std;

#define MAXM (500+10)

int N,  m,  k;
//每本书的页数
int a[MAXM];
```

参考程序

```
//判断最大堆的页数小于等于ans是否可行
bool check( long long ans ) {
    int s = 0;
    long long now = 0;

    //从后向前判断每本书，若可以就放入当前堆
    for (int i = m-1;i >= 0;--i) {
     if (now + a[i] > ans) {
         ++s;
         now = a[i];
     }else now += a[i];
    }
    if (now > 0) ++s;
    //判断最后的总堆数是否小于等于k
    if (s > k) return false;
    else return true;
}
```

参考程序

```
//用递归的方式输出最后的结果
//当前处理到第book本书，第scriber堆，当前堆的页数为now，每堆页数不能超过ans
void print( int book, int ans, int scriber, long long now){
   bool sepa = false;

   if (book < 0) return ;

    //若当前书不能放入当前堆，则说明要加入分隔符
   if (book == scriber-1 || now+a[book] > ans) {
    print( book-1, ans, scriber-1, a[book]);
    sepa = true;
   }else print( book-1, ans, scriber, now+a[book]);
```

参考程序

```
  //输出当前书的页数和分隔符
    if (book > 0) printf(" %d",  a[book]);
    else printf("%d",  a[book]);
    if (sepa) printf(" /");
}
```

参考程序

```
int main() {
    long long l, r, mid;
    scanf("%d", &N);
    for (int i = 0;i < N; ++i) {
        l = 0; r = 0;
        scanf("%d%d", &m, &k);
        for (int j = 0;j < m; ++j){
            scanf("%d",  &a[j]);
            r += a[j];
            if (a[j] > l) l = a[j];
        }
```

参考程序

```
        //二分查找，判断最大堆的页数小于等于mid是否可行
        while (l <= r){
            mid = (l+r)/2;
            if (check(mid)) r = mid - 1;
            else l = mid + 1;
        }
        //递归输出最后结果
        print( m-1,  l,  k-1,  0 );
        printf("\n");
    }
    return 0;
}
```