1278：复制书稿(book)

题目描述

现在要把m本有顺序的书分给k个人复制（抄写），每一个人的抄写速度都一样，一本书不允许给两个（或以上）的人抄写，分给每一个人的书，必须是连续的，比如不能把第一、第三和第四本书给同一个人抄写。

现在请你设计一种方案，使得复制时间最短。复制时间为抄写页数最多的人用去的时间。

输入

第一行两个整数m，k；（k≤m≤500）

第二行m个整数，第i个整数表示第i本书的页数。

输出

共k行，每行两个整数，第i行表示第i个人抄写的书的起始编号和终止编号。k行的起始编号应该从小到大排列，如果有多解，则尽可能让前面的人少抄写。

输入样例

```
9 3
1 2 3 4 5 6 7 8 9
```

输出样例

```
1 5
6 7
8 9
```

解析

这是一道区间DP问题，看到区间DP，首先要想到前缀和。在这里，我们使用前缀和数组sum[i]来记录抄写第1本到第m本书所用的总时间，根据测试数据，我们可以得出：

索引	1	2	3	4	5	6	7	8	9
前缀和	1	3	6	10	15	21	28	36	45

将m本书分给k个人，我们一上来是无法直接计算的，首先需要缩小规模。

我们设dp[i][j]表示前i个人抄写j本书的最少时间。

根据分析，我们知道无论如何最后一个人都会抄写t本书，所用的时间即为sum[j]-sum[j-t]。

除去最后一个人抄写的情况后，我们还剩下dp[i-1][j-t]。如下图所示：

此时，我们需要比较一下前i-1个人抄写j-t本书的时间和最后一个人抄写的时间哪个更大，这个更大的时间便是在这种情况下真实的使用时间。即：

max(dp[i-1][j-t],sum[j]-sum[j-t])。

为了得到最优结果，于是我们不断的调整t的情况，例如只抄写1本，抄写2本……，然后再将不同情况下的数值与最值进行比较，即可得出结果。核心代码如下：

编码

```
#include <bits/stdc++.h>

using namespace std;

int a[501];  //最多500本书
int sum[501];  //前缀和
int m, k;  //定义书的数量和人数
int dp[500][500];  //第一维代表书的数量，第二维代表几个人抄书

//打印组合情况
//i为书的数量,j为人数
void print(int i, int j) {
    int t, x;
    //没有人可以分配了
    if (j == 0) return;
    //只剩余一人，就打印全部
    if (j == 1) {
```

```
            cout << 1 << " " << i << endl;
            return;
        }
        t = i; //从最后一本书开始尝试
        x = a[i]; //抄写第i本书所需要的时间
        //从最后一本书按逆序分配第j个人抄写，只要能写，就给他
        while (x + a[t - 1] <= dp[m][k]) {
            //累加抄书的时间，如果不大于当前的dp值，就继续交给这个人抄写
            x = x + a[t - 1];
            //向前推进一本书
            t--;
        }
        print(t - 1, j - 1);
        //打印起始的书籍编号
        cout << t << " " << i << endl;
    }


    int main(int argc, char **argv) {
        //因为比较的是最小值，所以一开始需要把数组的数据设置为最大
        memset(dp, 0x3f, sizeof(dp));
        cin >> m >> k;
        for (int i = 1; i <= m; ++i) {
            //输入每本书的页数
            cin >> a[i];
            //计算前缀和
            sum[i] = sum[i - 1] + a[i];
        }
        //处理边界情况，1个人读前i本书所用的时间
        for (int i = 1; i <= m; ++i) {
            dp[i][1] = sum[i];
        }
        //遍历m本书
        for (int i = 1; i <= m; ++i) {
            //遍历k个人
            //为什么不从
            for (int j = 2; j <= k; ++j) {
                //遍历最后一个人读t本书的状态
                for (int t = 0; t <= i; ++t) {
                    //比较在不同的情况下所需要的最长时间
                    int value = max(dp[i - t][j - 1], sum[i] - sum[i - t]);
                    //在最长时间中找到最短的时间
                    dp[i][j] = min(value, dp[i][j]);
                }
            }
        }
        print(m, k);
        return 0;
    }
```