题目描述

区块链底层存储是一个链式文件系统，由顺序的N个文件组成，每个文件的大小不一，依次为F1,F2,...,Fn。随着时间的推移，所占存储会越来越大。

云平台考虑将区块链按文件转储到廉价的SATA盘，只有连续的区块链文件才能转储到SATA盘上，且转储的文件之和不能超过SATA盘的容量。

假设每块 SATA 盘容量为M，求能转储的最大连续文件之和。

输入描述

第一行为SATA盘容量M，1000 ≤ M ≤ 1000000

第二行为区块链文件大小序列F1,F2,...,Fn。其中 1 ≤ n ≤ 100000，1 ≤ Fi ≤ 500

输出描述

求能转储的最大连续文件大小之和

用例

输入	1000 100 300 500 400 400 150 100
输出	950
说明	最大序列和为950，序列为[400,400,150]

输入	1000 100 500 400 150 500 100
输出	1000
说明	最大序列和为1000，序列为[100,500,400]

题目解析

由于本题需要求解最大连续文件大小之和，因此可以考虑使用双指针+滑动窗口来解题。

本题的滑动窗口的左边界l,右边界r的运动逻辑如下：

- 如果滑动窗口内部和 < m，则r++
- 如果滑动窗口内部和 > m，则l++
- 如果滑动窗口内部和 = m，则已得到最大值，直接返回m即可。

在计算滑动窗口内部和的过程中，如果r++，则说明内部和可能会增大产生最大值，因此我们需要在r++时，判断并保留最大值。

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    const m = lines[0] - 0;
    const F = lines[1].split(" ").map(Number);

    console.log(getResult(m, F));
    lines.length = 0;
  }
});
```

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 2) {
    const m = lines[0] - 0;
    const F = lines[1].split(" ").map(Number);

    console.log(getResult(m, F));
    lines.length = 0;
  }
});

function getResult(m, F) {
  let l = 0,
    r = 0;

  let n = F.length;

  let sum = 0;
  let max = 0;

  while (r < n) {
    // 尝试右指针右移一下的新和（注意初始时右指针右移后指向0）
    const newSum = sum + F[r];

    // 如果新和超过了m，即SATA盘容量，则右指针不能右移，并且还需要左指针右移来减少旧和
    if (newSum > m) {
      sum -= F[l++]; // 左指针右移只会减少旧和，因此不会产生最大值
    }
    // 如果新和小于m，则当前尝试的右指针右移可行，因此 sum += F[r]，并且我们下一步还可以继续尝试让右指针右移，即r++
    else if (newSum < m) {
      sum += F[r++];
      max = Math.max(sum, max); // 右指针右移时会增加旧和，因此可能会产生最大值
    }
    // 如果新和等于m，那么说明已经找到了一个容量和SATA盘相同的连续文件大小，即此时已经是最大值了，可以直接返回
    else {
      return m;
    }
  }

  return max;
}
```

Java算法源码

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int m = Integer.parseInt(sc.nextLine());
    Integer[] f =
        Arrays.stream(sc.nextLine().split(" ")).map(Integer::parseInt).toArray(Integer[]::new);

    System.out.println(getResult(m, f));
  }

  public static int getResult(int m, Integer[] f) {
    int l = 0, r = 0;
    int sum = 0, max = 0;

    int n = f.length;

    while (r < n) {
      // 尝试右指针右移一下的新和（注意初始时右指针右移后指向0）
      int newSum = sum + f[r];

      // 如果新和超过了m，即SATA盘容量，则右指针不能右移，并且还需要左指针右移来减少旧和
      if (newSum > m) {
        sum -= f[l++]; // 左指针右移只会减少旧和，因此不会产生最大值
      }
      // 如果新和小于m，则当前尝试的右指针右移可行，因此 sum += F[r]，并且我们下一步还可以继续尝试让右指针右移，即r++
      else if (newSum < m) {
        sum += f[r++];
        max = Math.max(sum, max); // 右指针右移时会增加旧和，因此可能会产生最大值
      }
      // 如果新和等于m，那么说明已经找到了一个容量和SATA盘相同的连续文件大小，即此时已经是最大值了，可以直接返回
      else {
        return m;
```

Java算法源码

```java
import java.util.Arrays;
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int m = Integer.parseInt(sc.nextLine());
    Integer[] f =
        Arrays.stream(sc.nextLine().split(" ")).map(Integer::parseInt).toArray(Integer[]::new);

    System.out.println(getResult(m, f));
  }

  public static int getResult(int m, Integer[] f) {
    int l = 0, r = 0;
    int sum = 0, max = 0;

    int n = f.length;

    while (r < n) {
      // 尝试右指针右移一下的新和（注意初始时右指针右移后指向0）
      int newSum = sum + f[r];

      // 如果新和超过了m，即SATA盘容量，则右指针不能右移，并且还需要左指针右移来减少旧和
      if (newSum > m) {
        sum -= f[l++]; // 左指针右移只会减少旧和，因此不会产生最大值
      }
      // 如果新和小于m，则当前尝试的右指针右移可行，因此 sum += F[r]，并且我们下一步还可以继续尝试让右指针右移，即r++
      else if (newSum < m) {
        sum += f[r++];
        max = Math.max(sum, max); // 右指针右移时会增加旧和，因此可能会产生最大值
      }
      // 如果新和等于m，那么说明已经找到了一个容量和SATA盘相同的连续文件大小，即此时已经是最大值了，可以直接返回
      else {
        return m;
      }
    }

    return max;
  }
}
```

Python算法源码

```python
m = int(input())
f = list(map(int, input().split()))


def getResult(m, f):
    l = 0
    r = 0
    n = len(f)

    sum = 0
    maxV = 0

    while r < n:
        # 尝试右指针右移一下的新和（注意初始时右指针右移后指向0）
        newSum = sum + f[r]

        # 如果新和超过了m，即SATA盘容量，则右指针不能右移，并且还需要左指针右移来减少旧和
        if newSum > m:
            # 左指针右移只会减少旧和，因此不会产生最大值
            sum -= f[l]
            l += 1
        # 如果新和小于m，则当前尝试的右指针右移可行，因此 sum += F[r]，并且我们下一步还可以继续尝试让右指针右移，即r++
        elif newSum < m:
            sum += f[r]
            r += 1
            maxV = max(sum, maxV) # 右指针右移时会增加旧和，因此可能会产生最大值
        # 如果新和等于m，那么说明已经找到了一个容量和SATA盘相同的连续文件大小，即此时已经是最大值了，可以直接返回
        else:
            return m

    return maxV


print(getResult(m, f))
```