【DP】-通过软盘拷贝文件

题目描述与示例

题目描述

有一名科学家想要从一台古董电脑中拷贝文件到自己的电脑中加以研究。

但此电脑除了有一个3.5寸软盘驱动器以外，没有任何手段可以将文件持贝出来，而且只有一张软盘可以使用。

因此这一张软盘是唯一可以用来拷贝文件的载体。

科学家想要尽可能多地将计算机中的信息拷贝到软盘中，做到软盘中文件内容总大小最大。

已知该软盘容量为 `1474560` 字节。文件占用的软盘空间都是按块分配的，每个块大小为 `512` 个字节。一个块只能被一个文件使用。拷贝到软盘中的文件必须是完整的，且不能采取任何压缩技术。

输入描述

第 `1` 行为一个整数 `N` ，表示计算机中的文件数量。 `1 ≤ N ≤ 1000` .

接下来的第 `2` 行到第 `N+1` 行(共 `N` 行)，每行为一个整数，表示每个文件的大小 `Si` ，单位为 `字节` 。

`0 ≤ i < N` , `0 ≤ Si <= 1474560`

输出描述

科学家最多能拷贝的文件总大小

备注

为了充分利用软盘空间，将每个文件在软盘上占用的块记录到本子上。即真正占用软盘空间的只有文件内容本身。

示例一

输入

```
3
737270
```

```
3 737272
4 737288
```

输出

```
1 1474542
```

说明

3个文件中，每个文件实际占用的大小分别为 `737280` 字节、 `737280` 字节、 `737792` 字节。只能选取前两个文件，总大小为 `1474542` 字节。虽然后两个文件总大小更大且未超过 `1474560` 字节，但因为实际占用的大小超过了 `1474560` 字节，所以不能选后两个文件。

示例二

输入

```
1 6
2 400000
3 200000
4 200000
5 200000
6 400000
7 400000
```

输出

```
1 1400000
```

说明

从 `6` 个文件中，选择 `3` 个大小为 `400000` 的文件和 `1` 个大小为 `200000` 的文件，得到最大总大小为 `1400000` 。

也可以选择 `2` 个大小为 `400000` 的文件和 `3` 个大小为 `200000` 的文件，得到的总大小也是 `1400000`

解题思路

我们需要在 `N` 个文件中挑选出若干个文件，使得总文件大小尽可能地大，因此本题属于一个典型的背包问题。

但本题中存在一个坑，**文件必须按照块进行储存，且一个块只能储存一个文件**。块的大小是 `512` 。

假设此时要储存一个大小为 `513` 的文件，就必须使用 `2` 个块来储存，且由于第二个块即使没有填满也不能再用于其他文件的存储，故占用的空间为 `1024` 。

由于软盘总容量为 `1474560` 字节，一共包含 `1474560 // 512 = 2880` 个块，而每一个文件所占用的块数，也可以很方便地通过 `ceil(file_i / 512)` 得到，故我们可以将问题进行转化为如下描述：

对于已知容量为 `2880` 的背包，我们知道每一个文件所占空间 `ceil(file_i / 512)` 和价值 `file_i` ，我们如何进行选择使得背包的总价值尽可能地大。

这就将题干描述的复杂情况，转化为了一个相对容易解决的**路径无关、求和最大值的01背包问题了**。其解法类似于【DP】2023A-工作安排

代码

Python

```python
# 题目：2024E-通过软盘拷贝文件
# 分值：200
# 作者：许老师-闭着眼睛学数理化
# 算法：背包DP
# 代码看不懂的地方，请直接在群上提问


from math import ceil


# 输入文件数目
n = int(input())
# 创建价值列表和空间列表
# 价值列表储存每一个文件的有效内容的字节数
# 空间列表储存每一个文件实际占用的块数
value_lst = list()
space_lst = list()

# 遍历n次
for _ in range(n):
```

```
    value = int(input())
    value_lst.append(value)
    space_lst.append(ceil(value / 512))

# 背包容量，即 1474560 // 512 = 2880 个块
m = 2880
# 构建长度为(m+1)的dp数组
dp = [0] * (m+1)

# 路径无关，先遍历每一个文件，后遍历背包容量
for i in range(n):
    # 分别获得第i份文件的实际内容字节数value和所占用块数space
    value = value_lst[i]
    space = space_lst[i]
    # 01背包，逆序遍历背包的前置容量pre_space
    # 遍历的起点可以从m-space开始，
    # 这样可以保证每个pre_space+space都不会超过m
    for pre_space in range(m-space, -1, -1):
        # 获取pre_space的对应最大价值pre_value
        pre_value = dp[pre_space]
        # 从前置容量pre_space出发，加上当前文件容量space
        # 所占据的总容量为cur_space
        cur_space = pre_space + space
        # 如果先前用pre_space空间能够获得的有效字节数pre_value
        # 加上当前有效字节数value
        # 比原先的dp[cur_space]更大（默认为0），则更新dp[cur_space]
        dp[cur_space] = max(dp[cur_space], pre_value + value)


# 输出dp数组中的最大值，即为答案
print(max(dp))
```

Java

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int[] value_lst = new int[n];
        int[] space_lst = new int[n];

        for (int i = 0; i < n; i++) {
            int value = scanner.nextInt();
```

```
            value_lst[i] = value;
            space_lst[i] = (int) Math.ceil(value / 512.0);
        }

        int m = 2880;
        int[] dp = new int[m + 1];

        for (int i = 0; i < n; i++) {
            int value = value_lst[i];
            int space = space_lst[i];
            for (int pre_space = m - space; pre_space >= 0; pre_space--) {
                int pre_value = dp[pre_space];
                int cur_space = pre_space + space;
                dp[cur_space] = Math.max(dp[cur_space], pre_value + value);
            }
        }

        int maxResult = 0;
        for (int i = 0; i <= m; i++) {
            maxResult = Math.max(maxResult, dp[i]);
        }

        System.out.println(maxResult);
    }
}

```

C++

```
#include <iostream>
#include <vector>
#include <cmath>
#include <algorithm>

using namespace std;

int main() {
    int n;
    cin >> n;
    vector<int> value_lst(n);
    vector<int> space_lst(n);

    for (int i = 0; i < n; i++) {
        int value;
        cin >> value;
```

```
        value_lst[i] = value;
        space_lst[i] = ceil(static_cast<double>(value) / 512);
    }

    int m = 2880;
    vector<int> dp(m + 1, 0);

    for (int i = 0; i < n; i++) {
        int value = value_lst[i];
        int space = space_lst[i];
        for (int pre_space = m - space; pre_space >= 0; pre_space--) {
            int pre_value = dp[pre_space];
            int cur_space = pre_space + space;
            dp[cur_space] = max(dp[cur_space], pre_value + value);
        }
    }

    int maxResult = 0;
    for (int i = 0; i <= m; i++) {
        maxResult = max(maxResult, dp[i]);
    }

    cout << maxResult << endl;

    return 0;
}

```

时空复杂度

时间复杂度： `O(NM)` 。其中 `N` 为文件个数， `M` 为背包的最大容量，对于本题恒有 `M = 2880`

空间复杂度： `O(M)` 。1维 `dp` 数组所占空间。