合并果子 解题报告

<问题描述>

在一个果园里，多多已经将所有的果子打了下来，而且按果子的不同种类分成了不同的堆。多多决定把所有的果子合成一堆。

每一次合并，多多可以把两堆果子合并到一起，消耗的体力等于两堆果子的重量之和。可以看出，所有的果子经过 n-1 次合并之后，就只剩下一堆了。多多在合并果子时总共消耗的体力等于每次合并所耗体力之和。

因为还要花大力气把这些果子搬回家，所以多多在合并果子时要尽可能地节省体力。假定每个果子重量都为 1，并且已知果子的种类数和每种果子的数目，你的任务是设计出合并的次序方案，使多多耗费的体力最少，并输出这个最小的体力耗费值。

例如有 3 种果子，数目依次为 1，2，9。可以先将 1、2 堆合并，新堆数目为 3，耗费体力为 3。接着，将新堆与原先的第三堆合并，又得到新的堆，数目为 12，耗费体力为 12。所以多多总共耗费体力=3+12=15。可以证明 15 为最小的体力耗费值。

- 输入文件

输入文件 fruit.in 包括两行，第一行是一个整数 n(1<＝n<=10000)，表示果子的种类数。第二行包含 n 个整数，用空格分隔，第 i 个整数 ai(1<＝ai<=20000)是第 i 种果子的数目。

- 输出文件

输出文件 fruit.out 包括一行，这一行只包含一个整数，也就是最小的体力耗费值。输入数据保证这个值小于 2^{31}。

- 样例输入

```
3
1 2 9
```

- 样例输出

```
15
```

- 数据规模

对于30％的数据，保证有n<=1000：
对于50％的数据，保证有n<=5000；
对于全部的数据，保证有n<=10000。

<算法分析>

将这个问题换一个角度描述：给定n个**叶结点**，每个结点有一个权值W[i]，将它们中两个、两个合并为树，假设每个结点从根到它的距离是D[i]，使得**最终∑(wi + di)最小**。

于是，这个问题就变为了经典的Huffman树问题。Huffman树的构造方法如下：

(1) 从森林里取两个权和最小的结点

(2) 将它们的权和相加，得到新的结点，并且把原结点删除，将新结点插入到森林中

(3) 重复(1)，直到整个森林里只有一棵树。

这个方法的正确性可以参见数据结构。

<数据结构>

很显然，问题当中需要执行的操作是：(1) 从一个表中取出最小的数 (2) 插入一个数字到这个表中。

支持动态Extract_Min和Insert操作的数据结构，我们可以选择用堆来实现。堆是

一种完全二叉树，且保证根结点的值严格大于（或小于）其子孙结点。具体实现方法可以参见数据结构。

于是整体算法的时间复杂度为O(nlogn)，空间复杂度为O(n)。

但是，有没有更好的方法呢？很显然，每次合并两个结点以后，得到的大小是严格递增的，于是我们可以维护两个表，一个是原数字A，一个是新加入的数字B。这样，每次就一定是在A和B的头部取数，在A和B的尾部删除。这样，时间复杂度就降到了O(n)。因为a[i] <= 20000，所以排序也可以用o(20000)的方法来实现，整体时间复杂度为O(n)。（感谢BCBill提供这个方法）

<代码清单>

```
#include <fstream>
#include <list>
#include <algorithm>
using namespace std;

ifstream fin("fruit.in");
ofstream fout("fruit.out");

int n;
list <int> a, b;

void init() {
    int p;

    fin >> n;
    for (int i = 0; i < n; i ++) {
        fin >> p;
        a.push_back(p);
    }
    a.sort();
}

int get() {
    int ans;

    if (a.empty()) {
        ans = b.front(); b.pop_front(); return ans;
    }
    if (b.empty()) {
        ans = a.front(); a.pop_front(); return ans;
    }
    if (a.front() < b.front()) {
        ans = a.front(); a.pop_front(); return ans;
```

```
    }
    else {
        ans = b.front(); b.pop_front();   return ans;
    }
}

void work() {
    int p, sum = 0;
    for (int i = 0; i < n - 1; i ++) {
        p = get() + get();
        b.push_back(p);
        sum += p;
    }
    fout << sum << endl;
}

int main() {
    init();
    work();
    return 0;
}
```

<小结>

读清问题的描述是很重要的！很多选手都将这个问题看成了最小代价子母树。审清题目是解决问题的首要条件。当然，灵活地使用数据结构也是解决问题的关键。简单的线性表在这里充分地发挥了它的优势，使程序的效率得到了很大的提高。