第二次编程作业

姓名：寇一笑　　学号：18020024016　　姓名：安皓源　　学号：18020022001

2020 年 5 月 1 日

Contents

1 第一题

1.1 实验目的和内容

1.1.1 题目描述

农夫要修理牧场的一段栅栏，他测量了栅栏，发现需要N块木头，每块木头长度为整数Li个长度单位，于是他购买了一条很长的、能锯成N块的木头，即该木头的长度是Li的总和。但是农夫自己没有锯子，请人锯木的酬金跟这段木头的长度成正比。请编写程序帮助农夫计算将木头锯成N块的最少花费。

1.2 输入和输出说明

1.2.1 输入

第一行是输入木头块数N　　第二行是以N块木头的长度

1.2.2 输出

将木头锯成N块的最少花费以及切割的步骤

1.2.3 样例

- 输入

  8

  4 5 1 2 1 3 1 1

- 输出

  割18成为8和10

  切割10成为5和5

  切割8成为4和4

  切割5成为2和3

  切割4成为2和2

  切割2成为1和1

  切割2成为1和1

  49

- 输入

  1

  1

- 输出

1.3 解题思路与问题分析

根据题目不难得知，所需要的费用是逐层叠加的。所以我们需要把长的木头最先锯下来。这个题目我们采用锯木头的逆过程即把锯下来的木头按次序还原。我们逐次筛选，出最短的两木头最先还原，将还原好的木头放在其他具好的木头当中，继续进行还原。依次进行，直至最后只剩下一块木头。

先输入数据，然后用堆排序得到最小堆。然后开始循环每一次循环代表一次切割提取第1和第2个位置的元素，然后两者做和成为新的元素。一直循环直至只剩下一个元素。每次循环时把提取出来的两个元素作和，最后只是队列中只剩下一个元素。

1.4 代码一

我们两个人写了两段代码，两者都能实现问题。这是第一段代码[1]

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
typedef struct
{
    int weight;                    //权值
    int parent, lchild, rchild; //双亲，左子树，右子树，注意这是一个静态链表
} HTNode, *HuffmanTree;            //结点，指针
void Select(HuffmanTree HT, int x, int &s1, int &s2)
{
    int f = 0, g = 0;               //初始化f，g，作用相当于flag
    for (int i = 1; i <= x; i++) //注意满足了一个就会直接跳到下一个循环if
    {
        if (HT[i].parent != 0) //如果说i结点已经被分配了双亲就跳过它
        {
            continue;
        }
        if (!f) //如果f是0，用于初始化s1
        {
            f = 1;
            s1 = i;
        }
        else if (!g) //如果g是0，用于初始化s2
        {
            g = 1;
            s2 = i;
            if (HT[s2].weight < HT[s1].weight) //保证s2的权值大于s1
            {
                swap(s1, s2);
            }
        }
        else if (HT[i].weight < HT[s1].weight) //此时s2和s1最小
        {
            s2 = s1;
            s1 = i;
        }
        else if (HT[i].weight < HT[s2].weight) //此时s2和s1最小
        {
```

[1] latex编译注释存在中文和英文、数字排序混乱，部分混乱可能是漏加了逃逸字串

```
            s2 = i;
        }
    }
    cout << " " << HT[s1].weight << " " << HT[s2].weight << endl;
}
void CreateHuffmanTree(HuffmanTree &HT, int n)
{
    if (n <= 1)
        return;               //n如果小于等于1，无需建立霍夫曼树
    int m = 2 * n - 1;        //m是需要建立的节点数
    HT = new HTNode[m + 1]; //分配m+1个空间，0号单元未使用
    for (int i = 1; i <= m; i++)
    {
        HT[i].parent = 0;
        HT[i].lchild = 0;
        HT[i].rchild = 0; //一开始都初始化为0
    }
    for (int i = 1; i <= n; i++)
    {
        cin >> HT[i].weight; //静态链表，输入木头长度
    }
    for (int i = n + 1; i <= m; i++)
    {
        int s1, s2;
        Select(HT, i - 1, s1, s2); //s1和s2代表权值最小的编号
        HT[s1].parent = i;
        HT[s2].parent = i;
        HT[i].lchild = s1;
        HT[i].rchild = s2;
        HT[i].weight = HT[s1].weight + HT[s2].weight; //第i个结点是s1和s2的双亲，权值为s1和s2的和
    }
}

void readtree(HuffmanTree ht, int n)
{
    for (int i = 2 * n -1; i >=1; i--)
    if(ht[i].lchild && ht[i].rchild){
        printf("切割%d成为%d和%d\n",ht[i].weight, ht[ht[i].lchild].weight, ht[ht[i].rchild].weight);
    }
}

int main()
{
    int n;
    scanf("%d", &n);                    //n是木头数量
    HuffmanTree H = new HTNode[100]; //分配空间
    CreateHuffmanTree(H, n);            //建立霍夫曼树
    int sum = 0;
    for (int i = n + 1; i <= 2 * n - 1; i++)
    {
        sum += H[i].weight; //双亲权值相加得到花费
    }
    readtree(H, n);
    cout << sum << endl;
}

```

Listing 1: 第一段代码

1.5 代码二

```
#include <stdio.h>
#include <string.h>
void paixv(int a[], int i, int length) //最小堆排序
{
int mins = i;
int lch = i * 2;
int rch = i * 2 + 1;
if(lch <= length && a[lch] < a[i]) mins = lch;
if(rch <= length && a[rch] < a[mins]) mins = rch;//确定最小值的位置
if(mins != i)
{
int t = a[i];
a[i] = a[mins];
a[mins] = t;                                    //交换位置
paixv(a, mins, length);                         //递归进行
}
}
void buildHd(int a[], int length)//引入最小堆排序
{
for(int i = length / 2; i > 0; i--){
paixv(a,i,  length);              }
}
int pop_head(int a[], int& length)      //POP
{
int t = a[1];
a[1] = a[length];       //把最后一个位置的放到头上再重新排序data
paixv(a,1, —length);
return t;                   //打印出原第一个位置的data
}
void insert_paixv(int a[],int& length, int data ) //入队并排序
{
length++;
a[length] = data;
int i=length;
while(i > 1 && a[i] < a[i / 2])//上浮
{
int t = a[i];
a[i] = a[i / 2];
a[i / 2] = t;
i = i/2;
}
}//把新元素插入到最后然后只排序这一趟
int a[10000] = {0};
int main(){
int n;
scanf("%d",&n);
int length = n;
for(int i = 1; i < n+1; ++i)
{
scanf(" %d", &a[i]);
```

```
}
buildHd(a, length);//构建队列
int sum = 0;
while(length > 1){
int x = pop_head(a, length);
int y = pop_head(a, length);//队头两个元素是小的，出来POP
insert_paixv(a, length,x+y);
sum += x+y;
}
printf("%d", sum);
}
```

Listing 2: 第二段代码

1.6 运行结果截图

```
PS C:\Users\微软\Documents\course_back_up\c_algorithm\saw_log> cd "c:\Users\微软\Documents\course_back_up\c_algorithm\saw_log\" ; i
($?) { g++ log2.cpp -o log2 } ; if ($?) { .\log2 }
8
4 5 1 2 1 3 1 1
切割18成为8和10
切割10成为5和5
切割8成为4和4
切割5成为2和3
切割4成为2和2
切割2成为1和1
切割2成为1和1
49
```

图 1: 运行结果1

```
PS C:\Users\微软\Documents\course_back_up\c_algorithm\saw_log> cd "c:\Users\微软\Documents\course_back_up\c_algorithm\saw_log\" ; if
($?) { g++ log2.cpp -o log2 } ; if ($?) { .\log2 }
1 1
PS C:\Users\微软\Documents\course_back_up\c_algorithm\saw_log> cd "c:\Users\微软\Documents\course_back_up\c_algorithm\saw_log\" ; if
($?) { g++ log2.cpp -o log2 } ; if ($?) { .\log2 }
1 1
PS C:\Users\微软\Documents\course_back_up\c_algorithm\saw_log>
```

图 2: 运行结果2

1.7 总结体会

看到这道题，首先想到的就是用冒泡排序，然后把最小的两个合并再依次往上，一个一个的合并，其他的木头，这样就形成了一颗如图5所示的二叉树。把所有的非叶子节点求和即可得到最后答案，但是这种考虑忽视了，如图所示6的做法，只适用一些特殊的情况。采用优先队列的做法，可以实现对上图这种情况的处理。.通过做这道题可以对堆排序有更深刻的理解，中间的有些想法是在参考了网上的优先队列堆排序的实现之后才想到，过程有一些抽象的地方，有待巩固以熟练掌握。

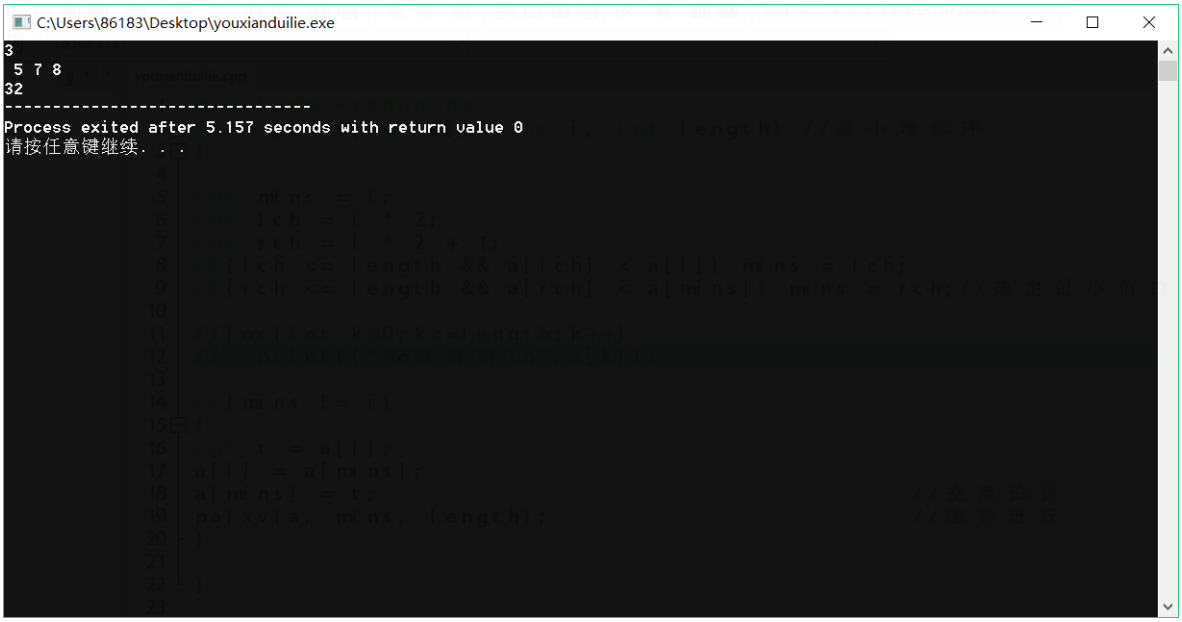

图 3: 运行结果3

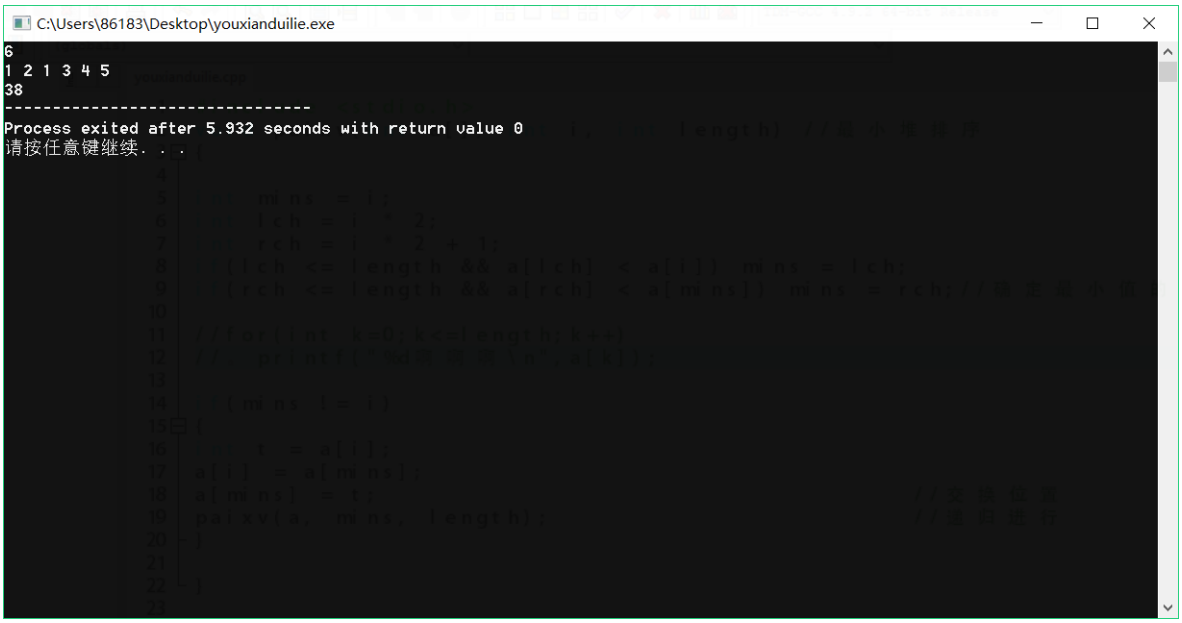

图 4: 运行结果4

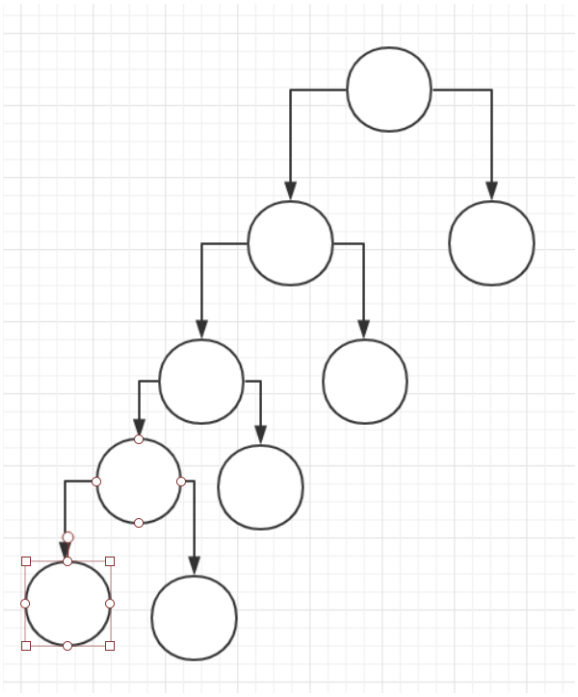

图 5: 第一次想到的，有遗漏

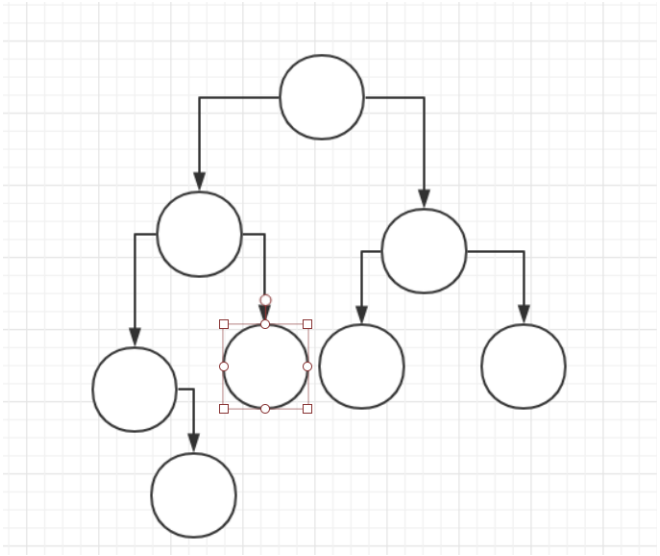

图 6: 最终实现的