Tongji University

Data Structure Course Curriculum Design Document

Task7: Repair the ranch

Author:
嘉杰 李
Jiajie Li

Supervisor:
颖 张
Prof. Ying Zhang

A companion use documentation for a program submitted in fulfillment of the requirements for the Data Structure curriculum design

in the

School of Software Engineering, Tongji

December 27, 2019

“Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism.”

Dave Barry

Contents

Chapter 1

Analysis

1.1 Background Analysis

The farmer wants to repair a fence in the ranch. He measured the fence and found that N pieces of wood are needed. Each piece of wood is an integer Li length unit, so he bought a long piece of wood that can be sawn into N pieces, that is, the wood. The length is the sum of Li.

But the farmer himself does not have a saw, and the reward for asking someone to saw the wood is proportional to the length of the wood. For the sake of simplicity, let us consider the remuneration equal to the length of the sawn wood. For example, to cut a piece of wood of length 20 into three sections of length 8, 7, and 5, the first sawn piece of wood will be sawn into 12 and 8, which costs 20; the second sawn piece of wood will be sawed to length 12 7 and 5 cost 12 and total cost 32 yuan. If the wood is sawn into 15 and 5 for the first time and the wood is sawed into 7 and 8 for the second time, the total cost is 35 (greater than 32)

1.2 Functional Analysis

It is required that the user input the first line to give a positive integer N (N <104), which means that the wood is to be sawn into N pieces. The second line gives N positive integers, indicating the length of each block.

It is required to output a positive integer, which represents the minimum cost of sawing into N blocks.

The analysis function shows that the problem is a Huffman tree problem, so the method of constructing the Huffman tree is used to calculate the minimum cost of sawing into N blocks.

Chapter 2

Design

2.1 Data structure design

From the functional analysis, the method of constructing the Huffman tree is used to solve the problem. To construct a Huffman tree, two extended binary trees with the smallest root node weights are selected as left and right sub-trees to construct a new binary tree. Therefore, the weights of the extended binary trees need to be arranged. The priority queue of the C ++ standard library can be used priority_queue solves the problem. Each time the length with the smallest weight is taken from the priority queue. After the two lengths are added, it takes sum to increase the sum of the two lengths and insert the sum of the two lengths into the priority queue again until there is only one left in the priority queue. The cost sum at this time is the minimum cost.

2.2 Member function design

Vector Class:

```
template <class T>
class Vector {
public:
   Vector(int size=0):theSize(size),theCapacity(size+SPACE_CAPACITY){ data = new
       T[theCapacity];} // 构造函数
   ~Vector(void) { delete[] data;} // 析构函数
   Vector& operator=(Vector& other){
      // 判断是否为自身赋值
      if (this == &other)
         return *this;
      else {
         delete[]data;
         theSize = other.theSize;
         theCapacity = other.theCapacity;
         this->data = new T[theCapacity];

         for (int i = 0; i < theSize; ++i) {
            data[i] = other.data[i];
            }
            return *this;
        }
    } // 重载赋值函数
    Vector(Vector& other) :theSize(0),theCapacity(0),data(NULL){ *this=other; }//
        复制构造函数

    // 重新分配空间大小函数
    void reServe(int newCapacity) {
        if (newCapacity <= theCapacity)
            return;

        T *temp = data;
        data = new T[newCapacity];
        theCapacity=newCapacity;
        //std::cout << newCapacity << std::endl;
        for (int i = 0; i < theSize; ++i)
            data[i] = temp[i];
        delete[] temp;
    }
    // push_back函数
    void push_back(T val) {
        if (theSize == theCapacity)
            reServe(2 * theCapacity + 1);
        data[theSize++] = val;
    }
    bool pop_back(){
        if (!theSize) {
            std::cerr << "Fail to pop back, the vector is empty!\n";
            return false;
        }
        theSize--;
        return true;
    }
    bool Delete(int ind){
        if (ind<0||ind>=theSize){
            std::cerr << "Fail to delete, the ind is out of range!\n";
            return false;
        }
        for (int i=ind;i<theSize-1;++i) data[i]=data[i+1];
        return pop_back();
    }
    T& operator[] (int n) {return data[n];}
    T *data;
    int size() const {return theSize;}
private:
    const int SPACE_CAPACITY = 16;
    int theCapacity;
    int theSize;
};
```

Heap Class:

```
template <typename T>
class Heap{
private:
    Vector<T> data;
public:
    Heap(){};
    Heap(Vector<T> v);
    void adj(int curNode=0);
    void push_heap(T val);
    void push_heap(T val, int pos);
    void ergodic();
    T pop_heap();
    void swap(T &x,T &y){
        T temp;
        temp=x;
        x=y;
        y=temp;
    }
    int size() {return data.size();}
};

template <typename T>
Heap<T>::Heap(Vector<T> v){
    data = v;
    for (int i=size()-1; i>=0; --i) adj(i);
}

template <typename T>
T Heap<T>::pop_heap(){
    T temp=data[0];
    swap(data[0], data[size()-1]);
    data.Delete(size()-1);
    adj();
    return temp;
}

template <typename T>
void Heap<T>::adj(int curNode){
    int left=2*curNode+1;
    int right=2*curNode+2;
    int min=curNode;
    if (left<data.size()) min=data[min]<data[left]?min:left;
    if (right<data.size()) min=data[min]<data[right]?min:right;
    if (min!=curNode) swap(data[min],data[curNode]), adj(min);
}

template <typename T>
void Heap<T>::push_heap(T val){
    data.push_back(val);
    push_heap(val, size()-1);
}

template <typename T>
void Heap<T>::push_heap(T val, int pos){
    if (pos==0) return;
    int father = (pos-1)/2;
    if (data[father]>data[pos]) swap(data[father],data[pos]), push_heap(val,
        father);
}
```

2.3 Algorithm design

First, create an increasing priority queue l, initialize the sum sum to 0, sequentially enter the length entered by the user into queue l, and then take the length with the smallest weight from the priority queue each time. After the two lengths are added, the sum is increased by two The sum of the two lengths is inserted into the priority queue again until there is only one length left in the priority queue, and the cost sum at this time is the minimum cost.

Chapter 3

Implementation

3.1 Main Function

3.1.1 Main core code

```
cout << "Please input the name to establish family: ";
          cin >> name;
          if (geneTree.find(name)== nullptr) {
             cerr << "The name '" << name << "' doesn't exist! try again plz" <<
                 endl;
             break;
          }
          __treenode<string>* father = geneTree.find(name);
          cout << "Please input the number of " << name << "'s offsprings: ";
          cin >> n;
          Vector<__treenode<string> *> temp;
          cout << "Please input the names of " << name << "'s offsprings in order
              : ";
          for (int i=0; i<n; ++i) {
             cin >> name_t;
             __treenode<string> *son = new __treenode<string>(name_t, father);
             temp.push_back(son);
          }
          geneTree.build(name, temp);
          cout << name << "'s first generation offsprings are: ";
          geneTree.outSpring(name);
          cout << endl;
```

3.1.2 Main flowchart

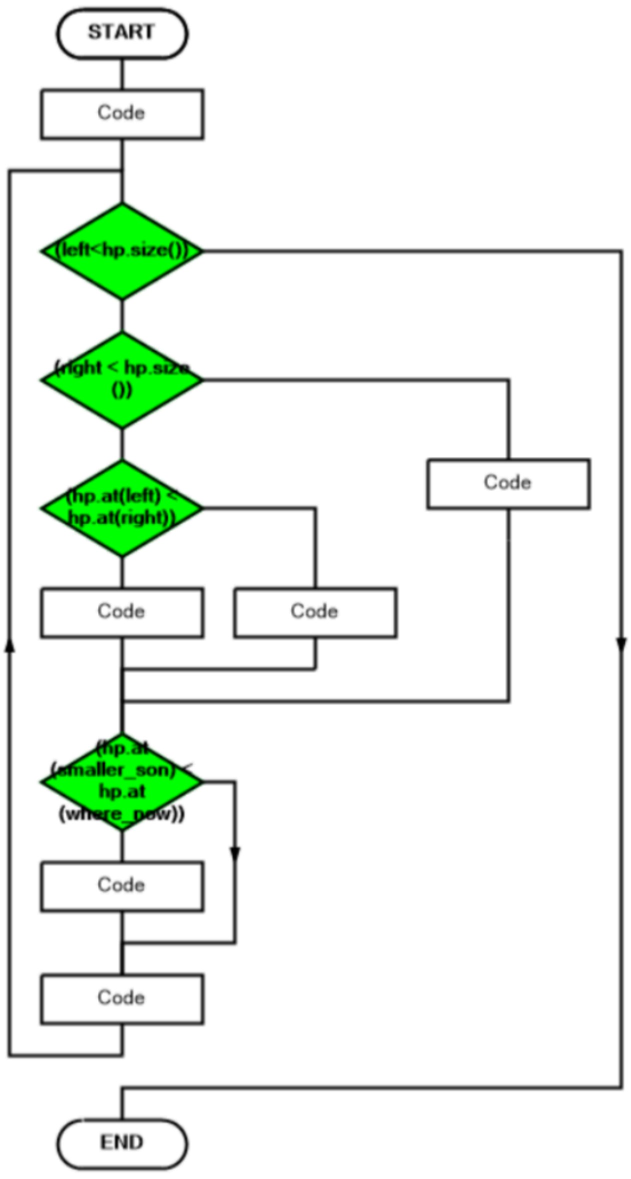

START
Code
(left<hp.size())
(right < hp.size())
(hp.at(left) < hp.at(right))
Code
Code
Code
(hp.at(smaller_son) < hp.at(where_now))
Code
Code
END

Figure 3.1: Main flowchart

Chapter 4

Test

4.1 Function Tests

4.1.1 Normal

Test Case:

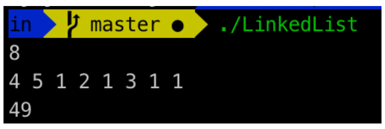
8

4 5 1 2 1 3 1 1

Expected Result:

49

Program Result:



```
in  master ●  ./LinkedList
8
4 5 1 2 1 3 1 1
49
```

Figure 4.1: Screenshots

4.1.2 Slice into two pieces

Test Case:

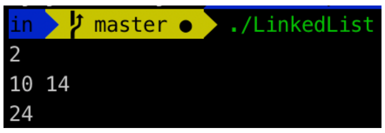
2

10 14

Expected Result:

24

Program Result:



```
in  master ●  ./LinkedList
2
10 14
24
```

Figure 4.2: Screenshots

Appendix A

Frequently Asked Questions

A.1 Why this document looks so sparse?

This document is built using LaTeX and is arranged in book format. There may be blank pages before and after each chapter.

A.2 Why is there header files in this project?

I wrote basic stl header files, such as vector.h, etc. Except for the header files I wrote, the only header files used in all projects are iostream and string.h