**实 验 报 告**

|  |  |
| --- | --- |
| **课程名称：** | **数据结构与算法** |
| **学生姓名：** | **林墨馨** |
| **学生学号：** | **201730682485** |
| **学生专业：** | **软件工程** |
| **开课学期：** | **2018-2019学年第1学期** |

**软件学院**

**2018年9月**

**目 录**

[实验一 线性表的基本操作实现和应用 1](#_Toc471221116)

[实验二 二叉树的建立及相关算法的实现 3](#_Toc471221117)

[实验三 基本排序算法的实现和应用 5](#_Toc471221118)

[实验四 自组织线性表以及哈希表的实现和应用 7](#_Toc471221119)

# 线性表的基本操作实现和应用

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **地 点：** | | B7 楼B座 | | 房 |  |
| **实验日期与时间：** | | 2018年10月13日 上午第1-4节 | | |
| **实验教师：** | 黄敏、齐海涛 | |

**实验内容：**

a.顺序表的实现及操作

① 输入一组整型元素序列，建立线性表的顺序存储结构。

② 实现该线性表的遍历。

③ 在该顺序表中查找某一元素,查找成功显示查找元素，否则显示查找失败。

④ 在该顺序表中删除或插入指定元素。

⑤ 建立两个按值递增有序的顺序表，将他们合并成一个按值递增有序的顺序表。

b.链表的实现及操作

① 建立线性表的链式存储结构。

② 输入两组非空非负的一位整数，使每个数据代表一位数字，整个链表从表头到表尾表示一个非负整数，构建出两个链表（如2 -> 4 -> 3、5 -> 6 -> 4，分别代表342和465）。

③ 求出②中两链表所代表的数字的和，并将数据以上述方式存入链表。

例：Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)

Output: 7 -> 0 -> 8

Explanation: 342 + 465 = 807.

c.给定一个链表1->2->3->4->5和一个小于链表长度的正整数n，要求使用一趟遍历移除链表倒数第n个元素。

d.附加题：

①给定一个链表，确定链表中是否存在环，如果存在，则给出环开始的位置。（不允许修改链表，思考如何使空间复杂度为O(1)）

②给定两个已排序好的数组array1和array2，长度分别为m,n，找出这两组数的中位数。（要求时间复杂度为O(log (m+n))）

例：array1 = [1, 3]

array2 = [2]

中位数则为 2.0

又如

array1 = [1, 2]

array2 = [3, 4]

中位数则为 (2 + 3)/2 = 2.5

**a)题**

//use link list

#include<iostream>

#include<string>

using std::string;

using std::cout;

using std::cin;

using std::endl;

template<typename T>

class LinearTable

{

public:

T \* getElement(int pos)//no check

{

return head[pos];

}

void setElement(const T e, int pos)

{

head[pos] = e;

}

int size()

{

return Size\_;

}

int maxsize()

{

return MaxSize;

}

LinearTable()//set MaxSize = 100 as default

{

MaxSize = 100;

Size\_ = 0;

head = new T[MaxSize];

}

LinearTable(int size)

{

head = new T[size + 10];

//for (int i = 0; i < size; i++)

//{

//cin >> head[i];

//}

Size\_ = size;

MaxSize = size + 10;

}

LinearTable(T \* e,int size)

{

head = new T[size + 10];

for (int i = 0; i < size; i++)

{

head[i] = e[i];

}

Size\_ = size;

MaxSize = size + 10;

}

~LinearTable()

{

delete[] head;

}

void resize(int size\_)//allocate more space but do nothing on data

{

if (size\_ > MaxSize)

{

MaxSize = size\_;

T \* tmp = head;

head = new T[size\_];

for (int i = 0; i < Size\_; i++)

head[i] = tmp[i];

delete[] tmp;

}

}

void traverse()

{

for (int i = 0; i < Size\_; i++)

cout << head[i] << " ";

}

void append(const T & e)//add element on the tail

{

resize(Size\_ + 1);

head[Size\_++] = e;

}

int findElement(const T & e)//if find->return 1 ,else return -1

{

for (int i = 0; i < Size\_; i++)

{

if (e == head[i])return i;

}

return -1;

}

bool deleteElement(int pos)

{

if (pos >= Size\_)return false;

for (int i = pos; i < Size\_ - 1; i++)

{

head[i] = head[i + 1];

}

Size\_--;

return true;

}

bool insertElement(const T e, int pos)

{

if (pos > Size\_)return false;

resize(Size\_ + 1); //allocate more space if necessary

for (int i = pos;i < Size\_; i++)

{

head[i + 1] = head[i];

}

Size\_++;

head[pos] = e;

return true;

}

private:

T \* head;

int Size\_;

int MaxSize;

};

//an simplified shell

int main()

{

int order,pos,tmp;

bool ifBuild = false;

LinearTable<int> \* a = nullptr, \* b = nullptr;

while (true)

{

cout << "---------------------------------------------------------------------------------\n";

cout << "enter order\n";

cout << "1.build a linear table\n";

cout << "2.traverse\n";

cout << "3.search an element and show whether this element is in the linear table\n";

cout << "4.insert an element\n";

cout << "5.delete an element\n";

cout << "6.merge two sequential linear table\n";

cin >> order;

switch (order - 1)

{

case 0:

if (a != nullptr)delete a;

cout << "the size is\n";

cin >> order;

a = new LinearTable<int>(order);

cout << "enter element of the linear table\n";

for (int i = 0; i < order; i++)

{

cin >> tmp;

a->setElement(tmp, i);

}

break;

case 1:

if (a == nullptr)

{

cout << "wrong!\nHas not bulit the linear table\n";

continue;

}

a->traverse();

cout << endl;

break;

case 2:

if (a == nullptr)

{

cout << "wrong!\nHas not bulit the linear table\n";

continue;

}

cout << "enter the element you want to search\n";

cin >> tmp;

pos = a->findElement(tmp);

if (pos == -1)cout << "can not be found\n";

else

{

cout << "has been found\nthis element is in\n" << pos + 1 << endl;

}

break;

case 3:

if (a == nullptr)

{

cout << "wrong!\nHas not bulit the linear table\n";

continue;

}

cout << "enter the place and the element you want to insert\n";

cin >> pos >> tmp;

if (a->insertElement(tmp, pos - 1))cout << "success!\n";

else cout << "failed\n";

break;

case 4:

if (a == nullptr)

{

cout << "wrong!\nHas not bulit the linear table\n";

continue;

}

cout << "enter the place and the element you want to insert\n";

cin >> pos;

if (a->deleteElement(pos - 1))cout << "success!\n";

else cout << "failed\n";

break;

case 5:

if (a == nullptr)

{

cout << "wrong!\nHas not bulit the linear table\n";

continue;

}

cout << "has not built\n";

break;

default:

cout << "you have enter wrong order,please try again\n";

}

}

if (a != nullptr)delete a;

if (b != nullptr)delete b;

}

**b)题**

#include<iostream>

using std::cout;

using std::cin;

using std::endl;

class Node

{

public:

Node(int e)

{

element\_ = e;

//next\_ = n;

}

int element\_;

Node \* next\_;

};

class LinkLine

{

public:

LinkLine()

{

//cin >> length;

int e;

cin >> e;

Node \* tmp = head = new Node(e % 10);

e /= 10;

while(e != 0)

//for (int i = 1; i < length; i++)

{

//cin >> e;

tmp->next\_ = new Node(e % 10);

e /= 10;

tmp = tmp->next\_;

}

};

LinkLine(int e)

{

Node \* tmp = head = new Node(e % 10);

e /= 10;

while (e != 0)

//for (int i = 1; i < length; i++)

{

//cin >> e;

tmp->next\_ = new Node(e % 10);

e /= 10;

tmp = tmp->next\_;

}

};

int getValue()

{

Node \*tmp = head;

int index = 1;

int ans = 0;

while (tmp != nullptr)

{

ans += index \* tmp->element\_;

tmp = tmp->next\_;

index \*= 10;

}

return ans;

}

void show()

{

Node \* tmp = head;

while (tmp != nullptr)

{

cout << tmp->element\_ ;

tmp = tmp->next\_;

if (tmp != nullptr)cout << "->";

}

cout << endl;

}

private:

Node \* head;

int length;

};

int main()

{

while (true)

{

LinkLine a1, a2;

a1.show();

a2.show();

LinkLine ans(a1.getValue() + a2.getValue());

ans.show();

//cout << a1.getValue() + a2.getValue() << endl;

}

return 0;

}

**c)题**

#include<iostream>

using std::cout;

using std::cin;

using std::endl;

class Node

{

public:

Node(int e)

{

element\_ = e;

}

int element\_;

Node \* next\_;

};

class LinkLine

{

public:

LinkLine()//enter the length and element of the link line

{

cout << "please enter the size of the link line\n";

cin >> length;

int e;

cout << "please enter the element of the link line\n";

cin >> e;

Node \* tmp = head = new Node(e);

for (int i = 1; i < length; i++)

{

cin >> e;

tmp->next\_ = new Node(e);

tmp = tmp->next\_;

}

}

int getValue()

{

Node \*tmp = head;

int index = 1;

int ans = 0;

while (tmp != nullptr)

{

ans += index \* tmp->element\_;

tmp = tmp->next\_;

index \*= 10;

}

return ans;

}

void show()

{

Node \* tmp = head;

while (tmp != nullptr)

{

cout << tmp->element\_;

tmp = tmp->next\_;

if (tmp != nullptr)cout << "->";

}

cout << endl;

}

int getLength() { return length; };

bool remove(int pos)

{

Node \* fast = head, \* slow = head,\* slow\_ = head;

for (int i = 0; i < pos; i++)

{

fast = fast->next\_;

}

while (fast != nullptr)

{

fast = fast->next\_;

slow\_ = slow;

slow = slow->next\_;

}

//find the element and delete it

//Node \* ptr = slow;

slow\_->next\_ = slow->next\_;

delete slow;

return true;

//if (pos > length)return false;

/\*Node \* tmp1 = head,\* tmp2 = head;

for (int i = 2; i < pos; i++)

{

tmp2 = tmp2->next\_;

tmp1 = tmp1->next\_;

}

if (pos != 1)

{

tmp1 = tmp1->next\_;

tmp2->next\_ = tmp1->next\_;

}

else

{

tmp1 = head;

head = head->next\_;

}

delete tmp1;

length--;

return true;\*/

};

private:

Node \* head;

int length;

};

int main()

{

int n;

while (true)

{

LinkLine a;

a.show();

cout << "enter n:";

cin >> n;

while (n > a.getLength())//make sure n is smaller than length

{

cout << "false\nenter n again\n";

cin >> n;

}

a.remove(n);

//a.remove(a.getLength() - n + 1);//maybe some problem

a.show();

}

}

**d)1.**

#include<iostream>

#include<time.h>

using std::cin;

using std::cout;

using std::endl;

//maybe with the problem of memory let out.

//without delete

template <typename T>

class Node

{

public:

Node()

{

next\_ = nullptr;

}

Node(T e)

{

element\_ = e;

//next\_ = n;

}

T element\_;

Node \* next\_;

};

//give the head of the link lined and the length of the link line

template <typename T>

bool solve(T \* head)

{

T \* fast = head, \* slowly = head;

//use two pointer.One is faster and the other is slow

//if they meet in the loof, this link line has loop

while (fast != nullptr)

{

fast = fast->next\_;

if (fast == nullptr)break;

fast = fast->next\_;

slowly = slowly->next\_;

if (fast == slowly)return true;

}

return false;

}

template<typename T>

Node<T> \*loop\_link(int size\_)//return the head of a linked line with a loop

{

Node<T> \* current;

Node<T> \* loop;

current = loop = new Node<T>();

for (int i = 0; i < size\_; i++)

{

current->next\_ = new Node<T>();

current = current->next\_;

}

//this is a link line with a loop

current->next\_ = loop;

return loop;

}

template<typename T>

Node<T> \* non\_loop\_link(int size\_)

{

Node<T> \* current;

Node<T> \* non\_loop;

current = non\_loop = new Node<T>();

for (int i = 0; i < size\_; i++)

{

current->next\_ = new Node<T>();

current = current->next\_;

}

return non\_loop;

}

int main()

{

//srand((int)time(0));

//int x;

//Node<int> \* loop, \* non\_loop;

//Node<int> \* current;

//current = loop = new Node<int>(0);

//for (int i = 1; i < rand() % x; i++)

//{

// current->next\_ = new Node<int>(i);

// current = current->next\_;

//}

////this is a link line with a loop

//current->next\_ = loop;

////link line without loop

//current = non\_loop = new Node<int>(10);

//for (int i = 11; i < 20; i++)

//{

// current->next\_ = new Node<int>(i);

// current = current->next\_;

//}

int time = 1000;

while (time--)

{

Node<int> \* loop, \*non\_loop;

loop = loop\_link<int>(rand() % 100);

non\_loop = non\_loop\_link<int>(rand() % 100);

if (solve(loop) == 1 && solve(non\_loop) == 0)continue;

else cout << "wrong";

//test part.

//wrong will be input

//continue to make a non-loop + loop

}

system("pause");

return 0;

}

**2.**

#include<iostream>

#include<vector>

#include<algorithm>

#include<math.h>

using std::cout;

using std::cin;

using std::endl;

using std::min;

//assume that a\_amount is smaller

int find(int \*a, int a\_amount,int \*b,int b\_amount,int pos\_k)

{

if (a\_amount > b\_amount)return (b, b\_amount, a, a\_amount, pos\_k);//make sure a\_amount is smaller

if (pos\_k == 1)return a[0] < b[0] ? a[0] : b[0];//return the smaller

if (a\_amount == 0)return b[pos\_k];//a array is empty ,so number k is b[k]

int tmp\_a = min(pos\_k / 2 - 1,a\_amount - 1);

int tmp\_b = min(pos\_k / 2 - 1, b\_amount - 1);

if (a[tmp\_a] < b[tmp\_b])return find(a + tmp\_a, a\_amount - tmp\_a, b, b\_amount, pos\_k - tmp\_a);

else if (a[tmp\_a] > b[tmp\_b]) return find(a, a\_amount, b + tmp\_b, b\_amount - tmp\_b, pos\_k - tmp\_a);

else return a[tmp\_a];

}

double solve(int a[], int b[], int size\_A, int size\_B)

{

int sum\_ = size\_A + size\_B;

if ((sum\_ % 2) == 0) //sum % 2 == 0

return double(find(a, size\_A, b, size\_B, sum\_ / 2 )+ find(a, size\_A, b, size\_B, (sum\_ / 2) ) + 1 )/ 2.0;

else

return find(a, size\_A, b, size\_B, sum\_ / 2 + 1);

}

int main()

{

int \*a = nullptr, \*b = nullptr;

int size\_A, size\_B;

while (cin>>size\_A>>size\_B)

{

if (a != nullptr)delete[] a;

if (b != nullptr)delete[] b;

a = new int[size\_A];

b = new int[size\_B];

for (int i = 0; i < size\_A; i++)

{

cin >> a[i];

}

for (int i = 0; i < size\_B; i++)

{

cin >> b[i];

}

//input a,b

cout<<"the answer is : "<<solve(a, b, size\_A, size\_B)<<endl;

cout << "enter amount of two array\n";

}

}

**（电子版实验报告内容打印页）**

# 二叉树的建立及相关算法的实现

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **地 点：** | | B7 楼B座 | | 房 |  |
| **实验日期与时间：** | | 2018年10月20日 上午第1-4节 | | |
| **实验教师：** | 黄敏、齐海涛 | |

**实验内容：**

a.编程实现建立一棵二叉树，并对其进行先序、中序和后序遍历。

b.给出二叉树的前序遍历：1 2 4 5 8 9 3 6 7 10及中序遍历：4 2 8 5 9 1 6 3 7 10，请编程实现二叉树的重构，并输出该二叉树的后序遍历。

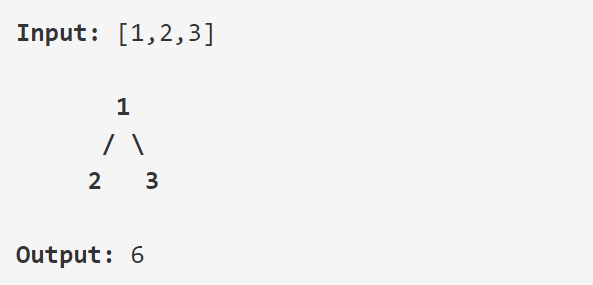
c.使用你的二叉树将给定序列构建成二叉搜索树，实现查找函数。

d.给出一组权值，将它们构建成Huffman树，并输出Huffman编码。

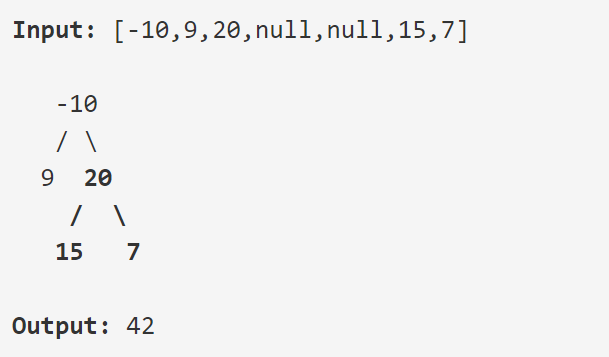
e.附加题：Given a **non-empty** binary tree, find the maximum path sum.

For this problem, a path is defined as any sequence of nodes from some starting node to any node in the tree along the parent-child connections. The path must contain **at least one node** and does not need to go through the root.

**Example 1:**



**Example 2:**



（通过链接<https://leetcode.com/problems/binary-tree-maximum-path-sum/description/>测试你的代码）

a)

#include<iostream>

using std::endl;

using std::cout;

using std::cin;

//need test

//preorder postorder inorder

template<typename T>

class binNode

{

public:

binNode();//root

binNode(const T &);//root

binNode(const T &, binNode<T> \* f);

binNode(binNode \* f);

~binNode();

binNode<T> \* setLeft(const T &);

binNode<T> \* setRight(const T &);

binNode \* LeftChild() { return leftChild; };

binNode \* RightChild() { return rightChild; };

T \* left();

T \* right();

T& element() { return Element; };

bool ifLeaf();

void visit();

void preorder(binNode <T> \*);

void postorder(binNode <T> \*);

void inorder(binNode <T> \*);

binNode \* leftChild, \*rightChild, \*father;

T Element;

int level;//root is level 0

};

template<typename T>

binNode<T>::binNode()

{

leftChild = rightChild = nullptr;

Element = 0;

level = 0;

father = nullptr;

}

template<typename T>

binNode<T>::binNode(const T & e)

{

level = 0;

father = nullptr;

Element = e;

leftChild = rightChild = nullptr;

}

template<typename T>

binNode<T>::binNode(const T & e, binNode<T> \* f)

{

leftChild = rightChild = nullptr;

Element = e;

father = f;

}

template<typename T>

binNode<T>::binNode(binNode \* f)

{

leftChild = rightChild = nullptr;

Element = 0;

father = f;

}

template<typename T>

binNode<T>::~binNode()//delete tree

{

if (leftChild != nullptr)delete leftChild;

if (rightChild != nullptr)delete rightChild;

}

template<typename T>

binNode<T> \* binNode<T>::setLeft(const T & e)

{

if (leftChild == nullptr)

leftChild = new binNode(e, this);

//else leftChild->setLeft(e);

else leftChild->Element = e;

leftChild->level = level + 1;

return leftChild;

}

template<typename T>

binNode<T> \* binNode<T>::setRight(const T & e)

{

if (rightChild == nullptr)

rightChild = new binNode(e, this);

//else rightChild->setRight(e);

else rightChild->Element = e;

rightChild->level = level + 1;

return rightChild;

}

template<typename T>

T \* binNode<T>::left()

{

return leftChild;

}

template<typename T>

T \* binNode<T>::right()

{

return rightChild;

}

template<typename T>

bool binNode<T>::ifLeaf()

{

if (leftChild == nullptr && rightChild == nullptr)

return true;

return false;

}

template<typename T>

void binNode<T>::visit()

{

std::cout << element() << " ";

}

template<typename T>

void binNode<T>::preorder(binNode <T> \* r)

{

if (r == nullptr) return;

r->visit();

preorder(r->leftChild);

preorder(r->rightChild);

}

template<typename T>

void binNode<T>::postorder(binNode <T> \* r)

{

if (r == nullptr) return;

postorder(r->leftChild);

postorder(r->rightChild);

r->visit();

}

template<typename T>

void binNode<T>::inorder(binNode <T> \* r)

{

if (r == nullptr) return;

inorder(r->leftChild);

r->visit();

inorder(r->rightChild);

}

template<typename T>

binNode<T> \* buildTree( binNode<T> \* root)

{

binNode<T> \* right = root->setRight(4);

binNode<T> \* left = root->setLeft(5);

left->setRight(2);

left->setLeft(8);

right = right->setLeft(7);

right->setRight(10);

return root;

}

int main()

{

binNode<int> tree(1);

buildTree<int>(&tree);

cout << "preorder is :";

tree.preorder(&tree);

cout << endl;

cout << "inorder is :";

tree.inorder(&tree);

cout << endl;

cout << "post order is :";

tree.postorder(&tree);

cout << endl;

system("pause");

return 0;

}

b)

#define DEBUG

#include<iostream>

using namespace std;

template<typename T>

class binNode

{

public:

binNode()//root

{

Element = -1;

};

binNode(T e)

{

Element = e;

};

~binNode() {};

void visit()const

{

if (this == nullptr) return;

cout << element() << " ";

};

void preorder(binNode<T> \* r)

{

if (r == nullptr)return;

r->visit();

preorder(r->left());

preorder(r->right());

}

void postorder(binNode <T> \* r)

{

postorder(r->left());

postprder(r->right());

visit(r);

};

binNode \* left()

{

return lc;

};

binNode \* right()

{

return rc;

};

binNode \* setLeft(const T & t)

{

lc = new binNode(t);

return lc;

}

binNode \* setLeft( binNode<T> \* ll)

{

lc = ll;

return lc;

}

binNode \* setRight(const T & t)

{

rc = new binNode(t);

return rc;

}

binNode \* setRight( binNode<T> \* rr)

{

rc = rr;

return rc;

}

T element()const

{

return Element;

}

//private:

binNode \* lc, \*rc;

T Element;

};

template<typename T>

class binTree

{

public:

void buildTree(T \* pre, T \* in, int len)

{

root = buildhelp(root ,pre, in, len, 0, 0, len);

};

void preorder()const

{

root->preorder(root);

}

private:

binNode<T> \* root;

int findPosition(T \* seq,int start,int end,const T & targe)

{

for (int i = start; i < end; i++)

{

if (targe == seq[i])

return i;

}

return -1;

};

binNode<T> \* buildhelp(binNode<T> \* r,T \* pre, T \* in, int len, int pos, int left, int right)//pos is the position of preorder sequence

{

if (right - left <= 1)

{

if (right - left == 1)

{

r = new binNode<T>(in[left]);

return r;

}

else return nullptr;

}

int p = findPosition(in, left, right, pre[pos]);

while (p == -1 && p < len && pos < len - 1)

{

p = findPosition(in, left, right,pre[++pos]);

}

#ifdef DEBUG

cout << "p is:" << p << endl;

#endif

//pos is the position of pre

//p is the position of in

r = new binNode<T> (in[p]);

r->setLeft(buildhelp(r->lc,pre,in,len,pos + 1,left,p));

r->setRight(buildhelp(r->rc,pre,in,len,pos + 1,p + 1,right));

return r;

};

};

int main()

{

int size = 10;

int pre[] = { 1, 2, 4, 5, 8, 9, 3, 6, 7, 10 };

int in[] = { 4, 2, 8, 5, 9, 1, 6, 3, 7, 10 };

binTree<int> tree;

tree.buildTree(pre, in, size);

tree.preorder();

system("pause");

return 0;

}

c)

//need to be test

#include<stdio.h>

template <typename E>

class BinNode

{

public:

virtual ~BinNode() {};

//virtual E & element()const = 0;/////////////////////////////////////////////////

virtual E element()const = 0;

virtual void setElement(const E &) = 0;

//virtual BinNode \* left()const = 0;

//virtual BinNode \* right() const = 0;

//virtual void setRight(BinNode \*) = 0;

virtual bool isLeaf()const = 0;

};

template<typename Key,typename E>

class BSTNode : public BinNode<E>

{

public:

BSTNode()

{

lc = rc = nullptr;

};

BSTNode(Key kk, E e, BSTNode \* l = nullptr, BSTNode \* r = nullptr)

{

k = kk;

it = e;

lc = l;

rc = r;

};

~BSTNode()

{

};

//E & element() const////////////////////////////////

E element() const

{

return it;

};

void setElement(const E & e)

{

it = e;

};

Key & key()

{

return k;

};

void setKey(const Key & K)

{

k = K;

};

inline BSTNode \* left()const

{

return lc;

}

inline BSTNode \* right()const

{

return rc;

}

void setLeft(BinNode<E> \* b)

{

//lc = (BSTNode \*)b;

lc = (BSTNode \*)b;

}

void setRight(BinNode<E> \* b)

{

rc = (BSTNode \*)b;

}

bool isLeaf()const

{

return(lc == nullptr && rc == nullptr);

}

void visit()const

{

cout << k << " " << it << endl;

}

void preorder(BSTNode \* r)const

{

if (r == nullptr)return ;

r->visit();

preorder(r->left());

preorder(r->right());

}

void inorder(BSTNode \* r)const

{

if (r == nullptr)return;

inorder(r->left());

r->visit();

inorder(r->right());

}

void postorder(BSTNode \* r)const

{

if (r == nullptr)return;

postorder(r->left());

postorder(r->right());

r->visit();

}

private:

Key k;

E it;

BSTNode \* lc;

BSTNode \* rc;

};

template<typename Key,typename E>

class BST //:public Dictionary<Key,E>

{

public:

BST()

{

root = nullptr;

}

~BST()

{

clearhelp(root);

}

BST(const Key & k,const E & e)

{

root = new BSTNode<Key, E>(k, e);

}

void insert(const Key & k, const E & e)

{

root = inserthelp(root, k, e);

nodecount++;

}

E remove(const Key & k)

{

E temp = findhelp(root, k);

if (temp != NULL) //???

{

root = removehelp(root, k);

nodecount--;

}

return temp;

}

E removeAny()

{

if (root != nullptr)

{

E temp = root->element();

root = removehelp(root, root->key());

nodecount--;

return temp;

}

else return nullptr;

}

E find(const Key & k)const

{

return findhelp(root, k);

}

int size()

{

return nodecount;

}

//void print()const

//{

// if(root == nullptr)cout<<"The BST tree is empty";

// else printhelp(root,0);

//}

void preorder()const

{

root->preorder(root);

}

void inorder()const

{

root->inorder(root);

}

void postprder()const

{

root->postorder(root);

}

private:

BSTNode<Key, E>\*root;

int nodecount;

void clearhelp(BSTNode<Key, E> \* r)

{

if (r == nullptr)return;

clearhelp(r->left());

clearhelp(r->right());

delete r;

};

BSTNode<Key, E>\* inserthelp(BSTNode<Key, E> \* root\_, const Key & k, const E & it)

{

if (root\_ == nullptr)

{

BSTNode<Key, E> \* temp = new BSTNode<Key, E>(k, it, nullptr, nullptr);

return temp;

//return new BSTNode<Key, E>(k, it, nullptr, nullptr);

}

if (k < root\_->key())

{

root\_->setLeft(inserthelp(root\_->left(), k, it));

}

else root\_->setRight(inserthelp(root\_->right(), k, it));

return root\_;

// do not understand

}

BSTNode<Key, E> \* deletemin(BSTNode<Key, E> \* rt)

{

if (rt->left() == nullptr)return rt->right();

else

{

rt->setLeft(deletemin(rt->left()));

return rt;

}

};

BSTNode<Key, E> \* getmin(BSTNode<Key, E> \* rt)

{

if (rt->left() == nullptr)return rt;

else return getmin(rt->left());

};

BSTNode<Key, E> \* removehelp(BSTNode<Key, E> \* rt,const Key & k)

{

if (rt == nullptr) return nullptr;

else if (k < rt->key())rt->setLeft(removehelp(rt->left(), k));

else if (k > rt->key())rt->setRight(removehelp(rt->right(), k));

else

{

BSTNode<Key, E> \* temp = rt;

if (rt->left() == nullptr)

{

rt = rt->right();

delete temp;

}

else if (rt->right() == nullptr)

{

rt = rt->left();

delete temp;

}

else

{

BSTNode<Key, E> \* temp = getmin(rt->right());

rt->setElement(temp->element());

rt->setKey(temp->key());

rt->setRight(deletemin(rt->right()));

delete temp;

}

}

return rt;

};

E findhelp(BSTNode<Key, E>\* root\_,const Key & k)const

{

if (root\_ == nullptr)return NULL;

if (k < root\_->key())return findhelp(root\_->left(), k);

else if (k > root\_->key())return findhelp(root\_->right(), k);

else return root\_->element();

};

void printhelp(BSTNode<Key, E> \* rt)

{

cout << "preorder is \n";

if (rt == nullptr)return;

cout << rt->element() << " ";

printhelp(rt -> left());

printhelp(rt->right());

};

};

#include<iostream>

using namespace std;

int main()

{

int order, element, key;

BST<int, int> tree;

while (true)

{

cout << "enter order\n";

cout << "1.insert an element(enter key and element)\n";

cout << "2.see the tree\n";

cout << "3.find a certain element\n";

cout << "4.delete an element\n";

cin >> order;

switch (order)

{

case 1:

cout << "enter the key and element:";

cin >> key >> element;

tree.insert(key, element);

break;

case 2:

cout << "preorder:\n";

tree.preorder();

cout << "\ninorder:\n";

tree.inorder();

cout << "\npostorder:\n";

tree.postprder();

cout << endl;

break;

case 3:

cout << "please enter the key to find the element\n";

cin >> key;

tree.find(key);

break;

case 4:

cout << "enter the key of the element\n";

cin >> key;

tree.remove(key);

}

}

}

d)

e)

#include<iostream>

using namespace std;

class TreeNode

{

public:

TreeNode(int ww)

{

val = ww;

}

TreeNode \* left, \*right;

int val;

};

class Solution {

public:

int maxPathSum(TreeNode \* root)

{

int res = INT\_MIN;

solve(root, res);

return res;

}

int solve(TreeNode\* node, int & res)

{

if (node == nullptr) return 0;

int left\_ = solve(node->left, res);

int right\_ = solve(node->right, res);

if (left\_ < 0)left\_ = 0;

if (right\_ < 0)right\_ = 0;

int max\_ = left\_ > right\_ ? left\_ : right\_;//get the max of left\_ right\_ 0

max\_ += node->val;

if (res < (left\_ + right\_ + node->val))res = left\_ + right\_ + node->val;

return max\_;

}

};

int main()

{

TreeNode \* root = new TreeNode(4);

TreeNode \* temp = new TreeNode(11);

root->left = temp;

temp = new TreeNode(7);

root->left->left = temp;

temp = new TreeNode(2);

root->left->right = temp;

temp = new TreeNode(13);

root->right = temp;

Solution s;

cout << "max is " << s.maxPathSum(root) << endl;

system("pause");

return 0;

}

**（电子版实验报告内容打印页）**

# 基本排序算法的实现和应用

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **地 点：** | | B7 楼B座 | | 房 |  |
| **实验日期与时间：** | | 2018年10月27日 上午第1-4节 | | |
| **实验教师：** | 黄敏、齐海涛 | |

**实验内容：**

a.基本要求

请上机分别实现插入排序、起泡排序、选择排序、快速排序和合并排序算法，并比较这五种算法的时间和空间复杂度，说明相关算法的优势和劣势。

b.具体应用

本部分需要完成两题，第一题和第二题均是任选一题实现。

第一题，从下面两题中，任选一题实现：

①　数组中有一个数字出现的次数超过数组长度的一半，请找出这个数字。例如，输入一个长度为9的数组{1，2，3，2，2，2，5，4，2}。由于数字2在数组中出现了5次，超过数组长度的一半，因此输出2。

②　输入n个整数，找出其中最小的k个数。例如输入4、5、1、6、2、7、3、8这8个数字，则最小的4个数字是1、2、3、4。

第二题，从下面两题中，任选一题实现：

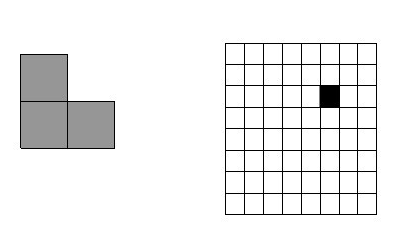
①　我们把只包含因子2、3和5的数称作丑数（Ugly Number）。求按从小到大的顺序的第1500个丑数。例如，6、8都是丑数，但14不是，因为它包含因子7。习惯上我们把1当做第一个丑数。

②　在数组中的两个数字如果前面一个数字大于后面的数字，则这两个数字组成一个逆序对。输入一个数组，求出这个数组中的逆序对的总数。

c.附加题

本部分需要完成以下两题：

①Tromino（更准确地说是“右Trominio”）是一个由棋盘上的三个1x1方块组成的L型骨牌。我们的问题是，如何用Tromino覆盖一个缺少了一个方块（可以在棋盘上的任何位置）的2^n\*2^n棋盘。除了这个缺失的方块，Tromino应该覆盖棋盘上的所有方块，Tromino可以任意转向但不能有重叠。为此问题设计一个分治算法。（PS：具体可以参考下图）



②假设我们有n个直径各不相同的螺钉以及n个相应的螺母。我们一次只能比较一对螺钉和螺母，来判断螺母是大于螺钉、小于螺钉还是正好适合螺钉。然而，我们不能拿两个螺母做比较，也不能拿两个螺钉做比较。我们的问题是要找到每一对匹配的螺钉和螺母。为该问题设计一个算法，它的平均效率必须属于集合O(nlogn)。

**（电子版实验报告内容打印页）**

# 自组织线性表以及哈希表的实现和应用

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **地 点：** | | B7 楼B座 | | 房 |  |
| **实验日期与时间：** | | 2018年11月3日 上午第1-4节 | | |
| **实验教师：** | 黄敏、齐海涛 | |

**实验内容：**

a.自组织线性表的实现和应用

本部分需要完成以下两题：

①假定值A到H存储在一个自组织线性表中，开始按照升序存放。请使用三种组织启发式规则（计数方法、移至前端方法和转置方法），按照下面顺序访问线性表，给出结果线性表和需要的比较总数。

D H H G H E G H G H E C E H G

②对于三种组织启发式规则（计数方法、移至前端方法和转置方法）中的每一种，给出一组记录访问顺序，使得按照这个规则需要的比较数最多。

b.哈希表的实现和应用

本部分需要完成两题，第一题和第二题均是任选一题实现。

第一题，从下面两题中，任选一题实现：

①已知某哈希表的装载因子小于1，哈希函数H（key）为关键字(标识符)的第一个字母在字母表中的序号，处理冲突的方法为线性探测开放定址法。试编写一个按第一个字母的顺序输出哈希表中所有关键字的算法。

②假设哈希表长为m，哈希函数为H(x)，用链地址法处理冲突。试编写输入一组关键字并构造哈希表的算法。

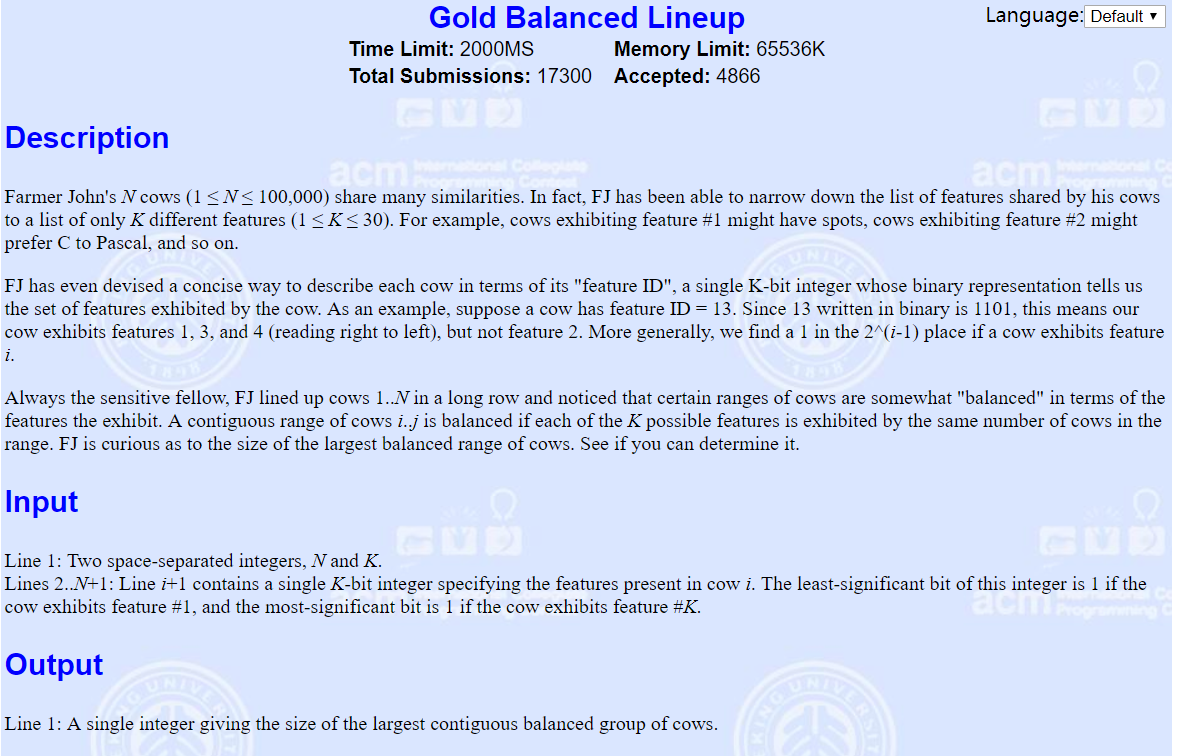
第二题，从下面两题中，任选一题实现：

①在字符串中找出第一个只出现一次的字符。如输入“abaccdeff”，则输出‘b’。

②数组中有一个数出现的次数超过了数组长度的一半，请找出这个数。

**c.附加题**

本部分需要完成下面一道题，题目链接：<http://poj.org/problem?id=3274>



**（电子版实验报告内容打印页）**