**数据结构C++**

**1.vecotr类的operator=**

**template<typename T>**

**class Vector{**

**const Vector&operator=(const Vector &rhs){**

**if(this != &rhs){**

**Delete [] objects;**

**size = rhs.size();**

**capacity = rhs.theCapacity;**

**objects = new Object[capacity()];**

**for(int k = 0 ; k < size() ; k++)**

**objects[k] = rhs.objects[k];**

**}**

**return \*this;**

**}**

**private:**

**int size;**

**int capacity;**

**T \*objects;**

**};**

**2.List类insert方法**

**#include<iostream>**

**using namespace std;**

**template<class T>**

**class List{**

**private:**

**struct Node{**

**T data;**

**Node \*prev;**

**Node \*next;**

**Node(const T &d = T(),Node \*p = NULL,Node \*n = NULL)**

**:data(d),prev(p),next(n){}**

**};**

**public:**

**class const\_iterator{**

**protected:**

**Node \*current;**

**const\_iterator(Node \*p):current(p){}**

**friend class List<T>;**

**};**

**class iterator:public const\_iterator{**

**protected:**

**iterator(Node \*p):const\_iterator(p){}**

**friend class List<T>;**

**};**

**public:**

**iterator insert(iterator itr,const T &x){**

**Node \*p = itr.current;**

**theSize++;**

**return iterator(p->prev=p->prev->next=new Node(x,p->prev,p));**

**}**

**private:**

**int theSize;**

**Node \*head;**

**Node \*tail;**

**};**

**3.AVL中的单旋转**

**include<iostream>**

**using namespace std;**

**template<class T>**

**class AvlTree{**

**public:**

**AvlTree():root(NULL){}**

**AvlTree(const AvlTree &rhs):root(NULL){ \*this = rhs; }**

**void remove(const T&x){ remove(x,root); }**

**private:**

**struct AvlNode{**

**T element;**

**AvlNode\*left;**

**AvlNode\*right;**

**int height;**

**AvlNode(const T&theElement,AvlNode \*lt,AvlNode \*rt,int h = 0)**

**:element(theElement),left(lt),right(rt),height(h){}**

**};**

**AvlNode\*root;**

**void rotateWithLeftChild(AvlNode\*&k2)//单左旋转{**

**AvlNode\*k1=k2->left;**

**k2->left=k1->right;**

**k1->right=k2;**

**k2->height=max(height(k2->left),height(k2->right))+1;**

**k1->height=max(height(k1->left),k2->height)+1;**

**k2=k1;**

**}**

**void rotateWithRightChild(AvlNode\*&k1)//单右旋转{**

**AvlNode\*k2=k1->right;**

**k1->right=k2->left;**

**k2->left=k1;**

**k1->height=max(height(k1->left),height(k1->right))+1;**

**k2->height=max(height(k2->right),k1->height)+1;**

**k1=k2;**

**}**

**int height(AvlNode\*t)const{return t==NULL?-1:t->height;}**

**int max(int lhs,int rhs)const{return lhs>rhs?lhs:rhs;}**

**};**

**4.二叉查找树中的remove函数**

**template<typename T>**

**class BinarySearchTree{**

**public:**

**const T&findMin()const;**

**void remove(const T&x){remove(x,root);}**

**private:**

**struct BinaryNode{**

**T element;**

**BinaryNode \*left;**

**BinaryNode \*right;**

**int num;**

**BinaryNode(const T &e,BinaryNode \*lt,BinaryNode \*rt,int n)**

**:element(e),left(lt),right(rt),num(n){}**

**};**

**BinaryNode\*root;**

**void remove(const T&x,BinaryNode \*&t)const{**

**if(t == NULL)**

**return;**

**if(x<t->element) remove(x,t->left);**

**else if(t->element<x) remove(x,t->right);**

**else if(t->left!=NULL&&t->right!=NULL){**

**t->element=findMin(t->right)->element;**

**remove(t->element,t->right);**

**}**

**else{**

**BinaryNode\*oldNode=t;**

**t=(t->left!=NULL)?t->left:t->right;**

**delete oldNode;**

**}**

**}**

**BinaryNode\*findMin(BinaryNode\*t)const{**

**if(t==NULL)**

**return NULL;**

**if(t->left==NULL)**

**return t;**

**return findMin(t->left);**

**}**

**}**

5.对于分离链接表，写出其contains()方法。

#include<list>

#include<vector>

using namespace std;

template<class T>

class HashTable{

public:

bool contains(const T &x) const{

const list<T>&whichList = theLists[myhash(x)];

return find(whichList.begin(),whichList.end(),x)!=whichList.end();

}

private:

vector<list<T>>theLists;

int myhash(const T&x)const{

int hashVal=hash(x);

hashVal%=theLists.size();

if(hashVal<0)

hashVal+=theLists.size();

return hashVal;

}

};

**6.写出二叉堆中的percolateDown()方法。**

**#include<vector>**

**using namespace std;**

**template<class T>**

**class BinaryHeap{**

**private:**

**int currentSize;**

**vector<T>array;**

**void percolateDown(int hole){**

**int child;**

**T tmp=array[hole];**

**for(;hole\*2<=currentSize;hole=child){**

**child=hole\*2;**

**if(child != currentSize && array[child + 1] < array[child]) child++;**

**if(array[child] < tmp)**

**array[hole] = array[child];**

**else break;**

**}**

**array[hole]=tmp;**

**}**

**};**

**7.插入排序**

**template<typename T>**

**void insertionSort(vector<T>&a){**

**int j;**

**for(int p=1;p<a.size();p++){**

**T tmp = a[p]; //保留第p个位置的数**

**//int j;//放到外层比较好**

**//下面的for循环就是对每个阶段,将从0到p-1个位置的元素**

**//如果满足条件tmp<a[j-1],则右移一位.**

**//如果前面的数小于等于tmp,则不动.**

**for(j = p ;j > 0 && tmp < a[j-1] ; j--)**

**a[j]=a[j-1];**

**a[j]=tmp;//最后将tmp保留**

**}**

**}**

**8.谢尔排序**

**template<typename T>**

**void shellsort(vector<T>&a){**

**for(int gap = a.size() / 2 ; gap > 0 ; gap /= 2)**

**for(int i = gap ; i < a.size() ; i++){**

**T tmp = a[i];**

**int j=i;**

**for( ; j >= gap && tmp < a[j-gap] ; j -= gap )**

**a[j]=a[j-gap];**

**a[j]=tmp;**

**}**

**}**

**else if(thisSum<0) thisSum=0;**

**}**

**return maxSum;**

**}**

9.写出左式堆的合并方法merge1()

#include<iostream>

using namespace std;

template<class T>

class LeftistHeap{

private:

struct LeftistNode{

T element;

LeftistNode\*left;

LeftistNode\*right;

int npl;

LeftistNode(const T&theElement,LeftistNode\*lt=NULL,LeftistNode\*rt=NULL,int np=0)

:element(theElement),left(lt),right(rt),npl(np){}

};

LeftistNode\*merge(LeftistNode\*h1,LeftistNode\*h2){

if(h1==NULL) return h2;

if(h2==NULL) return h1;

if(h1->element<h2->element) return merge1(h1,h2);

else return merge1(h2,h1);

}

LeftistNode\*merge1(LeftistNode\*h1,LeftistNode\*h2){

if(h1->left==NULL) h1->left=h2;

else{

h1->right=merge(h1->right,h2);

if(h1->left->npl<h1->right->npl)

swapChildren(h1);

h1->npl=h1->right->npl+1;

}

return h1;

}

void swapChildren(LeftistNode\*&h1){

LeftistNode\*h2;

h2=h1->left;

h1->left=h1->right;

h1->right=h2;

}

};