山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201700130009 | 姓名： 张愈博 | | 班级： 计科17.3 |
| 实验题目：散列表 | | | |
| 实验学时：2 | | 实验日期： 2018.11.15 | |
| 实验目的：  1、掌握散列表结构的定义和实现。  2、掌握散列表结构的应用。 | | | |
| 软件环境：  Dev C++ 5.11  Windows 10 | | | |
| 1. 实验内容（题目内容，输入要求，输出要求）   （1）用线性开型寻址和链表散列解决溢出，创建散列表类；  （2）列表设计实现一个字典，假设关键字为整数且 D 为 961，在字典中插入随机产生的 500 个不同的整数，实现字典的建立和搜索操作。\*实现字典的删除。   1. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）   （1）线性探查方法：  首先定义一个pair数对类，内含关键字（first）和内容（second）。然后在散列类中成员变量有一个指向指针数组的pair类型指针，hash函数（作为一个类单独实现）,数对个数dSize，散列函数除数divisor。成员函数方法有判断是否空、计算大小、搜索、插入、删除。值得一提的是本方法的删除操作：输入一个关键字为theKey的数对，在散列中搜索到它的位置pos（若找不到就什么也不做），下面从pos开始遍历整个数组并移动key%divisor与theKey相同元素的位置直到回到pos。本算法复杂度Θ(n)。  （2）链表散列：  首先需要定义一个sortedChain类作为桶中的元素链，其节点为PairNode类，内含一个pair数对元素和next指针。成员函数方法有判断是否空、计算大小、搜索、插入、删除、输出等，它们主要利用sortedChain的成员方法完成。这里的关键的是sortedChain的有序性，它可以显著减少链表散列的搜索时间。  （3）实现字典：  有了前两题的基础，本题就十分容易了。只需生成随机数并且用他们初始化散列，并根据要求调用相应的成员函数即可。   1. 测试结果（测试输入，测试输出，结果分析）     上图是线性开型散列表的建立、搜索和删除过程。    上图是链式散列表的建立、搜索和删除过程。   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   1.  在对sortedChain进行insert操作时发生了这样的问题。经查错，发现在insert函数内部new了一个PairNode，由于没有指定参数需要调用其内成员变量pair的无参构造函数，而相应的Pair类没有无参的构造函数，因此出错。  2.  在对hashChain进行insert操作时又出现一个没有找到对应函数的问题，经反复试错，发现只要把insert函数的参数类型Pair<const K,E>改为Pair<K,E>就不会报错。说明模板类似乎并不能完成泛型K在使用中int转const int的默认类型转换。这是与一般变量传参很不同的一个的地方。  3.在调试hashTable的erase函数出现了断点，原因是终止条件出错。把跳出循环条件改为(divisor+b)%divisor即可解决这个问题。     1. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）   Main.cpp  #include<iostream>  #include <time.h>  #include<stdlib.h>  #include"sortedChain.h"  #include"pair.h"  #include"hashTable.h"  #include"hashChain.h"  using namespace std;  int main()  {  //先生成500个随机数然后初始化哈希表  srand((unsigned)time(0));  Pair<int,int> \*P\_array=new Pair<int,int> [500];  for(int i=0;i<500;i++)  {  P\_array[i].first=(rand()%1001);  P\_array[i].second=i;  //cout<<P\_array[i]<<endl;  }    /\* hashChain<int,int> C(961); //499是不大于哈希表容量500的最大素数，详情见书本P254  for(int i=0;i<500;i++)  {  C.insert( P\_array[i] );  }  cout<<C;  int n; cout<<"search:"; cin>>n;  Pair<int,int> \*p=C.find(n);  if(p!=NULL)  cout<<"The element is:"<<(\*p)<<endl;  else cout<<"NULL!"<<endl;    cout<<"erase:"; cin>>n;  C.erase(n);  p=C.find(n);  if(p!=NULL)  {  cout<<"erase failed. "<<(\*p)<<endl;  }  else cout<<"erase succeed."<<endl;\*/    /\* 以上是对hashChain的调试 \*/  hashTable<int,int> B(961);  for(int i=0;i<500;i++)  {  B.insert( P\_array[i] );  }  cout<<B<<endl;  int n; cout<<"search:"; cin>>n;  Pair<int,int> \*p=B.find(n);  cout<<"The element is:"<<(\*p)<<endl;  cout<<"delete:"; cin>>n;  B.erase(n);    return 0;  }  hashTable.h  #include<iostream>  #include"hash.h"  #include"pair.h"  using namespace std;  template<class K,class E>  class hashTable //使用线性探查方法  {  public:  hashTable(int theDivisor)  {  divisor=theDivisor;  dSize=0;  //这里的桶数b和除数d相同,把每个桶初始化为Null  table = new Pair<K,E>\* [divisor];  for(int i=0;i<divisor;i++)  table[i]=NULL;  }  ~hashTable()  {  for(int i=0;i<divisor;i++)  {  if(table[i]!=NULL)  delete [] table[i];  }  }  bool empty() const {return dSize == 0;}  int size() const {return dSize;}  int search(const K& theKey) const  {//搜索theKey返桶的序号  //如果匹配的数对存在，返回它的位置，否则，如果散列表不满，  //则返回关键字为theKey的数对可以插入的位置  int i=(int)hash(theKey)%divisor; //起始桶  int j=i;  do  {  if(table[j]==NULL||table[j]->first==theKey) //起始桶为空或者找到已被填满的桶  return j;  else  j=(j+1)%divisor; //下一个桶  }while(i!=j); //跳出则已经遍历一圈    return j; //表满  }  Pair<K,E>\* find(const K& theKey) const  {//返回匹配数对的指针，若不存在则返回Null  int b=search(theKey);  if(table[b]==NULL||table[b]->first!=theKey)  {//即没有匹配  return NULL;  }  return table[b]; //找到匹配  }  void insert(const Pair<K,E>& thePair)  {//把数对thePair插入字典，若存在关键字相同数对，则覆盖  //表满则禁止插入  int b=search(thePair.first);  if(table[b]==NULL)  {//起始桶为空  table[b]=new Pair<K,E> (thePair); //resembles new int (5);  dSize++;  }  else  {//不为空的情况下  if(table[b]->first==thePair.first)  {//有重复的数对  table[b]->second=thePair.second;  }  if(table[b]->first!=thePair.first)  {//表满  cerr<<"表满，无法插入。"<<endl;  cout<<"table["<<b<<"]->first="<<table[b]->first;  cout<<"\tthePair.first="<<thePair.first<<endl;  }  }  //cout<<b<<endl;  //cout<<table[b]->first<<"\t"<<table[b]->second;  }  void erase(const K& theKey)  {//删除关键字为theKey的数对，如果没有就什么也不做  //需要把它后面的某些元素向前移动，（直到回到初始桶或者遇到空）  int b=search(theKey); //初始桶  if(table[b]==NULL||table[b]->first!=theKey)  {//数对不存在or表满  cout<<"delete failed."<<endl;  return ;  }  int pre=b,post; cout<<"pre is:"<<pre<<endl;  for(int i=b;i<=(divisor+b)%divisor;i++)  {//移动起始桶后的key%divisor与theKey相同的元素，直到回到初始桶  if( table[i]!=NULL && (table[i]->first) % divisor==theKey%divisor)  {  post=i;  table[pre]=table[post];  pre=post;  }  }  //delete table[post];  table[post]=NULL;  dSize--;  cout<<"erase succeed."<<endl;  //写的比较粗暴，复杂度Θ(n)  }  friend ostream& operator<<(ostream& out,const hashTable<K,E> &A)  {//怎么输出...  for(int i=0;i<A.divisor;i++)  {  if(A.table[i]==NULL)  out<<"NO."<<i<<" bucket="<<"NULL"<<endl;  else  out<<"NO."<<i<<" bucket=("<<A.table[i]->first<<","<<A.table[i]->second<<")"<<endl;  }  return out;  }  private:  Pair<K,E>\*\* table; //散列表  hash<K> hash; //把类型K映射到一个非负整数  int dSize; //字典中数对个数  int divisor; //散列函数除数  };  pair.h  #pragma once  #include<iostream>  using namespace std;  //模板类作为模板类的成员变量好像就有问题？  template<class K,class E>  class Pair  {  public:  K first;  E second;  Pair() { }  Pair(K fir,E sec)  {  first=fir; second=sec;  }  Pair(const Pair<K,E> &x)  {  first=x.first; second=x.second;  }  void operator=(Pair<K,E> &x)  {  first=x.first; second=x.second;  }  friend ostream& operator<<(ostream& out,Pair<K,E> Pair)  {  out<<Pair.first<<" "<<Pair.second;  return out;  }  };  template<class K,class E>  class PairNode  {  public:  Pair<K,E> element;  PairNode<K,E>\* next;  PairNode(Pair<K,E> &thePair,PairNode<K,E> \*p)  {  element=thePair;  next=p;  }  friend ostream& operator<<(ostream& out,PairNode<K,E> Pair)  {  out<<"Key:"<<Pair.element.first<<"\tValue:"<<Pair.element.second;  return out;  }  };  hashChain.h  #pragma once  #include <iostream>  #include "hash.h"  #include "sortedChain.h"  using namespace std;  template<class K,class E>  class hashChain  {//使用链表散列  public:  hashChain(int div)  {  divisor=div;  table=new sortedChain<K,E> [divisor];  }  ~hashChain()  {  delete [] table;  }  bool empty()  {  for(int i=0;i<divisor;i++)  {  if(table[i].size()!=0)  return false;  }  return true;  }  int size()  {  int sum=0;  for(int i=0;i<divisor;i++)  {  sum+=table[i].size();  }  return sum;  }  Pair<K,E>\* find(const K& theKey) const  {//返回匹配数对的指针，若不存在则返回NULL  int pos= hash(theKey) %divisor;  return table[pos].find(theKey);  }  void insert(Pair<K,E>& thePair)  {//首先根据他的key计算出pos，然后直接调用sortedChain的insert函数即可。  int pos=hash(thePair.first)%divisor; //cout<<"pos="<<pos<<' ';  table[pos].insert(thePair);  }  void erase(K &theKey)  {  int pos=hash(theKey);  table[pos].erase(theKey);  }  friend ostream& operator<<(ostream& out,hashChain<K,E> &A)  {  for(int i=0;i<A.divisor;i++)  {  if(A.table[i].size()==0) out<<"NO."<<i<<" bucket:NULL"<<endl;  else  {  out<<"NO."<<i<<" bucket:";  A.table[i].output();  out<<endl;  }  }  return out;  }  private:  sortedChain<K,E> \*table;  int divisor;  hash<K> hash; //把类型K映射到一个非负整数    };  sortedChain.h  #pragma once  #include<iostream>  #include"pair.h"  #include "hash.h"  using namespace std;  template<class K,class E>  class sortedChain  {//有序的关键在于insert函数  public:  sortedChain()  {  firstNode = NULL;  dSize = 0;  }  ~sortedChain()  {  // PairNode<K,E> \*p1=firstNode->next,\*p2=firstNode; //这里firtstnode根本没有next!  while(firstNode!=NULL)  {//还是这样用吧...  PairNode<K,E>\* nextNode=firstNode->next;  delete firstNode;  firstNode=nextNode;  }  }  bool empty() const {return dSize == 0;}  int size() const {return dSize;}  Pair<K, E>\* find(const K &Key) const  {//返回匹配数对的指针，若不存在则返回NULL  PairNode<K,E>\* curNode=firstNode;  while(curNode!=NULL&&curNode->element.first!=Key)  {  curNode=curNode->next;  }  //判断是否匹配  if(curNode!=NULL&&curNode->element.first==Key)  return &(curNode->element);  else  return NULL;  }  void erase(K &Key)  {  PairNode<K,E> \*p=firstNode,\*tp=NULL; //tp在p后  while(p!=NULL&& Key < p->element.first)  {  tp=p;  p=p->next;  }  if(p==NULL||Key!=p->element.first)  {//说明已走完或者无匹配  cout<<"no match,erase failed."<<endl;  }  else  {  if(p==firstNode)  firstNode=NULL;  else  tp->next=p->next;  delete p;  dSize--;  }  }  void insert(Pair<K,E> &thePair)  {//向字典中插入thePair，若thePair的key已经存在，则覆盖  PairNode<K,E> \*p=firstNode;  PairNode<K,E> \*tp=NULL; //tp跟在p后  while(p!=NULL&& thePair.first < p->element.first)  {  tp=p;  p=p->next;  }  if(p!=NULL&&thePair.first==p->element.first)  {  p->element.second=thePair.second; //覆盖  return;  }  else  {//加入一个新节点，在合适的位置  PairNode<K,E> \*k=new PairNode<K,E>(thePair,p);  if(tp==NULL)  firstNode=k;  else  tp->next=k; /\* tp...k...p \*/  // cout<<(\*k)<<endl;  dSize++;  }  }  void output() const  {  for (PairNode<K,E>\* curNode = firstNode;curNode != NULL;curNode = curNode->next)  {  cout<<(\*curNode)<<'|';  }  }  protected:  PairNode<K,E>\* firstNode; // 指向首节点  int dSize; // 节点数量  };  hash.h  #pragma once  #include <iostream>  #include <string>  using namespace std;  template <class K> class hash;  template<>  class hash<string>  {//从一个string类型转换为一个size\_t类型的非负整数  public:  size\_t operator()(const string theKey) const  {  unsigned long hashValue = 0;  int length = (int) theKey.length();  for (int i = 0; i < length; i++)  hashValue = 5 \* hashValue + theKey.at(i);    return size\_t(hashValue);  }  };  template<>  class hash<int>  {//从int转size\_t  public:  size\_t operator()(const int theKey) const  {return size\_t(theKey);}  }; | | | |