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：201705130113 | 姓名： 黄瑞哲 | | 班级：计科17.3 |
| 实验题目：散列表 | | | |
| 实验学时：2 | | 实验日期： 2018.11.22 | |
| 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：  Visual Studio Community 2017 | | | |
| 1. 实验内容（题目内容，输入要求，输出要求） 2. 分别使用线性开型寻址和链表散列解决溢出，创建散列表类 3. 使用散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 4. 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） 5. Hash方法为 string类型映射成128进制的数对应的十进制，128进制下的每一位对应ascii码。其余整型int、long long、short等全部转换为size\_t类型的非负整数 6. HashTable为线性开型的散列表，容器是长度为D的数组。Insert方法：先求出插入数对key值对应的桶，尝试放在这个桶中，如果桶为空则结束否则向前寻找空着的桶并插入，如果循环一边没有找到空桶则说明表满，插入失败，抛出异常。Find方法：同insert先求出key对应的桶并不断向前探查直至遇到空桶，在这期间如果找到key则查找成功返回结果，否则查找失败返回NULL。Erase方法：先找到key值对应的桶的位置并清空这个桶。假定这个桶为i，不断向前查找并将下一个非空桶标记为j，空桶则结束。如果i在j的左边，说明还没有循环一圈，此时如果桶j里面放的元素应该放在i到j之间的桶中，说明这个元素不需要移动，否则将这个元素移动到桶i中并将i设为j；如果j在i的左边，说明已经饶了一圈从0的方向开始，此时如果桶j中的元素如果应该放在j到i之间的桶中，则将其移动到桶i中并将i设为j。 7. hashChain为链表散列，其中有长度为D的数组，数组中的每个元素都是一个链表描述的线性表。始终保持每个链表中的key是递增的。Insert方法：先求出key对应的桶的位置，并将数对插入到这个桶对应的链表中。Find方法：找到对应的桶，遍历这个链表，如果找到则返回否则查找失败。Erase方法：找到对应的桶，遍历这个链表，如果找到则删除。 8. 测试结果（测试输入，测试输出，结果分析）   链式散列  线性开型     1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   线性开型的删除操作经过了onlineJudge的评测，算法正确。删除操作的第二种情况下可以理解为和第一种情况是一样的，因为这个数组是可以循环的，可以将这个数组扩大为两倍后j仍然在i的左边，如果对应的桶在i到j之间则仍然是不可以移动的。   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）   /\*dictionary.h\*/  #ifndef dictionary\_  #define dictionary\_  using namespace std;  template<typename K, typename E>  class dictionary {  public:  virtual ~dictionary() {};  virtual bool empty() const = 0;  virtual int size() const = 0;  virtual pair<const K, E>\* find(const K&) const = 0;  virtual void erase(const K&) = 0;  virtual void insert(const K&, const E&) = 0;  };  #endif //dictionary\_  /\*hash.h\*/  #ifndef hash\_  #define hash\_  #include <string>  #include <iostream>  using namespace std;  template<typename T> class Hash;  template<>  class Hash<string> {  public:  size\_t operator()(const string key) const {  unsigned long long value = 0;  int n = key.length();  for (int i = 0; i < n; ++i)  value = value \* 128 + key.at(i);  return size\_t(value);  }  };  template<>  class Hash<int> {  public:  size\_t operator() (const int key) const {  return size\_t(key);  }  };  template<>  class Hash<short> {  public:  size\_t operator() (const short key) const {  return size\_t(key);  }  };  template<>  class Hash<long> {  public:  size\_t operator() (const long key) const {  return size\_t(key);  }  };  template<>  class Hash<long long> {  public:  size\_t operator() (const long long key) const {  return size\_t(key);  }  };  #endif //hash\_  /\*hashChain.h\*/  #ifndef hashChain\_  #define hashChain\_  #include "hash.h"  #include "dictionary.h"  #include <linearList/chain.h>  #include <utility>  #include <iostream>  #include <stdexcept>  using namespace std;  template<typename K, typename E>  class hashChain : public dictionary<K, E> {  public:  hashChain(int \_divisor = 11);  ~hashChain() { delete[] table; };  bool empty() const { return dsize == 0; }  int size() const { return dsize; }  pair<const K, E>\* find(const K&) const;  void erase(const K&);  void insert(const K&, const E&);  void output(ostream&) const;  protected:  chain<pair<const K, E> >\* table;  Hash<K> hash;  int dsize;  int divisor;  };  template<typename K, typename E>  hashChain<K, E>::hashChain(int \_divisor) {  if (\_divisor < 1) throw logic\_error("the divisor must greater than 0.");  divisor = \_divisor;  dsize = 0;  table = new chain<pair<const K, E> >[divisor];  }  template<typename K, typename E>  pair<const K, E>\* hashChain<K, E>::find(const K& key) const {  int index = hash(key) % divisor;  for (auto it = table[index].begin(); it != table[index].end(); ++it)  if (it->first == key)  return &\*it;  return nullptr;  }  template<typename K, typename E>  void hashChain<K, E>::erase(const K& key) {  int index = hash(key) % divisor;  int j = 0;  for (auto it = table[index].begin(); it != table[index].end(); ++it)  {  if (it->first == key) {  table[index].erase(j);  dsize--;  break;  }  j++;  }  }  template<typename K, typename E>  void hashChain<K, E>::insert(const K& key, const E& value) {  int index = hash(key) % divisor;  int j = 0;  for (auto it = table[index].begin(); it != table[index].end(); ++it) {  if (it->first == key) {  it->second = value;  return;  }  if (it->first > key) {  break;  }  j++;  }  dsize++;  table[index].insert(j, make\_pair(key, value));  }  template<typename K, typename E>  void hashChain<K, E>::output(ostream& out) const {  for (int i = 0; i < divisor; ++i) {  //cout << i << ": " << endl;  out << "NO." << i << " bucket:";  if (table[i].empty()) out << "NULL";  else  for (auto it = table[i].begin(); it != table[i].end(); ++it)  out << it->first << ' ' << it->second << " | ";  out << endl;  }  }  #endif //hashChain\_  /\*hashTable.h\*/  #ifndef hashTable\_  #define hashTable\_  #include "dictionary.h"  #include "hash.h"  #include <stdexcept>  template<typename K, typename E>  class hashTable :public dictionary<K, E> {  public:  hashTable(int = 10);  ~hashTable() {  for (int i = 0; i < divisor; ++i) delete table[i];  delete[] table;  }  bool empty() const { return dsize == 0; };  int size() const { return dsize; };  pair<const K, E>\* find(const K&) const;  void erase(const K&);  void insert(const K&, const E&);  void output(ostream&) const;  protected:  Hash<K> hash;  pair<const K, E>\*\* table;  int divisor;  int dsize;  };  template<typename K, typename E>  hashTable<K, E>::hashTable(int \_divisor) {  if (\_divisor < 1) throw logic\_error("the divisor must greater than 0.");  divisor = \_divisor;  dsize = 0;  table = new pair<const K, E>\*[divisor];  for (int i = 0; i < divisor; ++i) table[i] = nullptr;  }  template<typename K, typename E>  pair<const K, E>\* hashTable<K, E>::find(const K& key) const {  int o = hash(key) % divisor;  int i = o;  while (true) {  if (table[i] == nullptr) return nullptr;  else if (table[i]->first == key) return table[i];  i = (i + 1) % divisor;  if (i == o) break;  }  return nullptr;  }  template<typename K, typename E>  void hashTable<K, E>::erase(const K& key) {  int o = hash(key) % divisor;  int i = o;  bool find = false;  while (true) {  if (table[i] == nullptr) break;  else if (table[i]->first == key) {  find = true;  delete table[i];  table[i] = nullptr;  dsize--;  break;  }  i = (i + 1) % divisor;  if (i == o) break;  }  if (find) {  o = i;  int Next = (i + 1) % divisor;  while (Next != o && table[Next] != nullptr) {  int ash = hash(table[Next]->first) % divisor;  if (Next > i) {  if (ash > Next || ash <= i)  {  pair<const K, E>\* tmp = table[Next];  table[Next] = table[i];  table[i] = tmp;  i = Next;  }  }  else {  if (ash > Next && ash <= i)  {  pair<const K, E>\* tmp = table[Next];  table[Next] = table[i];  table[i] = tmp;  i = Next;  }  }  Next = (Next + 1) % divisor;  }  }  }  template<typename K, typename E>  void hashTable<K, E>::insert(const K& key, const E& value) {  int o = hash(key) % divisor;  int i = o;  while (true) {  if (table[i] == nullptr) {  table[i] = new pair<const K, E>(key, value);  dsize++;  return;  }  else if (table[i]->first == key) {  table[i]->second = value;  return;  }  i = (i + 1) % divisor;  if (i == o) throw overflow\_error("dicitionary is full");  }  }  template<typename K, typename E>  void hashTable<K, E>::output(ostream& out) const {  for (int i = 0; i < divisor; ++i) {  out << "NO." << i << " bucket:";  if (table[i] == nullptr) out << "NULL" << endl;  else out << table[i]->first << ' ' << table[i]->second << endl;  }  }  #endif //hashTable\_  /\*pch.h\*/  #ifndef PCH\_H  #define PCH\_H  // TODO: 添加要在此处预编译的标头  #include <hash/hashChain.h>  #include <hash/hashTable.h>  #include <ctime>  #endif //PCH\_H  /\*exe8.cpp\*/  // exe8.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。  //  #include "pch.h"  #include <iostream>  int main()  {  int rmd[500];  hashChain<int, int> h(961);  srand(time(0));  for (int i = 0; i < 500; ++i) {  rmd[i] = rand();  for (int j = 0; j < i; ++j) {  if (rmd[j] == rmd[i]) {  i--;  break;  }  }  }  for (int i = 0; i < 500; ++i) {  h.insert(rmd[i], i);  }  h.output(cout);  int key;  while (true) {  cout << "search:";  cin >> key;  if (key == -1) break;  auto kv = h.find(key);  if (kv == nullptr) cout << "It is NULL" << endl;  else cout << "The element is:" << kv->first << ' ' << kv->second << endl;  }  return 0;  } | | | |