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：201900161140 | 姓名： 张文浩 | | 班级： 19级人工智能 |
| 实验题目：散列表（第二题） | | | |
| 实验学时：2 | | 实验日期： 10.1 | |
| 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。 | | | |
| 软件开发工具：  Vscode | | | |
| 1. 实验内容   **题目描述**：  给定散列函数的除数D和操作数m，输出每次操作后的状态。  有以下三种操作：  插入x，若散列表已存在x，输出"Existed"；  查询x，若散列表不含有x，输出"Not Found"，否则输出x所在的链表长度；  删除x，若散列表不含有x，输出"Delete Failed"，否则输出x所在链表删除x后的长度；  **输入输出格式**：  **输入**：  第一行两个整数D(1<=D<=3000)和m(1<=m<=3000)，其中D为散列函数的除数，m为操作数。  接下来的m行，每行两个整数opt和x，分别代表操作类型和操作数。  若opt为0，则代表向散列表中插入x；  若opt为1，代表查询散列表中x是否存在；  若opt为2，(如果散列表中含有x)，删除x。  **输出**：  按需输出。   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）   **第二题：**  2.因为第二个题要求使用链式散列表的形式，所以需要先构造一个struct结构体定义链表的节点，包括element元素pair和next指向下一个节点的指针。还要构造一个有序链表的类sortedChain。其中的数据成员包括pairNode链表的头指针firstNode和了当前链表的长度dSize。  其中的方法函数包括：  **查找函数find(thekey):**  返回匹配的数对的指针，如果不存在，则返回NULL。根据链表遍历的方法从firstnode开始遍历element.first如果找到，根据题目要求，就返回dSize，如果遍历完都没找到，就返回NUL，找到了就返回pair二元组。  **插入函数insert(thepair):**  因为这个链表要求是有序链表，所以插入的时候要找到第一个不比准备要插入的thepair.first小的节点，如果这个节点已经是想要插入的数对了，即已经有匹配的数对了，根据题目要求输出Existed，否则建立新节点，根据链表插入的语法新建一个节点插入链表即可。  定义两个指针，tp，p，tp跟在p的后面，只要p的key值比要插入的thekey值小，就向后移动，如果p的key值正好等于要插入的thekey，就说明已经存在了想插入的数对，根据题目要求，输出Existed。否则进行插入，为新插入的元素建立一个新的pair节点，这个时候一直跟在p后面的tp就派上用场了，把新建立的节点插在tp后面，把新节点的next设为p，完成插入操作。  **删除函数erase(thekey):**  遍历链表，找到第一个不比准备要插入的thepair.first小的节点，如果这个节点的first==thekey，说明找到了匹配的数对，则删除，并且dSize--，并输出dSize的值。如果没找到匹配的数对，根据题目要求则输出Delete Failed。  也是定义两个指针tp，p，tp跟在p后面，当找到p为要删除的元素的时候，利用tp直接越过p，即tp->next=p->next，再把p的空间释放掉。  **hashChains**  有了数对链表的类之后构建一个类hashChain，将前面建立的数对的链表集成到一个table数组中，其中的成员变量为table、dSize字典中数对的个数、divisor散列表函数除数  其中的方法函数包括：  **查找函数find(thekey):**  利用hash函数找到thekey元素所在的桶，直接调用数对链表的方法函数的查找函数find，在这个桶中查找这个元素，如果没找到就输出Not Found  **删除函数erase(thekey):**  利用hash函数找到thekey元素所在的桶，直接调用数对链表的方法函数的删除函数erase在这个桶中删除元素即可。  **插入函数insert(thepair)：**  利用hash函数找到thekey元素所在的桶，直接调用数对链表的方法函数的插入函数insert在这个桶中插入元素即可   1. 测试结果（测试输入，测试输出）   C:\WINDOWS\TEMP\WeChat Files\c334c61bbbaa2508cefe46a289e37e4.jpg   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   在本次实验中，我学习实践了散列表的构建过程，第一个小实验实现了利用数组实现线性表的构建，这么实现的问题是每个key位置只能放一个pair，如果有多个相同key的pair想要存放，就要采用线性探查法。第二个小实验利用链表实现了同一个key位置可以存储多个pair，不过为此要多构建一个有序数对链表sortedChain。  在完成实验要求1的插入与查找较为容易实现，难点在于删除操作。在进行删除时需要注意，哈希表的桶是呈环形排列的，并且在删除元素时需要注意不能把某个数对移到它的起始位置之前，否则可能会导致查找错误。同时也要考虑到保证查找函数的设计没有错误，不然会影响整个程序的正确性。  实验要求2可以设计链表，用于存放桶内的每一组数对，之后将这些链表组合起来构成哈希表。整个要求并不复杂，但在实际执行的时候需要注意许多地方，例如插入，删除元素的时候要注意元素是否在链表中的第一个位置，如果是处于链表头的话，直接更改firstNode即可。在第一次设计链表的时候我没有注意到这一点，于是总是出现报错。之后的检查中注意到了这一点，处理了前一个节点是NULL的情况。  在做实验题目时要善于利用网络，遇到不会实现的功能可以通过搜索引擎查找相关的资料，这样也有助于了解哈希表等数据结构实现的机制。书中的内容也很全面，如果有不懂的代码可以尝试在书中寻找相应的注释。   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）   #include <iostream>  #include <string>  using namespace std;  template <class T>  class Hash;  template <>  class Hash<int>  {  public:      size\_t operator()(const int theKey) const      {          return size\_t(theKey);      }  };  //链表的节点，二元组的形式  template <class K, class E>  struct pairNode  {      pair<K, E> element;      pairNode<K, E> \*next;      pairNode(const pair<K, E> &thepair) { element = thepair; }      pairNode(const pair<K, E> &thepair, pairNode<K, E> \*thenext)      {          element = thepair;          next = thenext;      }  };  template <class K, class E>  class sortedChain  {  public:      //构造函数      sortedChain()      {          firstNode = NULL;          dSize = 0;      }      //析构函数      ~sortedChain();      pair<K, E> \*find(const K &thekey) const;      bool insert(const pair<const K, E> &thepair);      void erase(const K &thekey);      int size() { return dSize; }  private:      pairNode<K, E> \*firstNode;      int dSize;  };  template <class K, class E>  sortedChain<K, E>::~sortedChain()  {      while (firstNode != NULL)      {          pairNode<K, E> \*nextNode = firstNode->next;          delete firstNode;          firstNode = nextNode;      }  }  template <class K, class E>  pair<K, E> \*sortedChain<K, E>::find(const K &thekey) const //返回匹配的数对的指针，如果不存在，则返回NULL  {      pairNode<K, E> \*currentNode = firstNode;      //搜索关键字为thekey的数对      while (currentNode != NULL && currentNode->element.first != thekey)      {          currentNode = currentNode->next;      }      //判断是否匹配      if (currentNode != NULL && currentNode->element.first == thekey) //找到匹配对      {          cout << dSize << endl;          return &currentNode->element;      }      //无匹配的数对      return NULL;  }  template <class K, class E>  bool sortedChain<K, E>::insert(const pair<const K, E> &thepair) //往字典中插入thepair，覆盖已经存在的匹配的数对  {      pairNode<K, E> \*p = firstNode, \*tp = NULL;      //移动指针tp，使thepair可以插在tp后面      while (p != NULL && p->element.first < thepair.first)      {          tp = p;          p = p->next;      }      //检查是否有匹配的数对      if (p != NULL && p->element.first == thepair.first)      {          cout << "Existed" << endl;          return false;      }      //无匹配的数对，为thepair建立新的节点      pairNode<K, E> \*newNode = new pairNode<K, E>(thepair, p);      //在tp之后插入新节点      if (tp == NULL)          firstNode = newNode;      else          tp->next = newNode;      dSize++;      return true;  }  template <class K, class E>  void sortedChain<K, E>::erase(const K &thekey)  {      //删除关键字为thekey的数对      pairNode<K, E> \*p = firstNode, \*tp = NULL;      //搜索关键字为thekey的数对      while (p != NULL && p->element.first < thekey)      {          tp = p;          p = p->next;      }      //确定是否匹配      if (p != NULL && p->element.first == thekey)      {          //找到一个匹配的数对          if (tp == NULL)              firstNode = p->next;          else              tp->next = p->next;          delete p;          dSize--;          cout << dSize << endl;          return;      }      //没找到匹配的数对      cout << "Delete Failed" << endl;  }  template <class K, class E>  class hashChains  {  public:      hashChains(int thedivisor);      ~hashChains() { delete[] table; }      int size() { return dSize; }      void find(const K &) const;      void insert(const pair<const K, E> &thepair);      void erase(const K &thekey);  private:      sortedChain<K, E> \*table;      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  };  template <class K, class E>  hashChains<K, E>::hashChains(int thedivisor)  {      dSize = 0;      divisor = thedivisor;      table = new sortedChain<K, E>[divisor];  }  template <class K, class E>  void hashChains<K, E>::find(const K &thekey) const  {      if (table[hash(thekey) % divisor].find(thekey) == NULL)      {          cout << "Not Found" << endl;      }  }  template <class K, class E>  void hashChains<K, E>::insert(const pair<const K, E> &thepair)  {      if (table[(int)hash(thepair.first) % divisor].insert(thepair))          dSize++;  }  template <class K, class E>  void hashChains<K, E>::erase(const K &thekey)  {      table[hash(thekey) % divisor].erase(thekey);  }  int main()  {      int D, m;      cin >> D >> m;      pair<int, int> p;      hashChains<int, int> h(D);      while (m--)      {          int opt, x;          cin >> opt >> x;          switch (opt)          {          case 0:              p = make\_pair(x, x);              h.insert(p);              break;          case 1:              h.find(x);              break;          case 2:              h.erase(x);          }      }      //system("pause");      return 0;  } | | | |