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（第一题） | | | |
| 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”，否则插入x到散列表中，输出所在的下标。  查询x，若散列表不含有x，输出“-1”，否则输出x对应下标。  删除x，若散列表不含有x，输出“Not Found”，否则输出删除x过程中移动元素的个数。  **输入输出格式**：  **输入**：  第一行两个整数D，m。分别代表散列函数的除数D和操作数m。  接下来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，代表插入x；  若opt为1，代表查询x；  若opt为2，代表删除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）   **第一题**  1.构建一个类class hashTable，其中的成员变量包括，散列表数组table，把类型K映射成一个非负整数的hash函数，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search**：  根据输入的K值查找对应的数对在table中的下标  从利用key值算出来的散列表数组的索引（想要放的位置）i开始向后查找，终止条件是散列表中对应位置table[j]存的元素正好是查询的key值或table[j]==NULL即找到了一个散列表空的位置。或者找了一圈都没找到又回到了位置i。  **查询函数find**：  这个函数是为实验要求建立的查询函数，里面调用了查找函数search，如果找到了K类型的输入thekey，则返回对应的下标索引，否则输出-1。  调用search函数，得到一个返回值是search找到的索引，我们判断这个位置上的元素值是否是我们想找的key，设search返回的索引为b，如果table[b]==NULL或者table[b]!=thekey，说明散列表中没有thekey元素，否则说明有。  **插入函数insert**：  根据输入数对thepair的first的K类型的thekey找到想要插入的位置，然后调用search函数查找得到一个返回的索引b，如果table[b]==NULL，说明散列表这个位置为空，可以插入（不一定是thekey通过哈希函数真正对应的位置，可能是search函数利用线性探查找到的）。  如果table[b]的key值与我想插入的thekey相等，根据题目要求输出Existed。  否则table[b]既不为空，又不等于thekey，说明散列表已经满了，无法再插入了。  **删除函数erase**：  根据输入的K类型的thekey，调用search函数得到一个索引b，如果table[b]为空或者table[b]的key值不等于我想删除的thekey值，就说明散列表中没有我想删除的key。否则，先将table[b]置为NULL（删除），定义三个指针，分别是emp，x，want，含义分别是：  emp是删除元素空出来的位置；  x走在最前面，向前找到需要移动的元素；  want是x找到的需要移动的元素本来应该在的位置  有三种情况需要进行移动操作，如图所示：  C:\WINDOWS\TEMP\WeChat Files\ad0d4a7a472bd2a57511b904d5c6aa6.jpg  进行if判断，如果当前属于者三种情况之一，就将x的元素移动到emp的位置，再将x置为空，将emp移动到x的位置，再从emp的位置开始继续向后找需要移动的元素。直到找到一个table[x]==NULL，说明后面的元素一定不是因为线性探查移动到后面的，否则x为位置不会为空。或者x走了一圈重新回到b的位置，也退出循环。   1. 测试结果（测试输入，测试输出）   1.  C:\WINDOWS\TEMP\WeChat Files\0dea79014f1a74efe88341a0c0078e9.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）   1.  #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);      }  };  template <class K, class E>  class hashTable  {  public:      hashTable(int theDivisor = 16);      ~hashTable() { delete[] table; }      int search(const K &) const;      void find(const K &) const;      void insert(const pair<const K, E> &);      void output();      void erase(K &);  private:      pair<const 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  };  template <class K, class E>  hashTable<K, E>::hashTable(int theDivisor)  {      divisor = theDivisor;      dSize = 0;      //分配和初始化散列表数组      table = new pair<const K, E> \*[divisor];      for (int i = 0; i < divisor; i++)      {          table[i] = NULL;      }  }  template <class K, class E>  int hashTable<K, E>::search(const K &theKey) const  {      int i = (int)hash(theKey) % divisor;      int j = i;      do      {          if (table[j] == NULL || table[j]->first == theKey)              return j;          j = (j + 1) % divisor;      } while (j != i);      return j;  }  template <class K, class E>  void hashTable<K, E>::find(const K &theKey) const  {      //如果没找到就输出-1，找到了就输出下标索引      int b = search(theKey);      if (table[b] == NULL || table[b]->first != theKey)      {          cout << -1 << endl;          return;      }      cout << b << endl;      return;  }  template <class K, class E>  void hashTable<K, E>::insert(const pair<const K, E> &thepair)  {      int b = search(thepair.first);      if (table[b] == NULL) //当前位置没有pair，直接放到这个位置即可      {          table[b] = new pair<const K, E>(thepair);          dSize++;          cout << b << endl;      }      else if (table[b]->first == thepair.first)      {          cout << "Existed" << endl;      }      else      {          cout << "表满，插入失败" << endl;      }  }  template <class K, class E>  void hashTable<K, E>::output()  {      for (int i = 0; i < divisor; i++)      {          if (table[i] == NULL)              cout << "NULL" << endl;          else              cout << table[i]->first << " " << table[i]->second << endl;      }  }  template <class K, class E>  void hashTable<K, E>::erase(K &theKey)  {      int b = search(theKey);      if (table[b] == NULL || table[b]->first != theKey)          cout << "Not Found" << endl;      else      {          table[b] = NULL;          dSize--;          int res = 0;          int emp = b, x = b, want;          //emp是删除元素空出来的位置；          //x走在最前面，向前找到需要移动的元素；          //want是x找到的需要移动的元素本来应该在的位置          do          {              x = (x + 1) % divisor;              if (table[x] == NULL)                  break;              want = table[x]->first % divisor;              if ((emp < x && want > x) || (want > x && emp >= want) || (emp < x && emp >= want))              {                  table[emp] = table[x];                  table[x] = NULL;                  emp = x;                  res++;              }          } while (table[(x + 1) % divisor] != NULL && x != b);          cout << res << endl;      }  }  int main()  {      int D, m;      cin >> D >> m;      hashTable<int, int> h(D);      pair<int, int> p;      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; | | | |