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：202000130138 | 姓名：宋璎航 | | 班级： 20.3 |
| 实验题目：散列表 | | | |
| 实验学时：2 | | 实验日期： 2021/11/18 | |
| 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。 | | | |
| 软件开发环境：  Vsc | | | |
| 1. 实验内容   **1、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。  **2、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）   线性开型寻址即建立哈希表，对应元素实现插入、查询与删除即可，其中删除操作如果确实删除了元素，需要将其后符合特定条件的元素前移。  链表寻址中桶容量理论上是无穷大的，故不需要考虑删除操作时的元素移动问题。为了提高查找效率可以将链表做成有序链表。   1. 测试结果（测试输入，测试输出）   实验一 输入 7 12  1 21  0 1  0 13  0 5  0 23  0 26  0 33  1 33  1 33  1 13  1 5  1 1  **输出**  -1  1  6  5  2  0  3  3  3  6  5  1  实验2 样例输入 7 12  1 21  0 1  0 13  0 5  0 23  0 26  0 33  1 33  1 33  1 13  1 5  1 1 样例输出 Not Found  3  3  1  3  1   1. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）   将链式散列中的有序链表改成跳表时间复杂性会更低，相应的空间复杂性将会上升。   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* "bits/stdc++.h"  class hashTable  {      int \*\*table;      int dsize;      int divisor;  public:      hashTable(int theDivisor)      {          divisor = theDivisor;          dsize = 0;          table = new int \*[divisor];  *for* (size\_t i = 0; i < divisor; ++i)              table[i] = NULL;      }      ~hashTable()      {  *for* (int i = 0; i < divisor; ++i)          {              delete table[i];              table[i] = NULL;          }          delete[] table;      }      int search(const int &theKey) const      {          int i = theKey % divisor;          int j = i;  *do*          {  *if* (table[j] == NULL or \*table[j] == theKey)  *return* j;              j = (j + 1) % divisor;          } *while* (j != i);  *return* j;      }      void find(const int &theKey) const      {          int b = search(theKey);  *if* (table[b] == NULL)          {              std::cout << -1 << std::endl;  *return*;          }  *else* *if* (\*table[b] != theKey)          {              std::cout << -1 << std::endl;  *return*;          }  *else*              std::cout << b << std::endl;      }      void insert(const int &theKey)      {          int b = search(theKey);  *if* (table[b] == NULL)          {              table[b] = new int(theKey);              dsize++;              std::cout << b << std::endl;  *return*;          }  *else*          {  *if* (\*table[b] == theKey)              {                  std::cout << "Existed" << std::endl;  *return*;              }  *else*              {  *//???????????*              }          }      }      void remove(const int &theKey)      {          int cnt = 0;          int b = search(theKey);  *if* (table[b] == NULL)          {              std::cout << "Not Found" << std::endl;  *return*;          }          delete table[b];          table[b] = NULL;          int nul = b;  *for* (int i = (b + 1) % divisor; i != b and table[i] != NULL; i = (i + 1) % divisor)          {  *if* ((\*table[i]) % divisor != i)              {                  int to = (\*table[i]) % divisor;  *if* ((nul < i and nul >= to) or (nul >= to and to > i) or (to > i and i > nul))                  {                      table[nul] = table[i];                      table[i] = NULL;                      nul = i;                      cnt++;                  }              }          }          std::cout << cnt << std::endl;      }  };  int main()  {      int D, m;      std::cin >> D >> m;      hashTable H(D);  *while* (m--)      {          int opt, x;          std::cin >> opt >> x;  *switch* (opt)          {  *case* 0:              H.insert(x);  *break*;  *case* 1:              H.find(x);  *break*;  *case* 2:              H.remove(x);  *break*;  *default*:  *break*;          }      }  *return* 0;  }  }  #include "bits/stdc++.h"  class hashTable  {  int \*\*table;  int dsize;  int divisor;  public:  hashTable(int theDivisor)  {  divisor = theDivisor;  dsize = 0;  table = new int \*[divisor];  for (size\_t i = 0; i < divisor; ++i)  table[i] = NULL;  }  ~hashTable()  {  for (int i = 0; i < divisor; ++i)  {  delete table[i];  table[i] = NULL;  }  delete[] table;  }  int search(const int &theKey) const  {  int i = theKey % divisor;  int j = i;  do  {  if (table[j] == NULL or \*table[j] == theKey)  return j;  j = (j + 1) % divisor;  } while (j != i);  return j;  }  void find(const int &theKey) const  {  int b = search(theKey);  if (table[b] == NULL)  {  std::cout << -1 << std::endl;  return;  }  else if (\*table[b] != theKey)  {  std::cout << -1 << std::endl;  return;  }  else  std::cout << b << std::endl;  }  void insert(const int &theKey)  {  int b = search(theKey);  if (table[b] == NULL)  {  table[b] = new int(theKey);  dsize++;  std::cout << b << std::endl;  return;  }  else  {  if (\*table[b] == theKey)  {  std::cout << "Existed" << std::endl;  return;  }  else  {  //???????????  }  }  }  void remove(const int &theKey)  {  int cnt = 0;  int b = search(theKey);  if (table[b] == NULL)  {  std::cout << "Not Found" << std::endl;  return;  }  delete table[b];  table[b] = NULL;  int nul = b;  for (int i = (b + 1) % divisor; i != b and table[i] != NULL; i = (i + 1) % divisor)  {  if ((\*table[i]) % divisor != i)  {  int to = (\*table[i]) % divisor;  if ((nul < i and nul >= to) or (nul >= to and to > i) or (to > i and i > nul))  {  table[nul] = table[i];  table[i] = NULL;  nul = i;  cnt++;  }  }  }  std::cout << cnt << std::endl;  }  };  int main()  {  int D, m;  std::cin >> D >> m;  hashTable H(D);  while (m--)  {  int opt, x;  std::cin >> opt >> x;  switch (opt)  {  case 0:  H.insert(x);  break;  case 1:  H.find(x);  break;  case 2:  H.remove(x);  break;  default:  break;  }  }  return 0;  } | | | |