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：201805130155 | 姓名： 赵雨晗 | | 班级： 18计科3班 |
| 实验题目：散列表 | | | |
| 实验学时：2 | | 实验日期： 2019.11.06 | |
| 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：  MAC OS: Vs Code + MingW | | | |
| 1. 实验内容（题目内容，输入要求，输出要求）   （1） 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  （2） 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   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)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  (2) 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。  （3）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。   1. 测试结果（测试输入，测试输出，结果分析）   测试输入：  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  测试输出：  Not Found  3  3  1  3  1 结果：与手动模拟结果一致，符合题目要求。   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）   #include<cstdio>  #include<iostream>  #include<cstring>  #include<string>  #include<algorithm>  #include<utility>  #include<stdexcept>  using namespace std;  template<typename T>  struct chainNode  {  T element;  chainNode<T>\* \_next;  chainNode(const T& \_element, chainNode<T>\* \_next = NULL) : element(\_element), \_next(\_next) {}  chainNode(const chainNode<T>\*& c) : element(c->element), \_next(c->\_next) {}  };  template<typename T>  class chain{  public:  chain(int = 10);  chain(const chain<T>&);  ~chain();  bool empty() const;  int size() const;  int find(const T&) const;  void erase(int);  void insert(int, const T&);  void clear();  void push\_back(const T&);    chain<T>& operator=(const chain<T>&);  T& operator[](int);  const T& operator[](int) const;  class iterator;  class const\_iterator;  iterator begin() {return iterator(pHead -> \_next);}  iterator end() {return iterator(NULL);}  const\_iterator begin() const {return const\_iterator(pHead -> \_next);}  const iterator end() const {return const\_iterator(NULL);}  class iterator{  public:  typedef forward\_iterator\_tag iterator\_category;  typedef T value\_type;  typedef ptrdiff\_t difference\_type;  typedef T\* pointer;  typedef T& reference;  iterator(chainNode<T>\* theNode = NULL) :node(theNode) {}  T& operator\*() {return node -> element;}  T\* operator->() {return &node->element; }  iterator& operator++(){  node = node -> \_next;  return \*this;  }  iterator operator++(int){  iterator old = \*this;  node = node->\_next;  return old;  }    bool operator==(const iterator right) const {return node == right.node;}  bool operator!=(const iterator right) const {return node != right.node;}  protected:  chainNode<T>\* node;  };  class const\_iterator{  public:  typedef forward\_iterator\_tag iterator\_category;  typedef T value\_type;  typedef ptrdiff\_t difference\_type;  typedef T\* pointer;  typedef T& reference;  const\_iterator(chainNode<T>\* theNode) :node(theNode) {};  const T& operator\*() { return node->element; }  const T\* operator->() { return &node->element; }  const\_iterator& operator++()  {  node = node->\_next;  return \*this;  }  const\_iterator operator++(int)  {  const\_iterator old = \*this;  node = node->\_next;  return old;  }  bool operator==(const const\_iterator right) const { return node == right.node; }  bool operator!=(const const\_iterator right) const { return node != right.node; }  protected:  chainNode<T>\* node;  };  protected:  chainNode<T>\* pHead;  chainNode<T>\* pTail;  int listSize;  void checkIndex(int) const;  };  template<typename T>  chain<T>::chain(int initialCapacity){  if (initialCapacity < 1) throw out\_of\_range("the initial Capacity of arrayList must > 0");  listSize = 0;  pHead = new chainNode<T>(T());  pTail = pHead;  }  template<typename T>  chain<T>::chain(const chain<T>& c){  pHead = new chainNode<T>(c.pHead->element);  pTail = pHead;  chainNode<T>\* sourceNode = c.pHead ->\_next;  chainNode<T>\* currentNode = pHead;  while(sourceNode != NULL){  pTail = currentNode->\_next;  sourceNode = sourceNode->\_next;  }  listSize = c.listSize;  }  template<typename T>  chain<T>::~chain(){  chainNode<T>\* currentNode = pHead ->\_next;  chainNode<T>\* deleteNode;  while(currentNode != NULL){  deleteNode = currentNode;  currentNode = currentNode ->\_next;  delete deleteNode;  }  delete pHead;  }  template<typename T>  bool chain<T>::empty() const {return listSize == 0;}  template<typename T>  int chain<T>::size() const {return listSize;}  template<typename T>  int chain<T>::find(const T& theElement) const{  int index = 0;  chainNode<T>\* currentNode = pHead ->\_next;  while(currentNode != NULL){  if (currentNode->element == theElement) return index;  currentNode = currentNode->\_next;  ++index;  }  return -1;  }  template<typename T>  void chain<T>::erase(int theIndex){  checkIndex(theIndex);  chainNode<T>\* deleteNode;  chainNode<T>\* pre = pHead;  for (int i = 0; i < theIndex; ++i) pre = pre->\_next;  if (theIndex == listSize - 1) pTail = pre;  deleteNode = pre->\_next;  pre->\_next = pre->\_next->\_next;  --listSize;  delete deleteNode;  }  template<typename T>  void chain<T>::insert(int theIndex, const T& theElement){  if (theIndex < 0 || theIndex > listSize) throw out\_of\_range("illegalIndex");  chainNode<T>\* pre = pHead;  for (int i = 0; i < theIndex; ++i) pre = pre->\_next;  pre->\_next = new chainNode<T>(theElement, pre->\_next);  if (theIndex == listSize) pTail = pre->\_next;  ++listSize;  }  template<typename T>  void chain<T>::clear(){  chainNode<T>\* currentNode = pHead->\_next;  chainNode<T>\* deleteNode;  while(currentNode != NULL){  deleteNode = currentNode;  currentNode = currentNode->\_next;  delete deleteNode;  }  listSize = 0;  pHead ->\_next = NULL;  pTail = pHead;  }  template<typename T>  void chain<T>::push\_back(const T& theElement){  pTail->\_next = new chainNode<T>(theElement, pTail->\_next);  pTail = pTail->\_next;  listSize++;  }  template<typename T>  chain<T>& chain<T>::operator=(const chain<T>& c){  if (this == &c) return \*this;  clear();  chainNode<T>\* currentNode = pHead;  chainNode<T>\* sourceNode = c.pHead->\_next;  while (sourceNode != NULL)  {  pTail = currentNode->\_next = new chainNode<T>(sourceNode->element);  currentNode = currentNode->\_next;  sourceNode = sourceNode->\_next;  }  listSize = c.listSize;  return \*this;  }  template<typename T>  T& chain<T>::operator[](int index){  checkIndex(index);  chainNode<T>\* currentNode = pHead->\_next;  for (int i = 0; i < index; ++i) currentNode = currentNode->\_next;  return currentNode->element;  }  template<typename T>  const T& chain<T>::operator[](int index) const{  checkIndex(index);  chainNode<T> \*currentNode = pHead->\_next;  for (int i = 0; i < index ; ++i) currentNode = currentNode->\_next;  return currentNode->element;  }  template<typename T>  void chain<T>::checkIndex(int theIndex) const{  if (theIndex < 0 || theIndex >= listSize)  throw out\_of\_range("the index is out of range");  }  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<typename K, typename E>  class hashChain{  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&);  int getlen(const K&) const;  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>  int hashChain<K, E>::getlen(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 table[index].size();  return 0;  }  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){  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;  }  }  int main(){  int D, m, op, x;  scanf("%d%d", &D, &m);  hashChain<int, int>mp(D);  while(m--){  scanf("%d%d", &op, &x);  if (op == 0){  if (mp.find(x) != nullptr) printf("Existed\n");  else mp.insert(x, 1);  } else if (op == 1){  if (mp.getlen(x) == 0) printf("Not Found\n");  else printf("%d\n", mp.getlen(x));  } else {  if (mp.find(x) == nullptr) printf("Delete Failed\n");  else printf("%d\n", mp.getlen(x) - 1), mp.erase(x);  }  }  } | | | |