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：202000130143	姓名： 郑凯饶	班级： 计科 20.1
实验题目：散列表		
实验学时：2	实验日期： 1117	

实验目的：

1. 掌握散列表结构的定义与实现；
2. 掌握散列表结构的应用。

软件开发环境：

Windows 10 家庭中文版 64 位（10.0，版本 18363）

Dev-C++ IDE

1. 实验内容

使用线性探查以及链表的方法实现散列表。

2. 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> 使用数组实现

```
template<class K, class E>
class hashTable
{
   public:
      hashTable(int theDivisor = 11);
      ~hashTable(){delete [] table;}

      bool empty() const {return dSize == 0;}
      int size() const {return dSize;}
      pair<const K, E>* find(const K&) const;
      void insert(const pair<const K, E>&);
      void output(ostream& out) const;
        void del(const K&);

   protected:
      int search(const K&) const;
      pair<const K, E>** table;  // hash table
      hash<K> Hash;              // maps type K to nonnegative integer
      int dSize;                 // number of pairs in dictionary
      int divisor;               // hash function divisor
};
```

HashTable 的主要实现参考了课程网站上的代码。

Public 方法：

Find()：调用 Search()，根据其返回信息判断是否找到；

Insert()：调用 Search()，若找到则替换成相应的“值”（输出“Existed”），若没找到则插入；

Output()：输出，测试用。

Del()：调用 Search()，若未找到则返回 NULL（输出“Not Found”），若找到删除指定元素，并从起始桶开始搜索，将可以向前移动的元素移至“空出”的位置，直至遇到空桶或者是回到起始“桶”。通过当前元素的起始“桶”，当前所在的“桶”以及“空桶”的位置关系可以判断移动是否“合法”，实际有 3 种大小关系“合法”，另 3 种“不合法”。

Private：

Search()：比较核心的方法，从哈希映射的起始桶开始寻找指定的“键”；

Table：存储指向相应元素的指针，为什么不直接存储键值对的话是因为指针可以通过 NULL 标识空桶，不会占用“键值对”；

dSize：哈希表大小；

divisor：除数，根据题目条件设定。

2> 使用链表实现

```
template<class K, class E>
class HashChains : public dictionary<K,E>
{
   public:
      HashChains(int theDivisor = 11)
      {
         divisor = theDivisor;
         dSize = 0;

         // allocate and initialize Hash table array
         table = new sortedChain<K,E> [divisor];
      }

      ~HashChains(){delete [] table;}

      bool empty() const {return dSize == 0;}
      int size() const {return dSize;}

      pair<const K, E>* find(const K& theKey) const
        {return table[Hash_(theKey) % divisor].find(theKey);}  // find in sortedChain

      void insert(const pair<const K, E>& thePair)
      {
         int homeBucket = (int) Hash_(thePair.first) % divisor; // distributed bucket
         int homeSize = table[homeBucket].size();
         table[homeBucket].insert(thePair);     // insert into sortedChain
         if (table[homeBucket].size() > homeSize)
            dSize++;
      }

      void erase(const K& theKey)
        {table[Hash_(theKey) % divisor].erase(theKey);}

      void output(ostream& out) const   // out ?!!
      {
         for (int i = 0; i < divisor; i++)
            if (table[i].size() == 0)
               cout << "NULL" << endl;
            else
               cout << table[i] << endl;
      }


   protected:
      sortedChain<K, E>* table;  // Hash table
      Hash<K> Hash_;              // maps type K to nonnegative integer
      int dSize;                 // number of elements in list
      int divisor;               // Hash function divisor
};
```

与数组不同，链表结构不需要通过线性开型寻址的方式避免“键”的冲突。主要使用之前实现的 sortedChain 作为一个个“桶”，提高使用的效率。

Public 方法：

Find()：哈希映射之后，在对应的“桶”寻找元素；

Insert()： 哈希映射至对应“桶”，调用 sortedChain 的 insert()方法插入；

Erase()：哈希映射至对应“桶”，调用 sortedChain 的 erase()方法插入。

Private：

Table：存储每个“桶”的链表头结点。

3. 测试结果（测试输入，测试输出）

在 OJ 平台上成功提交。

4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）
在散列表的实现中我主要借助了课程网站的“标程”进行“二次设计”，在阅读别人代码的过程中深受启发。

5. 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<bits/stdc++.h>
using namespace std;

/* 1.hash -> Hash
*/

template <class K> class Hash;

template<>
class Hash<string>
{
   public:
      size_t operator()(const string theKey) const
      {// Convert theKey to a nonnegative integer.
         unsigned long hashValue = 0;
         int length = (int) theKey.length();
         for (int i = 0; i < length; i++)
            hashValue = 5 * hashValue + theKey.at(i);

         return size_t(hashValue);
      }
};

template<>
class Hash<int>
{
   public:
      size_t operator()(const int theKey) const
      {return size_t(theKey);}
};

template<>
class Hash<long>
{
   public:
      size_t operator()(const long theKey) const
      {return size_t(theKey);}
};
```

```
39.
40.template<class K, class E>
41.class hashTable
42.{
43.   public:
44.      hashTable(int theDivisor = 11);
45.      ~hashTable(){delete [] table;}
46.
47.      bool empty() const {return dSize == 0;}
48.      int size() const {return dSize;}
49.      pair<const K, E>* find(const K&) const;
50.      void insert(const pair<const K, E>&);
51.      void output(ostream& out) const;
52.  void del(const K&);
53.
54.   protected:
55.      int search(const K&) const;
56.      pair<const K, E>** table;  // hash table
57.      hash<K> Hash;              // maps type K to nonnegative integer
58.      int dSize;                 // number of pairs in dictionary
59.      int divisor;               // hash function divisor
60.};
61.
62.template<class K, class E>
63.hashTable<K,E>::hashTable(int theDivisor)
64.{
65.   divisor = theDivisor;
66.   dSize = 0;
67.
68.   // allocate and initialize hash table array
69.   table = new pair<const K, E>* [divisor];
70.   for (int i = 0; i < divisor; i++)
71.      table[i] = NULL;
72.}
73.
74.template<class K, class E>
75.int hashTable<K,E>::search(const K& theKey) const
76.{// Search an open addressed hash table for a pair with key theKey.
77. // Return location of matching pair if found, otherwise return
78. // location where a pair with key theKey may be inserted
79. // provided the hash table is not full.
80.
81.   int i = (int) Hash(theKey) % divisor;  // home bucket
82.   int j = i;    // start at home bucket
83.   do
84.   {
```

```
85.       if (table[j] == NULL || table[j]->first == theKey)
86.          return j;
87.       j = (j + 1) % divisor;  // next bucket
88.    } while (j != i);          // returned to home bucket?
89.
90.    return j;  // table full
91.}
92.
93.template<class K, class E>
94.pair<const K,E>* hashTable<K,E>::find(const K& theKey) const
95.{// Return pointer to matching pair.
96. // Return NULL if no matching pair.
97.    // search the table
98.    int b = search(theKey);
99.
100.    // see if a match was found at table[b]
101.    if (table[b] == NULL || table[b]->first != theKey) {
102.     cout << "-1\n";
103.     return NULL;           // no match
104.    }
105.
106. cout << b << '\n';
107.    return table[b];  // matching pair
108.}
109.
110.template<class K, class E>
111.void hashTable<K,E>::insert(const pair<const K, E>& thePair)
112.{// Insert thePair into the dictionary. Overwrite existing
113. // pair, if any, with same key.
114. // Throw hashTableFull exception in case table is full.
115.    // search the table for a matching pair
116.    int b = search(thePair.first);
117.
118.    // check if matching pair found
119.    if (table[b] == NULL)
120.    {
121.       // no matching pair and table not full
122.       table[b] = new pair<const K,E> (thePair);
123.       dSize++;
124.       cout << b << '\n';
125.    }
126.    else
127.    {// check if duplicate or table full
128.       if (table[b]->first == thePair.first)
129.       {// duplicate, change table[b]->second
130.//         table[b]->second = thePair.second;
```

```
131.    cout << "Existed\n";
132.       }
133.//      else // table is full
134.//         throw hashTableFull();
135.    }
136.}
137.
138.template<class K, class E>
139.void hashTable<K,E>::output(ostream& out) const
140.{// Insert the hash table into the stream out.
141.    for (int i = 0; i < divisor; i++)
142.       if (table[i] == NULL)
143.          cout << "NULL ";// << endl;
144.       else
145.          cout << table[i]->first << " ";
146.//               << table[i]->second << endl;
147.}
148.
149.// overload <<
150.template <class K, class E>
151.ostream& operator<<(ostream& out, const hashTable<K,E>& x)
152.    {x.output(out); return out;}
153.
154.template<class K, class E>
155.void hashTable<K,E>::del(const K& tar) {
156.  int b = search(tar);
157.  if (table[b] == NULL || table[b]->first != tar){
158.   cout << "Not Found\n";
159.   return;
160.//  return NULL;
161.  }
162.  delete table[b];
163.// table[b] = NULL;
164.  dSize--;
165.  // 1 2 3 4 5 6 7 8
166.  // 1 n 3 n 5 10 n n
167.  // ilegal:
168.  // ini . i . pre
169.  // i . pre . ini
170.  // pre . ini . i
171.  // legal:
172.  // i . ini . pre
173.  // ini . pre . i
174.  // pre . i . ini
175.// cout << "ok\n";
176.  int pre = b, cnt = 0; // 元素前移只能移动到上一个移动了的元素的位置或者是删除位
```

```
177.
178.// for (int i = 1; i < divisor; i++) { // dSize -> divisor
179. for (int i = 1; i < dSize; i++) { // dSize = divisor
180.  b = (b + 1) % divisor;
181.
182.  if (table[b] == NULL) break;
183.  else {
184.   int ini = Hash(table[b]->first) % divisor;
185.   if (ini == b) continue;
186.   else if ((ini < b && b < pre) || (b < pre && pre < ini) || (pre < ini &&
    ini < b)) { // 移动不合法： 将元素移至初始桶之前
187.    continue;
188.   }
189.   else { // if((b < ini && ini <= pre) || (ini <= pre && pre < b) || (pre <
     b && b < ini)){
190.    table[pre] = table[b];
191.    pre = b;
192.    cnt++;
193.   }
194.  }
195. }
196. table[pre] = NULL;
197.// cout << "del: " << cnt << '\n';
198. cout << cnt << '\n';
199.}
200.
201.int main(){
202.
203.// freopen("out.txt", "w", stdout);
204.
205. int D, m; cin >> D >> m;
206. hashTable<int, int> z(D);
207. pair<int, int> p;
208. p.second = 0;
209.
210. for (int i = 1, opt, d; i <= m; i++) {
211.  cin >> opt >> d;
212.  p.first = d;
213.  if (opt == 0) {
214.   z.insert(p);
215.  }
216.  else if (opt == 1) {
217.   z.find(p.first);
218.  }
219.  else if (opt == 2) {
220.   z.del(p.first);
```

```
  }
//  cout << z << '\n';
 }

 return 0;
}
```

2.

```
#include<bits/stdc++.h>
using namespace std;

template <class K> class Hash;

// "Hash.h"
template<>
class Hash<string>
{
   public:
      size_t operator()(const string theKey) const
      {// Convert theKey to a nonnegative integer.
         unsigned long HashValue = 0;
         int length = (int) theKey.length();
         for (int i = 0; i < length; i++)
            HashValue = 5 * HashValue + theKey.at(i);

         return size_t(HashValue);
      }
};

template<>
class Hash<int>
{
   public:
      size_t operator()(const int theKey) const
      {return size_t(theKey);}
};

template<>
class Hash<long>
{
   public:
      size_t operator()(const long theKey) const
      {return size_t(theKey);}
};

```

```
38.
39.// "dictionary.h"
40.template<class K, class E>
41.class dictionary
42.{
43.   public:
44.      virtual ~dictionary() {}
45.      virtual bool empty() const = 0;
46.                  // return true iff dictionary is empty
47.      virtual int size() const = 0;
48.                  // return number of pairs in dictionary
49.      virtual pair<const K, E>* find(const K&) const = 0;
50.                  // return pointer to matching pair
51.      virtual void erase(const K&) = 0;
52.                  // remove matching pair
53.      virtual void insert(const pair<const K, E>&) = 0;
54.                  // insert a (key, value) pair into the dictionary
55.};
56.
57.
58.template <class K, class E>
59.struct pairNode
60.{
61.   typedef pair<const K, E> pairType;
62.   pairType element;
63.   pairNode<K,E> *next;
64.
65.   pairNode(const pairType& thePair):element(thePair){}
66.   pairNode(const pairType& thePair, pairNode<K,E>* theNext)
67.            :element(thePair){next = theNext;}
68.};
69.
70.
71.//"sortedChain.h"
72.template<class K, class E>
73.class sortedChain : public dictionary<K,E>
74.{
75.   public:
76.      sortedChain() {firstNode = NULL; dSize = 0;}
77.      ~sortedChain();
78.
79.      bool empty() const {return dSize == 0;}
80.      int size() const {return dSize;}
81.      pair<const K, E>* find(const K&) const;
82.      void erase(const K&);
83.      void insert(const pair<const K, E>&);
```

```
84.       void output(ostream& out) const;
85.
86.    protected:
87.       pairNode<K,E>* firstNode;  // pointer to first node in chain
88.       int dSize;                 // number of elements in dictionary
89.};
90.
91.template<class K, class E>
92.sortedChain<K,E>::~sortedChain()
93.{// Destructor.  Delete all nodes.
94.   while (firstNode != NULL)
95.   {// delete firstNode
96.      pairNode<K,E>* nextNode = firstNode->next;
97.      delete firstNode;
98.      firstNode = nextNode;
99.   }
100.}
101.
102.template<class K, class E>
103.pair<const K,E>* sortedChain<K,E>::find(const K& theKey) const
104.{// Return pointer to matching pair.
105. // Return NULL if no matching pair.
106.   pairNode<K,E>* currentNode = firstNode;
107.
108.   // search for match with theKey
109.   while (currentNode != NULL &&
110.          currentNode->element.first != theKey)
111.      currentNode = currentNode->next;
112.
113.   // verify match
114.   if (currentNode != NULL && currentNode->element.first == theKey) {
115.      // yes, found match
116.      cout << this -> dSize << '\n';
117.      return &currentNode->element;
118.   }
119.
120.
121.   // no match
122.   cout << "Not Found\n";
123.   return NULL;
124.}
125.
126.template<class K, class E>
127.void sortedChain<K,E>::insert(const pair<const K, E>& thePair)
128.{// Insert thePair into the dictionary. Overwrite existing
129. // pair, if any, with same key.
```

```
130.    pairNode<K,E> *p = firstNode,
131.                 *tp = NULL; // tp trails p
132.
133.    // move tp so that thePair can be inserted after tp
134.    while (p != NULL && p->element.first < thePair.first)
135.    {
136.       tp = p;
137.       p = p->next;
138.    }
139.
140.    // check if there is a matching pair
141.    if (p != NULL && p->element.first == thePair.first)
142.    {// replace old value
143.      cout << "Existed\n";
144.//       p->element.second = thePair.second;
145.       return;
146.    }
147.
148.    // no match, set up node for thePair
149.    pairNode<K,E> *newNode = new pairNode<K,E>(thePair, p);
150.
151.    // insert newNode just after tp
152.    if (tp == NULL) firstNode = newNode;
153.    else tp->next = newNode;
154.
155.    dSize++;
156.    return;
157.}
158.
159.template<class K, class E>
160.void sortedChain<K,E>::erase(const K& theKey)
161.{// Delete the pair, if any, whose key equals theKey.
162.    pairNode<K,E> *p = firstNode,
163.                 *tp = NULL; // tp trails p
164.
165.    // search for match with theKey
166.    while (p != NULL && p->element.first < theKey)
167.    {
168.       tp = p;
169.       p = p->next;
170.    }
171.
172.    // verify match
173.    if (p != NULL && p->element.first == theKey)
174.    {// found a match
175.       // remove p from the chain
```

```
176.       if (tp == NULL) firstNode = p->next;  // p is first node
177.       else tp->next = p->next;
178.
179.       delete p;
180.       dSize--;
181.       cout << dSize << '\n';
182.    }
183.    else cout << "Delete Failed\n";
184. }
185.
186. template<class K, class E>
187. void sortedChain<K,E>::output(ostream& out) const
188. {// Insert the chain elements into the stream out.
189.    for (pairNode<K,E>* currentNode = firstNode;
190.                        currentNode != NULL;
191.                        currentNode = currentNode->next)
192.       out << currentNode->element.first << " "
193.           << currentNode->element.second << "  ";
194. }
195.
196. // overload <<
197. template <class K, class E>
198. ostream& operator<<(ostream& out, const sortedChain<K,E>& x)
199.    {x.output(out); return out;}
200.
201.
202. template<class K, class E>
203. class HashChains : public dictionary<K,E>
204. {
205.    public:
206.       HashChains(int theDivisor = 11)
207.       {
208.          divisor = theDivisor;
209.          dSize = 0;
210.
211.          // allocate and initialize Hash table array
212.          table = new sortedChain<K,E> [divisor];
213.       }
214.
215.       ~HashChains(){delete [] table;}
216.
217.       bool empty() const {return dSize == 0;}
218.       int size() const {return dSize;}
219.
220.       pair<const K, E>* find(const K& theKey) const
221.          {return table[Hash_(theKey) % divisor].find(theKey);} // find in so
```

```
   rtedChain
222.
223.       void insert(const pair<const K, E>& thePair)
224.       {
225.          int homeBucket = (int) Hash_(thePair.first) % divisor; // distribut
   ed bucket
226.          int homeSize = table[homeBucket].size();
227.          table[homeBucket].insert(thePair);  // insert into sortedChain
228.          if (table[homeBucket].size() > homeSize)
229.             dSize++;
230.       }
231.
232.       void erase(const K& theKey)
233.          {table[Hash_(theKey) % divisor].erase(theKey);}
234.
235.       void output(ostream& out) const // out ?!!
236.       {
237.          for (int i = 0; i < divisor; i++)
238.             if (table[i].size() == 0)
239.                cout << "NULL" << endl;
240.             else
241.                cout << table[i] << endl;
242.       }
243.
244.
245.    protected:
246.       sortedChain<K, E>* table;  // Hash table
247.       Hash<K> Hash_;              // maps type K to nonnegative integer
248.       int dSize;                 // number of elements in list
249.       int divisor;               // Hash function divisor
250. };
251.
252.
253. // overload <<
254. template <class K, class E>
255. ostream& operator<<(ostream& out, const HashChains<K,E>& x)
256.    {x.output(out); return out;}
257.
258.
259.
260. int main(){
261.
262.
263.  int D, m;
264.  cin >> D >> m;
265.
```

```
 HashChains<int, int> obj(D);
 pair<int, int> p;
 p.second = 0;

 for (int i = 1, opt, x; i <= m; i++) {
  cin >> opt >> x;

  if (opt == 0) {
   p.first = x;
   obj.insert(p);
  }
  else if (opt == 1) {
   obj.find(x);
  }
  else if (opt == 2) {
   obj.erase(x);
  }
 }
 return 0;
}


/*

1 Mary
2 Ken
3 Kitty

*/

```