chektien complete

1 contributor

Raw Blame

259 lines (214 sloc) 7.39 KB

```
1  //#define DEBUG_INSERT
2  //#define DEBUG_REMOVE
3
4  #include <iostream>
5  #include <math.h>
6  #include <string.h>
7  #include "support.h"
8  #include "ChHashTable.h"
9
10 using std::cout;
11 using std::endl;
12
13 template <typename T>
14 ChHashTable<T>::ChHashTable(const ChHashTable<T>::HTConfig& Config, ObjectAllocator* allocator)
15     // init stats
16     stats_.TableSize_ = config_.InitialTableSize_;
17     stats_.HashFunc_ = config_.HashFunc_;
18
19     // init table array
20     table_ = new ChHTHeadNode[stats_.TableSize_];
21 }
22
23 template <typename T>
24 ChHashTable<T>::~ChHashTable() {
25     clear();
26     delete[] table_;
27 }
28
29 template <typename T>
30 void ChHashTable<T>::insert(const char *Key, const T& Data) throw(HashTableException) {
```

```
31
32      // calc load factor
33      double load_factor = (stats_.Count_+1) / static_cast<double>(stats_.TableSize_);
34
35  #ifdef DEBUG_INSERT
36      cout << "insert: {" << Key << "} load_factor=" << load_factor << endl;
37  #endif
38
39      // expand table if > max load
40      if ( load_factor > config_.MaxLoadFactor_ ) {
41
42  #ifdef DEBUG_INSERT
43          cout << "insert: {" << Key << "} EXPANDing table size="  << stats_.TableSize_ << endl;
44  #endif
45
46          // get new table size
47          auto old_tablesize = stats_.TableSize_;
48          auto factor = ceil(stats_.TableSize_ * config_.GrowthFactor_);
49          stats_.TableSize_ = GetClosestPrime(static_cast<unsigned>(factor));
50
51          // re-insert all items into new table
52          auto old_table = table_;
53          table_ = new ChHTHeadNode[stats_.TableSize_];
54          for (auto i=0; i<old_tablesize; ++i) {
55              auto hnode = old_table[i];
56              auto node = hnode.Nodes;
57
58              while (node) {
59                  //++stats_.Probes_;
60                  ChHTNode* prev_node;
61                  auto tmp = node;
62                  node = node->Next;
63
64                  // get i into the table
65                  auto i = stats_.HashFunc_(tmp->Key, stats_.TableSize_);
66
67                  // check if node exists in new table
68                  auto new_node = find_node(i, tmp->Key, prev_node);
69
70                  // find the node to check for dupes (will throw if dupe)
71                  // - if this is the first node then no need to waste time checking
72                  if (new_node != nullptr)
73                      throw HashTableException(HashTableException::E_DUPLICATE, "Key to be re-ins
74
75                  tmp->Next = table_[i].Nodes;
76                  table_[i].Nodes = tmp;
77                  ++table_[i].Count;
78              }
79              old_table[i].Nodes = nullptr;
80          }
81          delete[] old_table;
82          ++stats_.Expansions_;
```

```
83
84  #ifdef DEBUG_INSERT
85          cout << "insert: {" << Key << "} EXPANDED table new size="  << stats_.TableSize_ << en
86  #endif
87
88      }
89
90      // get i into the table
91      auto i = stats_.HashFunc_(Key, stats_.TableSize_);
92
93      // find the node to check for dupes (will throw if dupe)
94      // - if this is the first node then no need to waste time checking
95      ChHTNode* prev_node;
96      auto node = find_node(i, Key, prev_node);
97      if (node != nullptr)
98          throw HashTableException(HashTableException::E_DUPLICATE, "Key to be inserted is alrea
99
100     // create and insert the node
101     node = make_node(Key, Data);
102     node->Next = table_[i].Nodes;
103     table_[i].Nodes = node;
104     ++table_[i].Count;
105     ++stats_.Count_;
106 }
107
108 template <typename T>
109 void ChHashTable<T>::insert_node(const unsigned& i, ChHTNode* node) {
110     //// get the correct i into table
111     //auto i = stats_.HashFunc_(node->Key, stats_.TableSize_);
112
113 //#ifdef DEBUG_INSERT
114     //cout << "insert_node: {" << node->Key << "," << node->Data << "}, hashed i=" << i << endl
115 //#endif
116
117     // insert into table
118     // - set new node as the head
119     // - update stats
120     node->Next = table_[i].Nodes;
121     table_[i].Nodes = node;
122     ++table_[i].Count;
123 }
124
125 //TODO try to use the shared find_node method
126 template <typename T>
127 void ChHashTable<T>::remove(const char *Key) throw(HashTableException) {
128     // get i into the table
129     auto i = stats_.HashFunc_(Key, stats_.TableSize_);
130
131 #ifdef DEBUG_REMOVE
132     cout << "remove: {" << Key << ", DATA}, hashed i=" << i << endl;
133 #endif
134
```

```
135     // iterate through linked list to find key
136     ChHTNode* prev_node;
137     auto node = find_node(i, Key, prev_node);
138 
139     if (node == nullptr)
140         throw HashTableException(HashTableException::E_ITEM_NOT_FOUND, "Key to be deleted not 
141 
142     // remove item
143     auto tmp = node;
144     if (prev_node == node)
145         table_[i].Nodes = node->Next;
146     else
147         prev_node->Next = node->Next;
148     delete_node(tmp);
149     --table_[i].Count;
150     --stats_.Count_;
151 }
152 
153 template <typename T>
154 const T& ChHashTable<T>::find(const char *Key) const throw(HashTableException) {
155     // get i into the table
156     auto i = stats_.HashFunc_(Key, stats_.TableSize_);
157 
158     // find node
159     ChHTNode* prev_node;
160     auto node = find_node(i, Key, prev_node);
161     if (node == nullptr)
162         throw HashTableException(HashTableException::E_ITEM_NOT_FOUND, "Key to find does not ex
163     return node->Data;
164 }
165 
166 template <typename T>
167 typename ChHashTable<T>::ChHTNode* ChHashTable<T>::find_node(const unsigned& i, const char* Key
168     // iterate through linked list to find key
169     if (table_[i].Count == 0) {
170         ++stats_.Probes_;
171         return nullptr;
172     }
173     else {
174         auto node = table_[i].Nodes;
175         prev_node = node;
176         while (node) {
177             ++stats_.Probes_;
178 
179 #ifdef DEBUG_INSERT
180     cout << "find_node: for " << Key << " looping checking against node{" << node->Key << "} Pr
181 #endif
182 
183             if (strcmp(node->Key, Key) == 0)
184                 return node;
185 
186             prev_node = node;
```

```
            node = node->Next;
        }
        ++stats_.Probes_;
        return node;
    }
}

//template <typename T>
//typename ChHashTable<T>::ChHTNode* ChHashTable<T>::find_node(const T& Data) const{
    //// get i into the table
    //auto i = stats_.HashFunc_(key, stats_.TableSize_);
    //++stats_.Probes_;

    //// iterate through linked list to find key
    //if (table_[i].Count == 0)
        //return nullptr;
    //else {
        //auto node = table_[i].Nodes;
        //while (node) {
            //if (strcmp(node->Key, key) == 0)
                //break;

            //node = node->Next;

            ////++stats_.Probes_;
        //}
        //return node;
    //}
//}

template <typename T>
void ChHashTable<T>::clear(void) {
    for (auto i=0; i<stats_.TableSize_; ++i) {
        auto hnode = table_[i];
        auto node = hnode.Nodes;
        while (node) {
            auto tmp = node;
            node = node->Next;
            delete_node(tmp);
            --table_[i].Count;
            --stats_.Count_;
        }
        table_[i].Nodes = nullptr;
    }
}

template <typename T>
HTStats ChHashTable<T>::GetStats(void) const {
    return stats_;
}

template <typename T>
```

```
239 const typename ChHashTable<T>::ChHTHeadNode* ChHashTable<T>::GetTable(void) const {
240     return table_;
241 }
242 
243 template <typename T>
244 typename ChHashTable<T>::ChHTNode* ChHashTable<T>::make_node(const char* Key, const T& Data) {
245     ChHTNode* node;
246     if (oa_)
247         node = new (oa_->Allocate()) ChHTNode(Data);
248     else
249         node = new ChHTNode(Data);
250     std::strcpy(node->Key, Key);
251     return node;
252 }
253 
254 template <typename T>
255 void ChHashTable<T>::delete_node(ChHTNode* node) {
256     if (oa_)
257         node->~ChHTNode();
258     delete node;
259 }
```