Hash Tables and Maps

CSCI 2100 Data Structures

Spring 2018

Tae-Hyuk (Ted) Ahn
Department of Computer Science
Program of Bioinformatics and Computational Biology
Saint Louis University
STL Containers

- Sequence Containers – store sequences of values
  - vector, deque, list

- Associative Containers – use “keys” to access data rather than position (Account #, ID, SSN, …)
  - set
  - multiset
  - map
  - multimap

- Container Adapters – specialized interfaces to general containers
  - stack, queue, priority_queue
Associative Containers

- Stores elements based on a key
- Key can consist of one or more attributes to uniquely identify each element (we will assume only one attribute).
- Example: Department of Motor Vehicles (DMV) uses license-plate # to identify a vehicle.
- Similar to vector & list – it is another storage structure with operations to access & modify elements.
- Main difference is that associative-container uses the key rather than an index (vector) or linear search (list) to retrieve an element.
Associative-Container : set

- Stores a set of values (i.e., “keys”)
- Values are unique (stored only once)
- Implemented as a binary search tree
  - #include <set>
  - set<string> s;
- Fast insert and delete
  - insert, erase
- Fast search
  - find
- Other operations
  - size, empty, clear, . . .
```cpp
#include <iostream>
#include <set>
#include <string>

using namespace std;

int main() {
    set<string> setOfNumbers;

    // Lets insert four elements
    setOfNumbers.insert("first");
    setOfNumbers.insert("second");
    setOfNumbers.insert("third");
    setOfNumbers.insert("first");

    // Only 3 elements will be inserted
    cout << "Set Size = " << setOfNumbers.size() << endl;

    // Iterate through all the elements in a set and display the value.
    for (set<string>::iterator it = setOfNumbers.begin(); it != setOfNumbers.end(); ++it) {
        cout << ' ' << *it;
    }
    cout << '
';

    // Search for element in set using find member function
    set<string>::iterator it = setOfNumbers.find("second");
    if (it != setOfNumbers.end()) {cout << "'first' found" << endl;}
    else {cout << "'first' not found" << endl;}

    // Search for element in set using find member function
    it = setOfNumbers.find("fourth");
    if (it != setOfNumbers.end()) {cout << "'fourth' found" << endl;}
    else {cout << "'fourth' not found" << endl;}

    return 0;
}
```
Associative Containers: multiset

- Stores a set of values (i.e., “keys”)
- Like set, but values need not be unique
- Implemented as a balanced binary search tree (red-black tree)
  - #include <set>
  - multiset<string> ms;
- Fast insert and delete
  - insert, erase
- Fast search
  - find
- Other operations
  - size, empty, clear, ...
Associative Containers: map

- Stores a set of (key, value) pairs
- Each key has one value
- Implemented as a binary search tree

```
#include <map>

//define a map with
//keys of type string
//and values of int
map<string, int> m;
```

- Fast insert and delete

```
m[“Ted”] = 99;
insert, erase
```

```
“Ted” | 99
“Michael” | 105
“James” | 23
```
Associative Containers: map

- Fast search
  - int x = m["Ted"];
  - find

- Other operations
  - size, empty, clear, ...
STL Maps: Constructors

- Copy constructor:

```cpp
map<char, int> m;
map<char, int> m2(m);
```
An STL map is implemented as a tree-structure, where each node holds a “pair”

Most important to know when retrieving data from the table
- Some functions return the pair, not just the value

A pair has two fields, first (holding the key) and second (holding the value)
STL Map: Data Storage

- If you have a **pair object**, you can use the following code to print the key and value:

  ```cpp
  cout << myPairObject.first << " " << myPairObject.second;
  ```

- If you have a **pointer to the pair object**, use the arrow operator instead

  ```cpp
  cout << myPairObject->first << " " << myPairObject->second;
  ```
STL Map: Data Storage

● Access element `at`
  - Returns a reference to the mapped value of the element identified with key k.
  - If k does not match the key of any element in the container, the function throws an `out_of_range` exception.

● Access element `[ ]`
  - If k matches the key of an element in the container, the function returns a reference to its mapped value.
  - If k does not match the key of any element in the container, the function inserts a new element with that key and returns a reference to its mapped value.
```cpp
#include <iostream>
#include <string>
#include <map>

using namespace std;

int main () {
    map<int,string> mymap;
    mymap[1] = "Banana";
    mymap[2] = "Peach";
    mymap[3] = mymap[2];
    cout << "mymap[1] is " << mymap[1] << '\n';
    mymap.at(1) = "Melon";
    mymap.at(2) = "Strawberry";
    mymap.at(3) = "Kiwi";
    map<int,string>::iterator it;
    for (it = mymap.begin(); it != mymap.end(); ++it)
        cout << it->first << " => " << it->second << endl;
    it = mymap.find(2);
    if (it != mymap.end())
        cout << "The value of the key [2] is " << it->second << endl;
    return 0;
}
```
What is a Hash Table?

- The simplest kind of hash table is an array of records.
- This example has 501 records.

An array of records
What is a Hash Table?

- Each record has a special field, called its **key**.
- In this example, the **key** is a long integer field called **Number**.

An array of records
What is a Hash Table?

- The **Number** might be a person's identification number (e.g., student ID, SSN), and the rest of the record has information about the person.

An array of records
What is a Hash Table?

- When a hash table is in use, some spots contain valid records, and other spots are "empty".

An array of records
Inserting a New Record

- In order to insert a new record, the key must somehow be converted to an array index.
- The index is called the hash value of the key.

An array of records

\[
\begin{array}{cccccc}
\hline
\text{Number} & 344991607 & \text{Number} & 775672751 & \text{Number} & 699072358 & \text{Number} \\
\text{Number} & \text{393802035} & \text{Number} & 582739652 & & & \\
\end{array}
\]
Inserting a New Record

- Typical way to create a hash value:
  - Number mod ArraySize
  - $393802035 \mod 501 = 3$

An array of records

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>[1]</td>
<td>[2]</td>
<td>[3]</td>
<td>[4]</td>
</tr>
<tr>
<td>Number 344991607</td>
<td>Number 775672751</td>
<td></td>
<td>Number 699072358</td>
<td></td>
</tr>
</tbody>
</table>
## Inserting a New

- The **hash value** is used for the **location** of the new record.

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td><img src="image.png" alt="Number" /></td>
<td><img src="image.png" alt="Number" /></td>
<td><img src="image.png" alt="Number" /></td>
<td><img src="image.png" alt="Number" /></td>
<td><img src="image.png" alt="Number" /></td>
<td><img src="image.png" alt="Number" /></td>
</tr>
<tr>
<td>Number 344991607</td>
<td>Number 775672751</td>
<td>Number 393802035</td>
<td>Number 699072358</td>
<td><img src="image.png" alt="Number" /></td>
<td><img src="image.png" alt="Number" /></td>
</tr>
</tbody>
</table>

An array of records
Hash Collisions

- Here is another new record to insert.

An array of records
Hash Collisions

- Here is another new record to insert.
  - $493375785 \mod 501 = 3$

<table>
<thead>
<tr>
<th>[ 0 ]</th>
<th>[ 1 ]</th>
<th>[ 2 ]</th>
<th>[ 3 ]</th>
<th>[ 4 ]</th>
<th>[ 5 ]</th>
<th>[ 500 ]</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="images/344991607.png" alt="Number 344991607" /></td>
<td><img src="images/775672751.png" alt="Number 775672751" /></td>
<td><img src="images/393802035.png" alt="Number 393802035" /></td>
<td><img src="images/699072358.png" alt="Number 699072358" /></td>
<td><img src="images/493375785.png" alt="Number 493375785" /></td>
<td><img src="images/582739652.png" alt="Number 582739652" /></td>
<td><img src="images/582739652.png" alt="Number 582739652" /></td>
</tr>
</tbody>
</table>

An array of records
Hash Collisions

- Here is another new record to insert.
  - $493375785 \mod 501 = 3$
  - Index 3 has already key and value.

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>[1]</td>
<td>[2]</td>
<td>[3]</td>
<td>[4]</td>
<td>[5]</td>
<td>[500]</td>
</tr>
<tr>
<td>Number 344991607</td>
<td>Number 775672751</td>
<td>Number 393802035</td>
<td>Number 699072358</td>
<td></td>
<td></td>
<td>Number 582739652</td>
</tr>
</tbody>
</table>

An array of records
This is called a **collision**, because there is already another valid record at [3].

An array of records
## Hash Collisions

- Let us make a collision rule:
  - When a collision occurs, move forward until you find an empty spot.

An array of records:

<table>
<thead>
<tr>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
<th>[500]</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Number 344991607" /></td>
<td><img src="image" alt="Number 775672751" /></td>
<td><img src="image" alt="Number 393802035" /></td>
<td><img src="image" alt="Number 699072358" /></td>
<td><img src="image" alt="Number 493375785" /></td>
<td><img src="image" alt="Number 582739652" /></td>
<td></td>
</tr>
</tbody>
</table>
Hash Collisions

- Let us make a collision rule:
  - When a collision occurs, move forward until you find an empty spot.

An array of records
Hash Collisions

- Let us make a collision rule:
  - When a collision occurs, move forward until you find an empty spot.

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number 344991607</td>
<td>Number 775672751</td>
<td>Number 393802035</td>
<td>Number 699072358</td>
<td>Number 493375785</td>
<td>Number 582739652</td>
</tr>
</tbody>
</table>

An array of records
Searching for a Key

- The data (or value) that is attached to a key can be found fairly quickly.

An array of records
Searching for a Key

- The data (or value) that is attached to a key can be found fairly quickly.
  - $493375785 \mod 501 = 3$

An array of records
Searching for a Key

- The data (or value) that is attached to a key can be found fairly quickly.
  - 493375785 mod 501 = 3
  - Follow the collision rule (keep moving forward) until you find the key, or you reach an empty spot.

An array of records

<table>
<thead>
<tr>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
<th>[500]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>344991607</td>
<td>Number</td>
<td>775672751</td>
<td>Number</td>
<td>393802035</td>
<td>Number</td>
</tr>
<tr>
<td>No!</td>
<td>Number</td>
<td>493375785</td>
<td>Number</td>
<td>493375785</td>
<td>Number</td>
<td>582739652</td>
</tr>
</tbody>
</table>
Searching for a Key

- The data (or value) that is attached to a key can be found fairly quickly.
  - $493375785 \mod 501 = 3$
  - Follow the collision rule (keep moving forward) until you find the key, or you reach an empty spot.

An array of records

Number 493375785

Number 344991607
Number 775672751
Number 393802035
Number 699072358
Number 493375785
Number 582739652
Searching for a Key

- The data (or value) that is attached to a key can be found fairly quickly.
  - When the item is found, the information can be copied to the necessary location.

**An array of records**
Deleting a Record

- Records may also be deleted from a hash table.

An array of records

Delete 699072358
Deleting a Record

- Records may also be deleted from a hash table.
- But the location must not be left as an ordinary "empty spot" since that could interfere with searches.
- The location must be marked in some special way so that a search can tell that the spot used to have something in it.

An array of records
Map Methods

- Delegate operations to a list-based map at each cell:

**Algorithm** search\( (k) \):

**Output:** The value associated with the key \( k \) in the map, or **null** if there is no entry with key equal to \( k \) in the map

return \( A[h(k)].search(k) \) \{delegate the search to the list-based map at \( A[h(k)] \}\)

**Algorithm** insert\( (k,v) \):

**Output:** If there is an existing entry in our map with key equal to \( k \), then we return its value (replacing it with \( v \)); otherwise, we return **null**

\[
\begin{align*}
t &= A[h(k)].insert(k,v) \quad \{\text{delegate the put to the list-based map at } A[h(k)]\} \\
\text{if } t = \text{null} \text{ then } & \quad \{k \text{ is a new key}\} \\
& \quad n = n + 1 \\
\text{return } t
\end{align*}
\]

**Algorithm** remove\( (k) \):

**Output:** The (removed) value associated with key \( k \) in the map, or **null** if there is no entry with key equal to \( k \) in the map

\[
\begin{align*}
t &= A[h(k)].remove(k) \quad \{\text{delegate the remove to the list-based map at } A[h(k)]\} \\
\text{if } t \neq \text{null} \text{ then } & \quad \{k \text{ was found}\} \\
& \quad n = n - 1 \\
\text{return } t
\end{align*}
\]
Interview Question

- Implement Hash Map
  - Hash function just returns the remainder when the key is divided by the hash table size.
  - Hash entry (node) has key and value structure.
  - In addition, the class contains search(key) function to access mapped value by key, insert(key,value) function to put key-value pair in table and remove(key) function to remove hash node by key.
  - For collision resolution, separate chaining strategy could be used.
### Implement Simple Hash Map: testHashMap.cpp

```cpp
#include <iostream>

using namespace std;

const int TABLE_SIZE = 128;

// HashEntry Class Declaration
class HashEntry
{
    public:
        int key;
        int value;
        HashEntry(int key, int value)
        {
            this->key = key;
            this->value = value;
        }
};

// HashMap Class Declaration
class HashMap
{
    private:
        HashEntry **table;
    public:
        HashMap()
        {
            table = new HashEntry *[TABLE_SIZE];
            for (int i = 0; i < TABLE_SIZE; i++)
            {
                table[i] = NULL;
            }
        }

    // Hash Function
    int HashFunc(int key)
    {
        return key % TABLE_SIZE;
    }

    // Insert Element at a key
    void Insert(int key, int value)
    {
        int hash = HashFunc(key);
        while (table[hash] != NULL && table[hash]->key != key)
        {
            hash = HashFunc(hash + 1);
        }
        if (table[hash] != NULL)
            delete table[hash];
        table[hash] = new HashEntry(key, value);
    }

    // Search Element at a key
    int Search(int key)
    {
        int hash = HashFunc(key);
        while (table[hash] != NULL && table[hash]->key != key)
        {
            hash = HashFunc(hash + 1);
        }
        if (table[hash] == NULL)
            return -1;
        else
            return table[hash]->value;
    }

    // Remove Element at a key
    void Remove(int key)
    {
        int hash = HashFunc(key);
        while (table[hash] != NULL)
        {
            if (table[hash]->key == key)
                break;
            hash = HashFunc(hash + 1);
        }
        if (table[hash] == NULL)
        {
            cout << "No Element found at key " << key << endl;
            return;
        }
        else
        {
            delete table[hash];
            cout << "Element Deleted" << endl;
        }
    }
};
```
Implement Simple Hash Map: testHashMap.cpp

```cpp
89 ~HashMap()
90 {
91     for (int i = 0; i < TABLE_SIZE; i++)
92         {
93             if (table[i] != NULL)
94                 delete[] table[i];
95             delete[] table;
96         }
97     }
98
99 int main()
100 {
101     HashMap hash;
102     int key, value;
103     int choice;
104     while (1)
105     {
106         switch(choice)
107             {
108                 case 1:
109                     cout<<"Enter element to be inserted: ";
110                     cin>>value;
111                     cout<<"Enter key at which element to be inserted: ";
112                     cin>>key;
113                     hash.Insert(key, value);
114                     break;
115                 case 2:
116                     cout<<"Enter key of the element to be searched: ";
117                     cin>>key;
118                     if (hash.Search(key) == -1)
119                         {
120                             cout<<"No element found at key ", key, "\n";
121                             continue;
122                         }
123                     else
124                         {
125                             cout<<"Element at key ", key, ": ";
126                             cout<<hash.Search(key)<<\n"\n";
127                             break;
128                     case 3:
129                         cout<<"Enter key of the element to be deleted: ";
130                         cin>>key;
131                         hash.Remove(key);
132                         break;
133                     case 4:
134                         exit(1);
135                     default:
136                         cout<<\n"Enter correct option\n";
137                         return 0;
138                     }
139     }
140     return 0;
141 ```
Reading C type declarations

- [http://unixwiz.net/techtips/reading-cdecl.html](http://unixwiz.net/techtips/reading-cdecl.html)
#include <iostream>

using namespace std;

int main () {
    int var;
    int *ptr;
    int **pptr;
    int var = 3000;

    // take the address of var
    ptr = &var;

    // take the address of ptr using address of operator &
    pptr = &ptr;

    // take the value using pptr
    cout << "Value of var :" << var << endl;
    cout << "Value available at *ptr :" << *ptr << endl;
    cout << "Value available at **pptr :" << **pptr << endl;
    return 0;
}
Why do we use double pointers?

If you want to have a list of characters (a word), you can use `char *word`

If you want a list of words (a sentence), you can use `char **sentence`

If you want a list of sentences (a monologue), you can use `char ***monologue`

If you want a list of monologues (a biography), you can use `char ****biography`

If you want a list of biographies (a bio-library), you can use `char *****biolibrary`

If you want a list of bio-libraries (a ??lol), you can use `char ******lol`

... ...

*yes, I know these might not be the best data structures*