Data Structure Review, C++ STL
CISC4080
CIS, Fordham Univ.

Instructor: X. Zhang

This class

- From CISC2200 to C++ STL
- ADT list and C++ STL vector, list
  - Principle of composition: vector of vectors, list of vector, …
- ADT Set and C++ STL’s set, unordered_set
- ADT Dictionary and C++ STL’s ordered_map, unordered_map
- ADT Priority Queue (heap) and C++ priority_queue

Intro. To C++ STL

- C++ Standard Template Library is a set of C++ **template classes** to provide common programming data structures and functions such as lists, stacks, arrays, etc.
- It is a generalized library (its components are parameterized) provides:
  - container classes: list, stack, array, queue, hashtable, BST, …
  - algorithms: swap, sorting, …
  - iterators: allow you to iterates through elements in the container

C++ STL in a nutshell

CISC 2200 Data Structure terminology	C++ STL	Explanation
ADT list Implemented with array, dynamic array, linked list, doubly list, …	**Sequence Containers** Vector (dynamic array) Array (fixed array) deque, forward list	data structures which can be accessed in a sequential manner. Double-ended queues are sequence containers with dynamic sizes that can be expanded or contracted on both ends (either its front or its back).
Queue, stack, heap/ priority queue (list with constrained access)	**Container Adaptors:** queue, priority_queue,stack	provide a different interface for sequential containers, FIFO for queue, LIFO for stack, …
Binary Search Tree	**Associative containers** set, multiset, map, multimap	Ordered data structures that can be quickly **searched** (O(log n) complexity) — searching by key
Hash table	**Unordered associative containers** unordered_set, unordered_multiset, unordered_map, unordered_multimap	implement unordered data structures that can be quickly searched

C++ vector

- A sequence container implemented using a dynamic array based container
  - You can assign a vector to another one, or use copy constructor,
  - Copy constructor: copy from partial vector
- https://www.geeksforgeeks.org/vector-in-cpp-stl/

Case studies: merge sort

```
MergeSort (vector<int> & list)
{
    If (list.size()<=1)
        return;

    int mid = (0+list.size()-1)/2;

    vector<int> leftHalf (list.begin(), list.begin()+mid+1);
    vector<int> rightHalf (list.begin()+mid+1, list.end());
    MergeSort (leftHalf);
    MergeSort (rightHalf);
    MergeSortedVectors (leftHalf, rightHalf, list);
    //to be developed later: merge sorted two halves back to list in
    sorted order
  }
```

What are set, multiset?

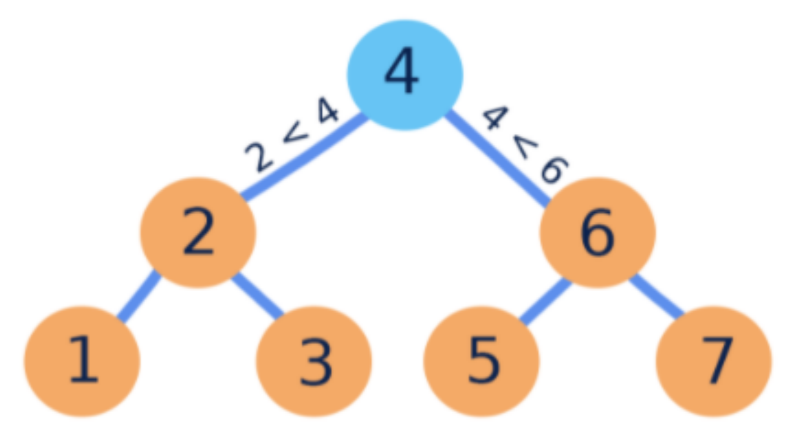
CISC 2200 Data Structure terminology	C++ STL	
ADT list Implemented with array, dynamic array, linked list, doubly list, …	**Sequence Containers** Vector (dynamic array) Array (fixed array) deque, forward list	data structures which can be accessed in a sequential manner. Double-ended queues are sequence containers with dynamic sizes that can be expanded or contracted on both ends (either its front or its back). •
Queue, stack, heap/ priority queue (list with constrained access)	**Container Adaptors:** queue, priority_queue,stack	provide a different interface for sequential containers
Binary Search Tree	**Associative containers** set, multiset, map, multimap	sorted data structures that can be quickly **searched** (O(log n) complexity) — searching by key
Hash table	**Unordered associative containers** unordered_set, unordered_multiset, unordered_map, unordered_multimap	implement unordered data structures that can be quickly searched

Set

- Sets are containers that store unique elements.
- Each element has a value, and each value must be unique.
- Support basic set operations such as: insert(), delete(), find(), …
- Usage: find all unique values in a vector of int, …

Set in C++ STL: BST

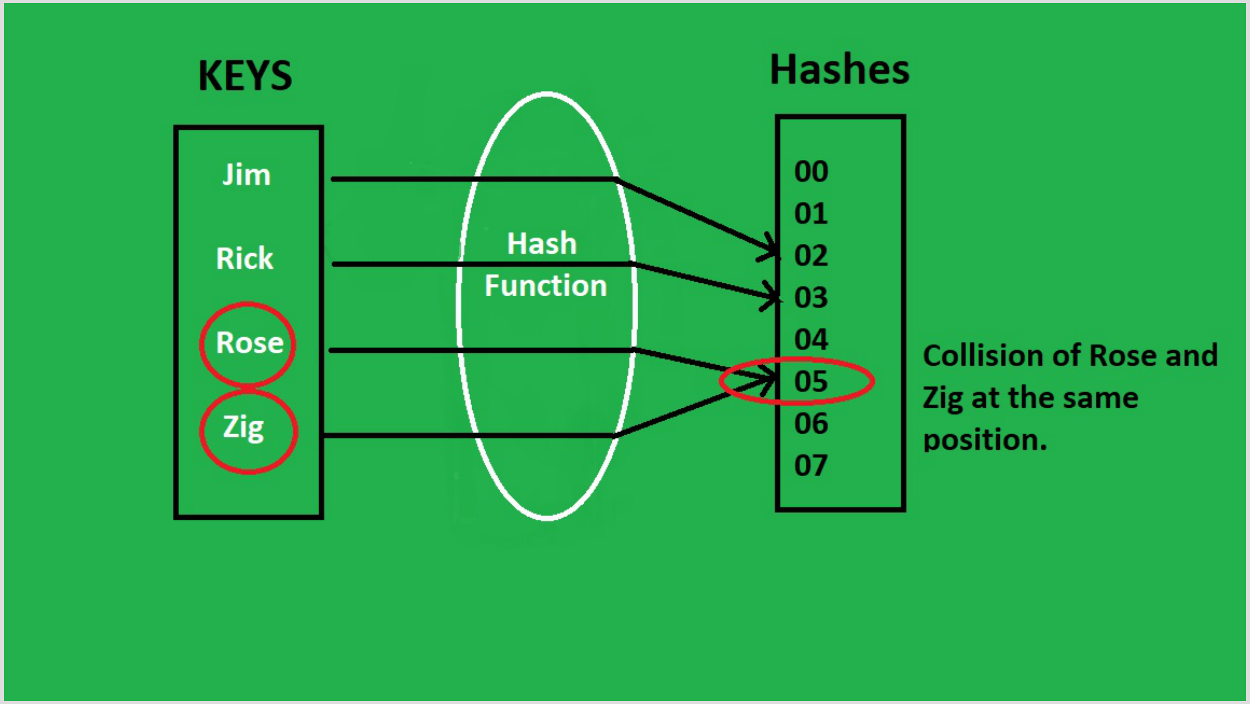
- C++ STL set: implement set container using Binary Search tree
    - => log n time insertion, deletion and searching time



  - a *binary tree* where every node in the left subtree is less than the root, and every node in the right subtree is of a value greater than the root.
  - [Example code]()

Set in C++ STL

- C++ STL **unordered_set** template class: implement set using hash table
  - ”almost” constant insertion, deletion and searching time



- [Code example]
- Discussion: how insertion, deletion and search works?

Multiset

- In mathematics, a **multiset** (or **bag**, or **mset**) is a modification of the concept of a set that, unlike a set, **allows for multiple instances for each of its elements.**
- The number of instances given for each element is called the **multiplicity** of that element in the multiset.

E.g., an infinite number of multisets exist which contain only elements a and b, but vary in the multiplicities of their elements:

- The set $\{a, b\}$ contains only elements a and b, each having multiplicity 1 when $\{a, b\}$ is seen as a multiset.
- In the multiset $\{a, a, b\}$, the element a has multiplicity 2, and b has multiplicity 1.
- In the multiset $\{a, a, a, b, b, b\}$, a and b both have multiplicity 3.

Multiset in C++ STL

- two multiset container class in C++ STL
  - Multiset: implemented using BST
  - unordered_multiset: implemented using hash table
- For more details & sample code
  - Multiset
  - unordered_multiset

What are map, multimap?

CISC 2200 Data Structure terminology	C++ STL	
ADT list Implemented with array, dynamic array, linked list, doubly list, …	**Sequence Containers** Vector (dynamic array) Array (fixed array) deque, forward list	data structures which can be accessed in a sequential manner. Double-ended queues are sequence containers with dynamic sizes that can be expanded or contracted on both ends (either its front or its back). •
Queue, stack, heap/ priority queue (list with constrained access)	**Container Adaptors:** queue, priority_queue,stack	provide a different interface for sequential containers
Binary Search Tree	**Associative containers** set, multiset, map, multimap	sorted data structures that can be quickly **searched** (O(log n) complexity) — searching by key
Hash table	**Unordered associative containers** unordered_set, unordered_multiset, unordered_map, unordered_multimap	implement unordered data structures that can be quickly searched

Map (C++ STL) or Dictionary

- Dictionary (or map in C++ STL) a data structure that stores a collection of (key, value) pairs
  - supporting INSERT, DELETE, SEARCH operations
  - Key are unique
  - Search: look up value associated with a key, i.e., mapping a key to the value associated with the key
- Sometimes called associative array, as it associate a value with a key, and use key as ”index” to the array

```
#include <unordered_map>

#include <fstream>


int main()
{
     unordered_map<string,int> wordsCount;
     char filename[256];
     ifstream input; //declare an ifstream object, which represents a disk file from which
       //we will read info.
     string word;

     cout <<"Enter the file you want to analyze:";
     cin >> filename;

     //Open the disk file
     input.open (filename);

     if (input.is_open())
     {
          //reading from the file is similar to reading from standard input (cin)
          while (input >> word){ //as long as we successfully read a word
               wordsCount[word]++;  //Increment the count for the word

               //when a word is encounted for the first time, wordsCount[word] is
               //accessed for the first time, the value will be initialized to 0
automatically
          } //continue until we reached the end of file

          //Close the file
          input.close();

     } else
     {
          cout <<"Failed to open file " << filename<<endl;
          exit(1);
     }
```

```
//Search a unordered_map
char cont;
do{
    cout <<"Enter a word:";
    cin >> word;
    map<string,int>::iterator it;

    it = wordsCount.find(word);
    if (it==wordsCount.end())
    {
        cout <<" does not appear\n";
        //if accessed (as below), it will be initialized to
        // default value, for int, it's 0
        cout <<"if accessed?"<<wordsCount[word]<<endl;
    }
    else
        cout <<" appears "<<wordsCount[word]<<" times\n";

    cout <<"Continue (y/n)?";
    cin >> cont;
} while (cont=='y');
```

```
    //iterate through a map
    cout <<"Display the words and count\n";
    map<string,int>::iterator it;
    cout <<"word   count\n";
    for (it=wordsCount.begin();it!=wordsCount.end();it++)
    {
        cout <<it->first<<" "<<it->second<<endl;
    }
}
```

BST Implementation of map

- If key type is ordered (i.e., one can compare two given keys, k1, k2), one can use binary search tree
  - each node stores a key, value pair
  - pairs with smaller keys => stored in left subtree
  - pairs with larger keys => stored in right subtree
  - insert O(log n), delete O(log n), search O(log n)

ordered_map

- In C++ STL, ordered_map implements dictionary using BST
  - #include <ordered_map>

  - // wordsCnt is a dictionary/map, key is string, value is int type
  - ordered_map<string, int> wordsCnt;
    - //stores occurrence for each word
  - string word;
  - inputFile>>word;
  - wordsCnt[word]++; //increment occurrence by 1

Hash table based map

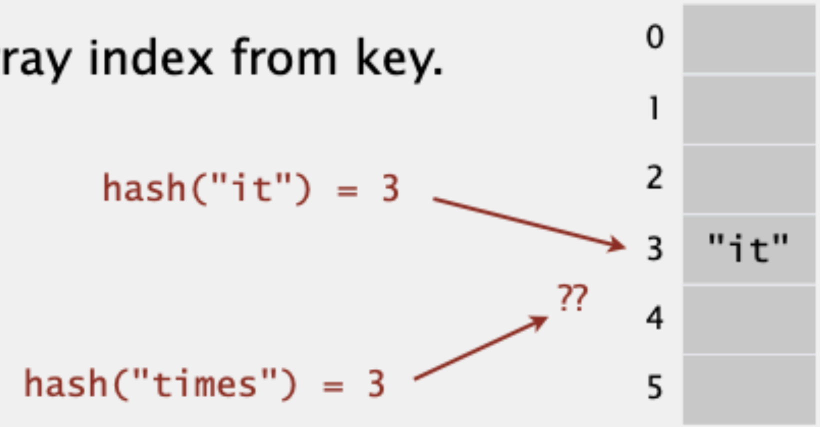
- If key type is not ordered (i.e., one cannot compare two given keys, k1, k2,), one can use hash table
  - insert, delete, search: almost constant time operation

- unordered_map in C++ STL
  - #include <unordered_map> wordsCnt;
  - unordered_map<string, int> wordsCnt; //stores occurrence for each word
  - string word;
  - inputFile>>word;
  - wordsCnt[word]++; //increment occurrence by 1

Hashing: basic plan

Save items in a key-indexed table (index is a function of the key).

Hash function. Method for computing array index from key.



Issues.

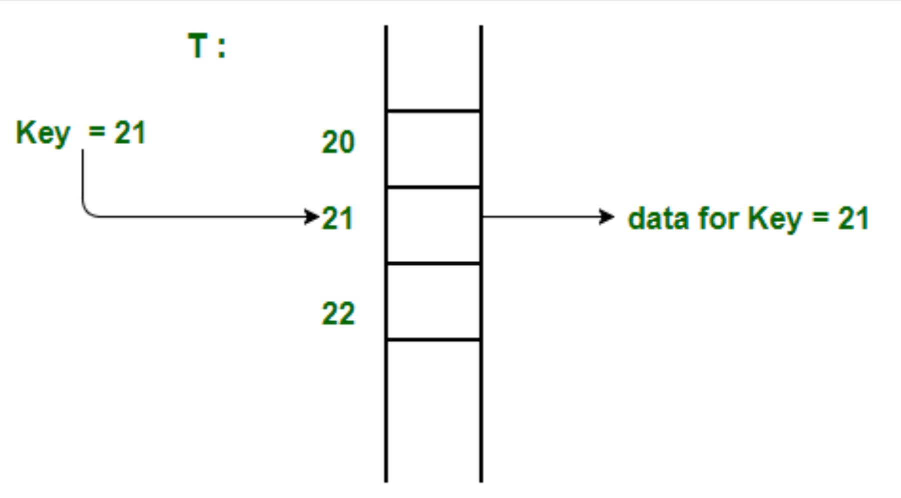
- Computing the hash function.
- Equality test: Method for checking whether two keys are equal.
- Collision resolution: Algorithm and data structure to handle two keys that hash to the same array index.

Classic space-time tradeoff.

- No space limitation: trivial hash function with key as index.
- No time limitation: trivial collision resolution with sequential search.
- Space and time limitations: hashing (the real world).

Direct Address Table

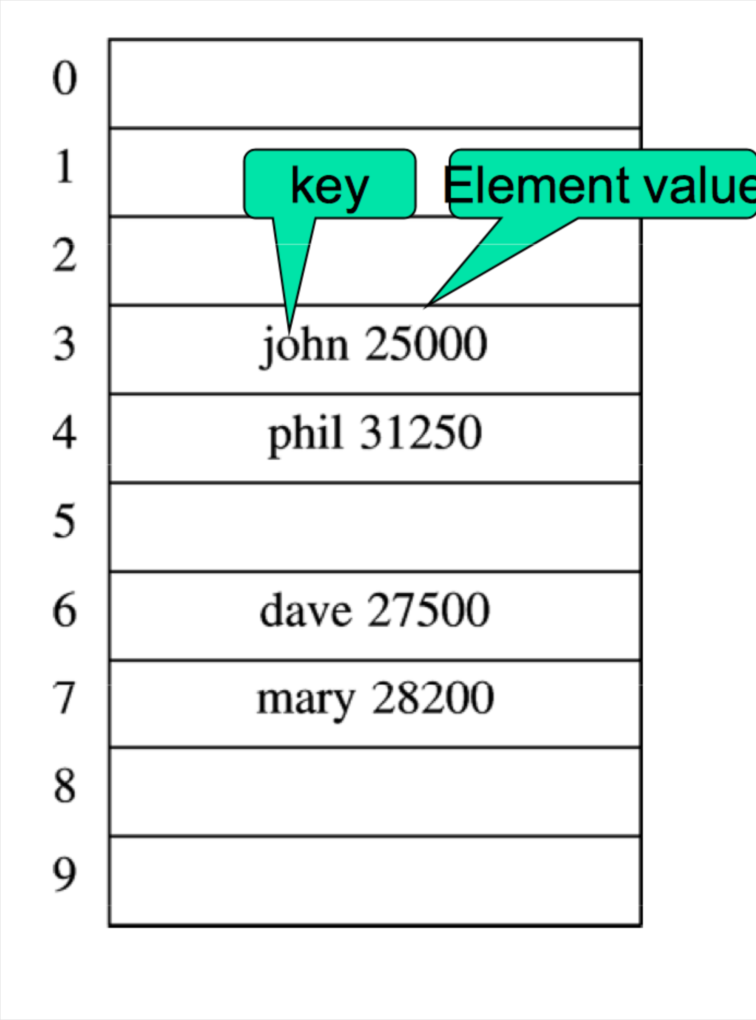
- Direct address table: <u>use key as index</u> into the array
  - only applicable when key is integer type
  - If T is the array, then T[k] stores the element whose key is k



- Limitations:
  - key has to be integer type
  - table/array needs to be big enough to have one slot for every possible key

HashTable Operations

- **Insert a new key value pair:**
  - Table[h("john")]=Element("John", 25000)
- **Delete element by key**
  - Table[h("john")]=NULL
- Search by key
  - return Table[h("dave")]
- Assuming running time of h() is constant, all above operations takes O(1) time

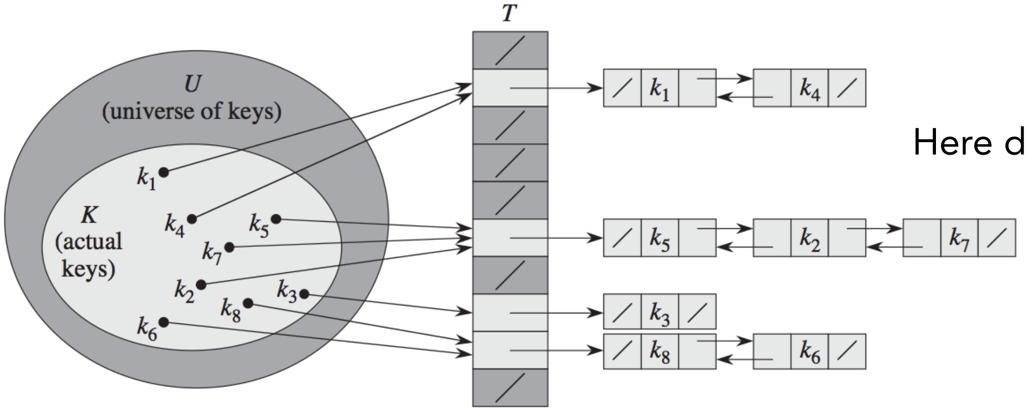


Collision Resolution

- Recall that h(.) is not one-to-one, so it maps multiple keys to same slot:
  - for distinct k1, k2, h(k1)=h(k2) => collision
- Two different ways to resolve collision
  - Chaining: store colliding keys in a linked list (bucket) at the hash table slot
    - dynamic memory allocation, storing pointers (overhead)
  - Open addressing: if slot is taken, try another, and another (a probing sequence)
    - clustering problem.

Chaining

- Chaining: store colliding elements in a linked list at the same hash table slot
    - if all keys are hashed to same slot, hash table degenerates to a linked list.



Here doubly-linked list is used

Figure 11.3 Collision resolution by chaining. Each hash-table slot $T[j]$ contains a linked list of all the keys whose hash value is j. For example, $h(k_1) = h(k_4)$ and $h(k_5) = h(k_7) = h(k_2)$. The linked list can be either singly or doubly linked; we show it as doubly linked because deletion is faster that way.

Chaining: operations

- Insert (x):
  - insert x at the head of T[h(x.key)]
  - Running time (worst and best case): O(1)
- Search (k)
  - search for an element with key x in list T[h(k)]
- Delete (x)
  - Delete x from the list T[h(x.key)]
- Running time of search and delete: proportional to length of list stored in h(x.key)

Chaining: analysis

- Consider a hash table T with m slots stores n elements.
    - load factor
- **Ideal case:** any given element is equally likely to hash into any of the m slots, independently of where any other element is hashed to
    - average length of lists is
    - search and delete takes
- Worst case: If all keys are hashed to same slot, hash table degenerates to a linked list
    - search and delete takes

Collision Resolution

- Open addressing: store colliding elements elsewhere in the table
  - Advantage: no need for dynamic allocation, no need to store pointers
- When inserting:
  - examine (probe) a sequence of positions in hash table until find empty slot
- When searching/deleting:
  - examine (probe) a sequence of positions in hash table until find element

Open Addressing

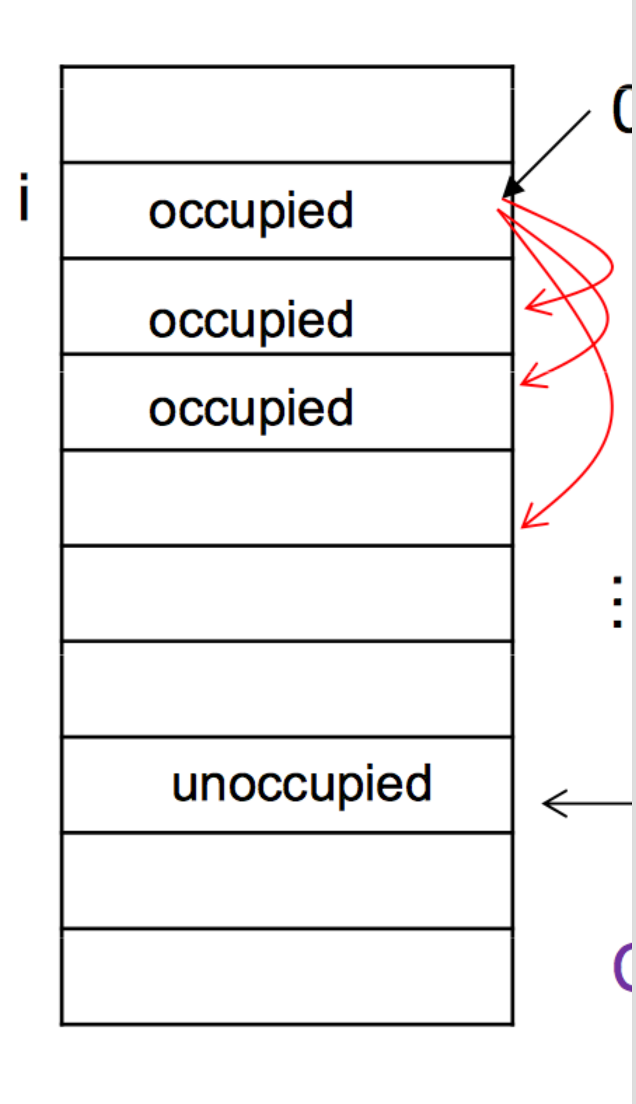
- Hash function: extended to probe sequence (m functions):

$$h_i(x), i = 0, 1, ..., m - 1$$

$$h_i(x) \neq h_j(x),\ \text{for } i \neq j$$

- **insert**: if $h_0(k)$ is taken, try $h_1(k)$, and then $h_2(k)$, until find an empty slot
- **Search** for key k: if element at $h_0(k)$ is not a match, try $h_1(k)$, and then $h_2(k)$, ..until find matching element, or reach an empty slot
- **Delete** key k: first search for k, then mark its slot as DELETED

Linear Probing



- Probing sequence

  $h_i(x)=(h(x)+i) \bmod m$

  - try following indices in sequence
    - $h(x) \bmod m$,
    - $(h(x)+1) \bmod m$,
    - $(h(x)+2) \bmod m$, …
  - Continue until an empty slot is found
- Problem: primary clustering
  - if there are multiple keys mapped to a slot, the slots after it tends to be occupied

Quadratic Probing

$$h_i(x) = (h(x) + c_1 i + c_2 i^2) \bmod m$$

- probe sequence:
  - $h_0(x)=h(x)$ mod m
  - $h_1(x)=(h(x)+c_1+c_2)$ mod m
  - $h_2(x)=(h(x)+2c_1+4c_2)$ mod m
  - …
- Problem:
  - secondary clustering
  - choose c_1,c_2,m carefully so that all slots are probed