CS 32 Week 9
Discussion 1I

Srinath

Outline

- Hash Tables
- Worksheet 9

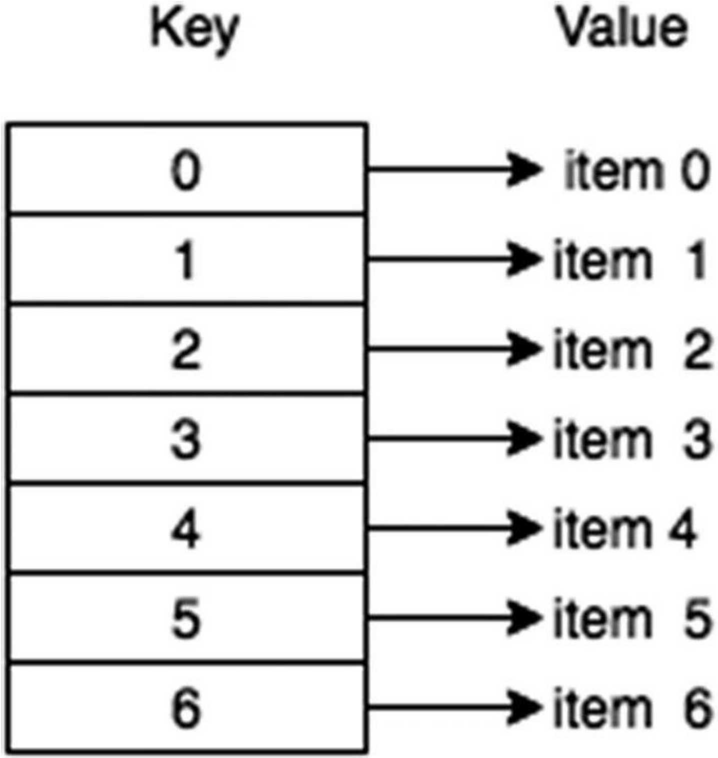
Hash Tables

Hash Tables :

A data structure to store key-value pairs.
Something like a dictionary.

Simplest

- may be an array of strings.
- array index is key



Key	Value
0	item 0
1	item 1
2	item 2
3	item 3
4	item 4
5	item 5
6	item 6

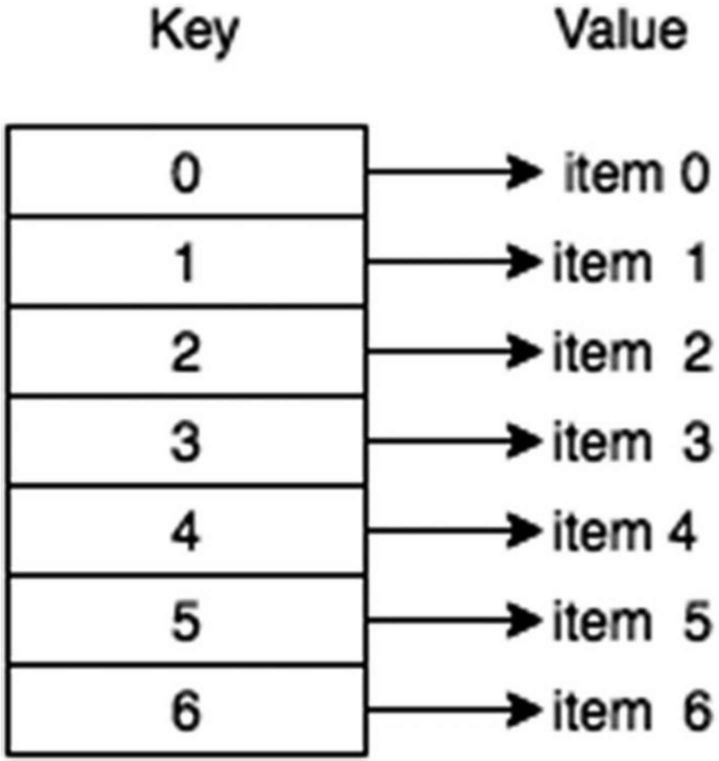
Hash Tables :

A data structure to store key-value pairs.
Something like a dictionary.

Simplest

- may be an array of strings.
- array index is key

What if keys are not in array index range?



Key	Value
0	item 0
1	item 1
2	item 2
3	item 3
4	item 4
5	item 5
6	item 6

Hash Tables :

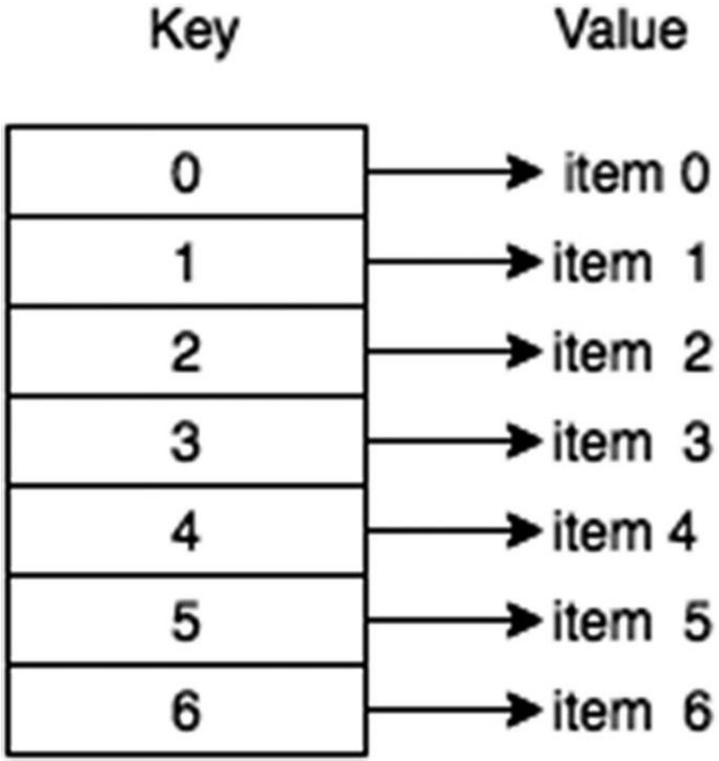
A data structure to store key-value pairs.
Something like a dictionary.

Simplest

- may be an array of strings.
- array index is key

What if keys are not in array index range?

- Need a way to map them to array-index (hash function)



Hash Tables : The Hash Function

A mapping from a given key to an index

```
int hash(int x){
        return x % 7;
}
```

Hash Tables : The Hash Function

A mapping from a given key to an index

```
int hash(int x){
        return x % 7;
}
```

What if our keys are strings instead?

Hash Tables : The Hash Function

A mapping from a given key to an index

```
int hash(int x){
        return x % 7;
}
```

What if our keys are strings instead?

```
int hash(string s){
        sm = 0;
        for(int i=0; i<s.length(); i++)
                sm += s[i]-'0';

        return sm%7;
}
```

Hash Tables : The Hash Function

A mapping from a given key to an index

```
int hash(int x){
        return x % 7;
}
```

What if our keys are strings instead?

- Create data structure to store Key-Value pair
- Store it in the array at index returned by hash function.

```
int hash(string s){
        sm = 0;
        for(int i=0; i<s.length(); i++)
                sm += s[i]-'0';

        return sm%7;
}
```

Hash Tables : The Hash Function

A mapping from a given key to an index

```
int hash(int x){
        return x % 7;
}
```

What if our keys are strings instead?

- Create data structure to store Key-Value pair
- Store it in the array at index returned by hash function.

Consider integer keys 23, 65

Their hashes?? -

```
int hash(string s){
        sm = 0;
        for(int i=0; i<s.length(); i++)
                sm += s[i]-’0’;

        return sm%7;
}
```

Hash Tables : The Hash Function

A mapping from a given key to an index

```
int hash(int x){
        return x % 7;
}
```

What if our keys are strings instead?

- Create data structure to store Key-Value pair
- Store it in the array at index returned by hash function.

Consider integer keys 23, 65
Their hashes?? - 2, 2   Oh No! we have a collision

```
int hash(string s){
        sm = 0;
        for(int i=0; i<s.length(); i++)
                sm += s[i]-'0';

        return sm%7;
}
```

Hash Tables : The Hash Function

Collision is when two keys map to the same index.

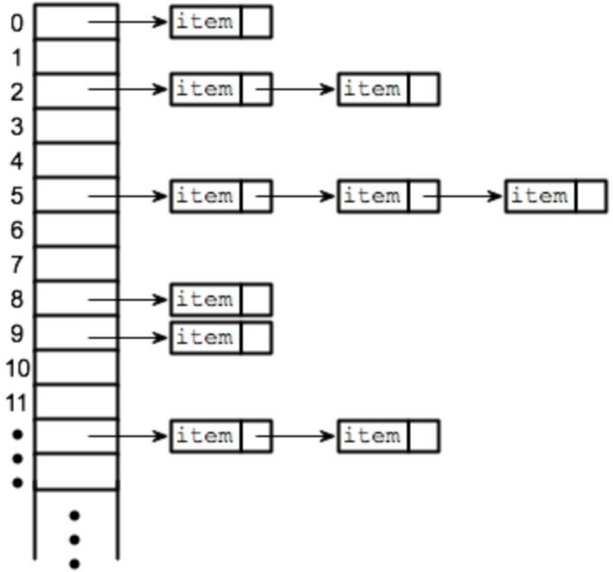
How to resolve a collision?

Hash Tables : The Hash Function

Collision is when two keys map to the same index.

How to resolve a collision?

- Store a list at that index instead of a single element.



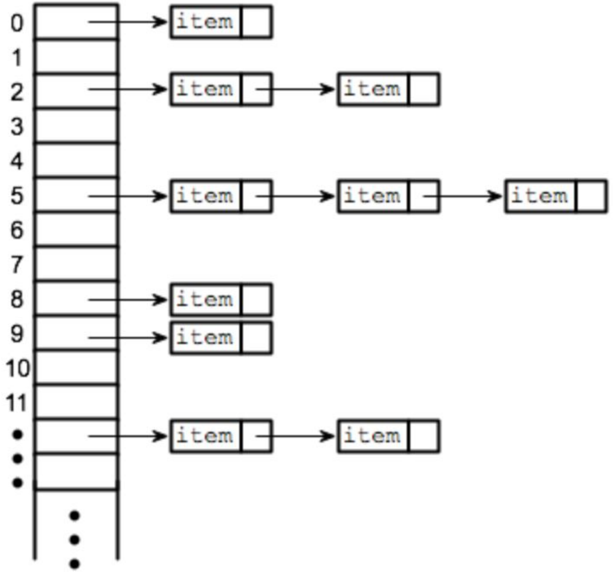
Hash Tables : The Hash Function

Collision is when two keys map to the same index.

How to resolve a collision?

- Store a list at that index instead of a single element.

Assume we have 10000 buckets, only 100 items, can we have a collision?



Hash Tables : The Hash Function

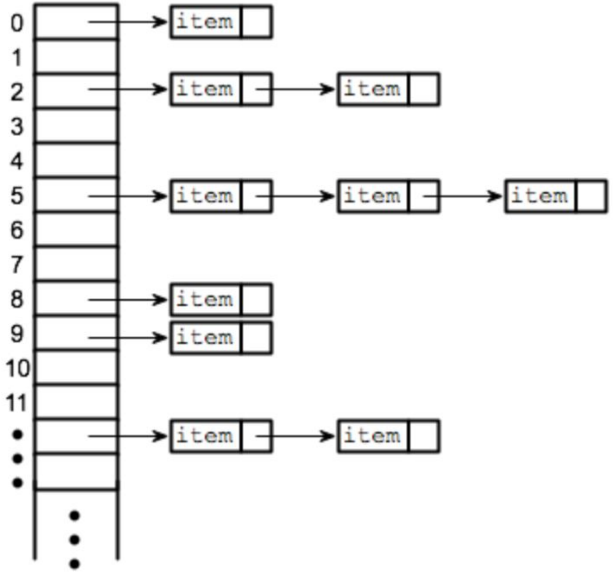
Collision is when two keys map to the same index.

How to resolve a collision?

- Store a list at that index instead of a single element.

Assume we have 10000 buckets, only 100 items, can we have a collision?

- Yes, possible. Two keys can still map to same index.



Hash Tables : The Hash Function

What is a good hash function?

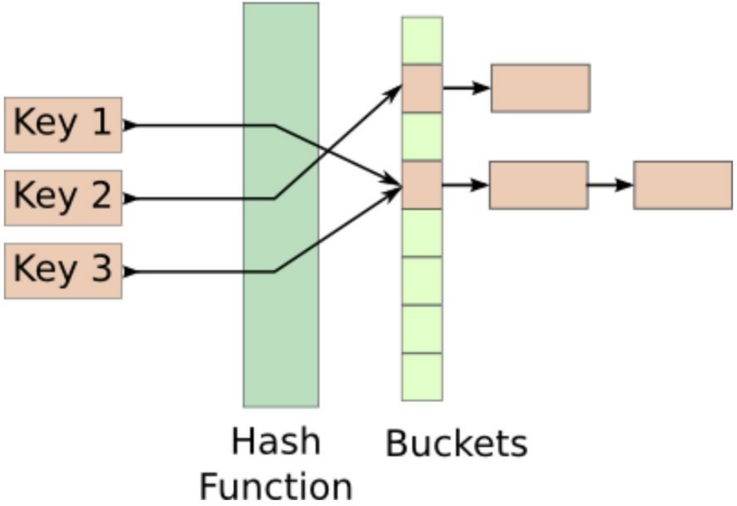


Image: https://vhanda.in/blog

Hash Tables : The Hash Function

What is a good hash function?

- Something which generates a uniform distribution over the buckets.
- Keep in mind, what input(key) distribution you have and what the corresponding output distribution that is generated.

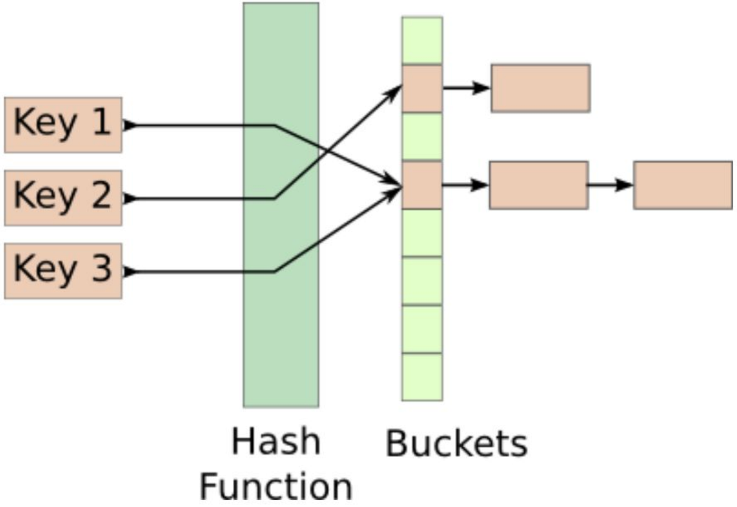


Image: https://vhanda.in/blog

Hash Tables : Complexity

Load Factor = #input size(N) / #buckets

If the hash distribution is uniform, represents how many elements each bucket holds on average

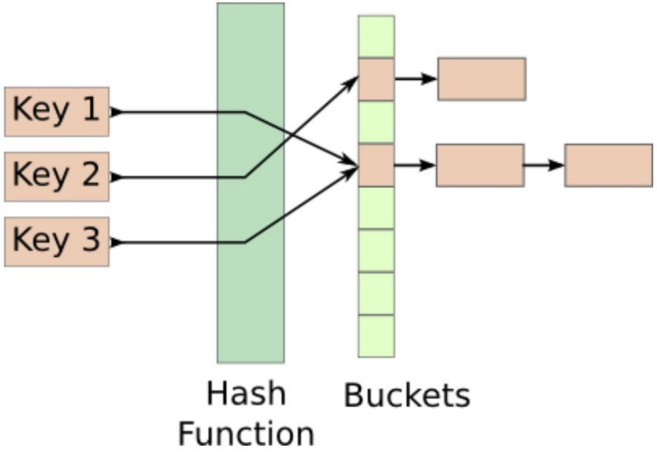


Image: https://vhanda.in/blog

Hash Tables : Complexity

Load Factor = #input size(N) / #buckets

If the hash distribution is uniform, represents how many elements each bucket holds on average

What is the average case time complexity?

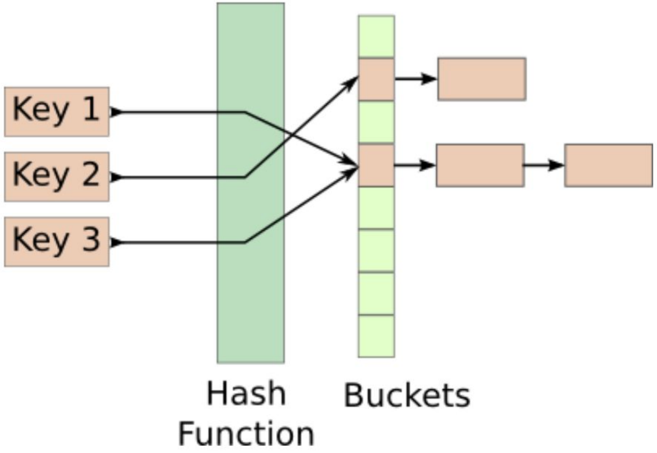


Image: https://vhanda.in/blog

Hash Tables : Complexity

Load Factor = #input size(N) / #buckets

If the hash distribution is uniform, represents how many elements each bucket holds on average

What is the average case time complexity?

- O(Load Factor)

What is the worst case time complexity?

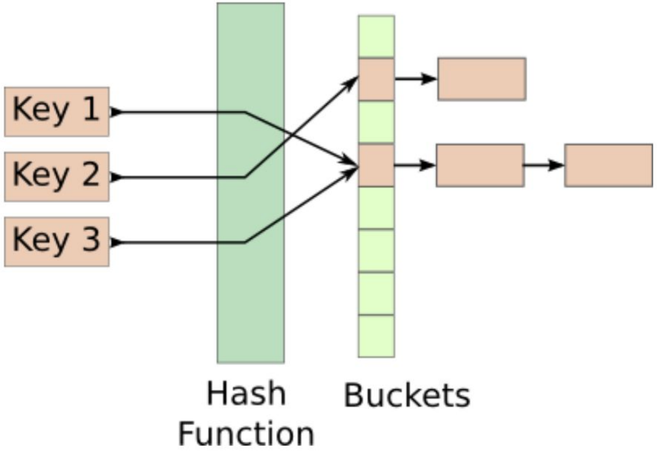


Image: https://vhanda.in/blog

Hash Tables : Complexity

Load Factor = #input size(N) / #buckets

If the hash distribution is uniform, represents how many elements each bucket holds on average

What is the average case time complexity?

- O(Load Factor)

What is the worst case time complexity?

- O(N), when all keys map to same hash value

If we have ~O(N) buckets, then Load Factor ~ O(1), therefore average case complexity is ~ O(1)

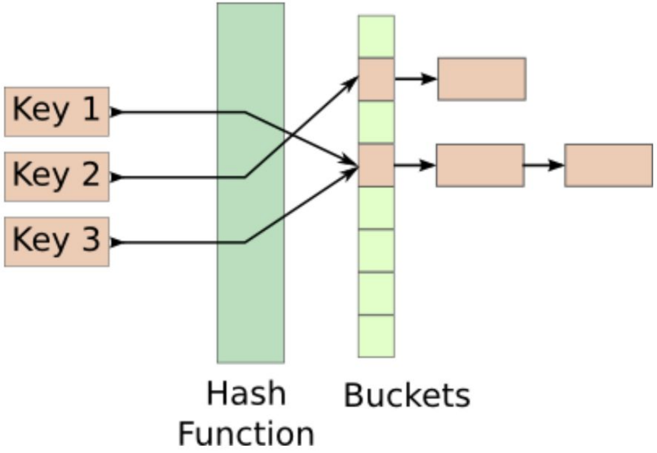


Image: https://vhanda.in/blog

Hash Tables : Complexity

Load Factor = #input size(N) / #buckets

For fixed number of buckets, time complexity grows linearly with input size N. i,e O(N)

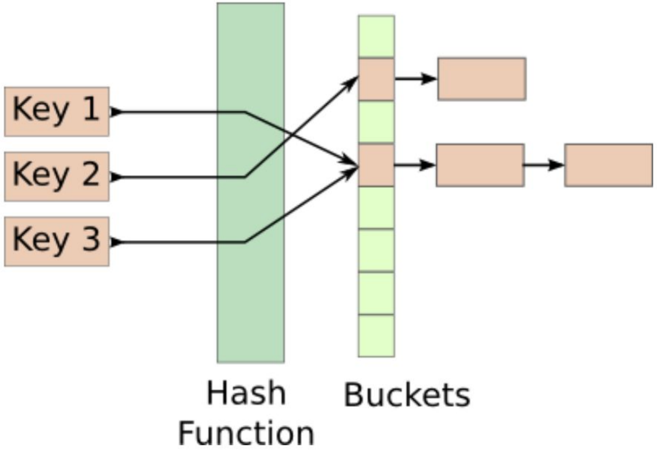


Image: https://vhanda.in/blog

Hash Tables : Complexity

Load Factor = #input size(N) / #buckets

For fixed number of buckets, time complexity grows linearly with input size N. i,e O(N)

Can we do better?

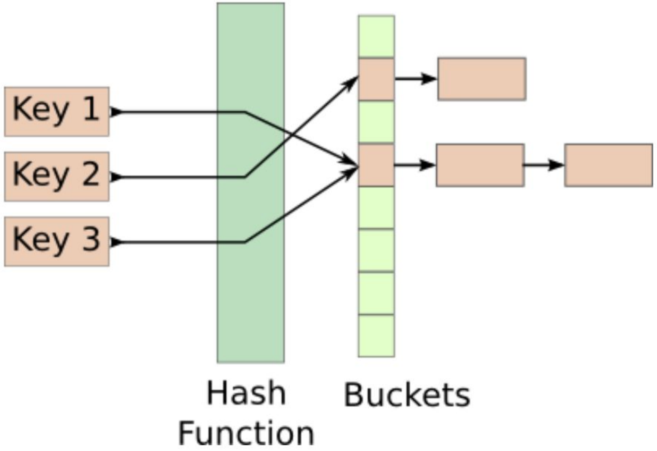


Image: https://vhanda.in/blog

Hash Tables : Complexity

Load Factor = #input size(N) / #buckets

For fixed number of buckets, time complexity grows linearly with input size N. i,e O(N)

Can we do better?

- Yes, increase number of buckets dynamically.

Dynamic hashing

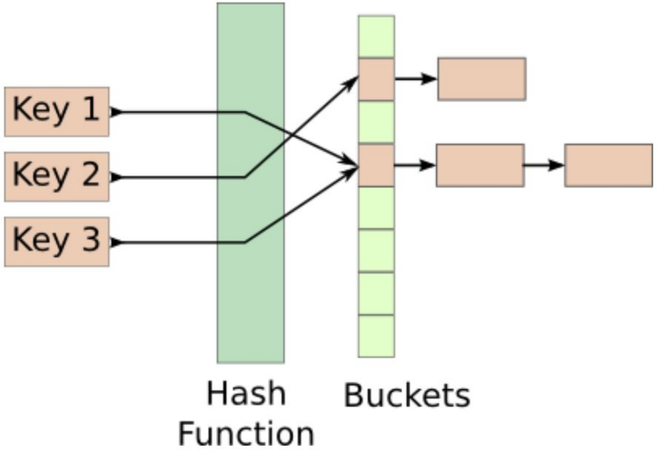


Image: https://vhanda.in/blog

Hash Tables : Complexity

Load Factor = #input size(N) / #buckets

For fixed number of buckets, time complexity grows linearly with input size N. i,e O(N)

Can we do better?

- Yes, increase number of buckets dynamically.

Dynamic hashing

- When load factor reaches a limit, create a new hash table with 2x size, move values from old to new(**re-hashing**)

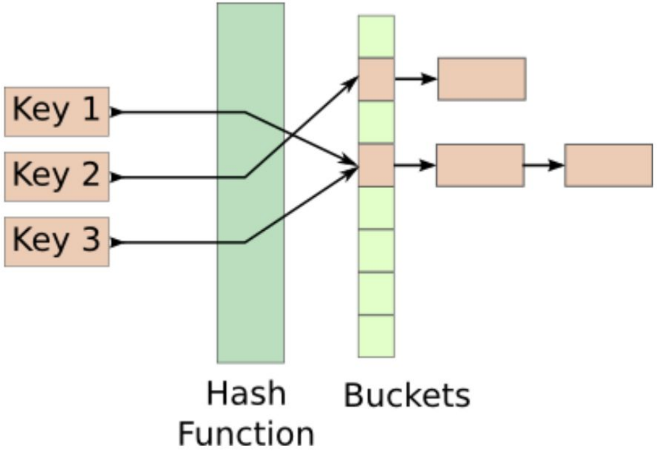


Image: https://vhanda.in/blog

Hash Tables : Complexity

Load Factor = #input size(N) / #buckets

For fixed number of buckets, time complexity grows linearly with input size N. i,e O(N)

Can we do better?

- Yes, increase number of buckets dynamically.

Dynamic hashing

- When load factor reaches a limit, create a new hash table with 2x size, move values from old to new(**re-hashing**)
- Drawbacks?

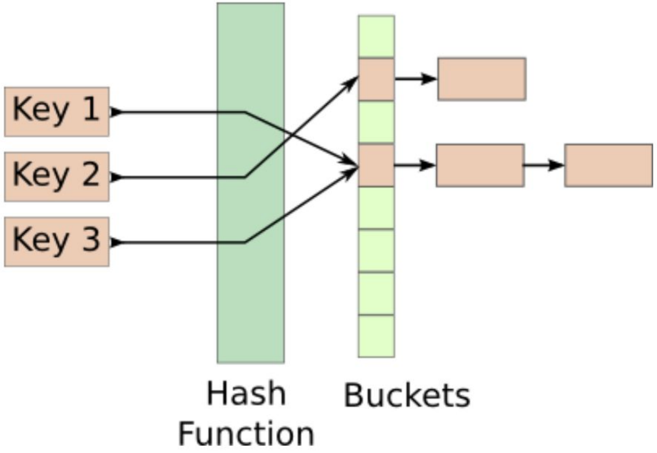


Image: https://vhanda.in/blog

Hash Tables : Complexity

Load Factor = #input size(N) / #buckets

For fixed number of buckets, time complexity grows linearly with input size N. i,e O(N)

Can we do better?

- Yes, increase number of buckets dynamically.

Dynamic hashing

- When load factor reaches a limit, create a new hash table with 2x size, move values from old to new(**re-hashing**)
- Drawbacks?
    - Copying takes O(N) time, inconsistent performance.

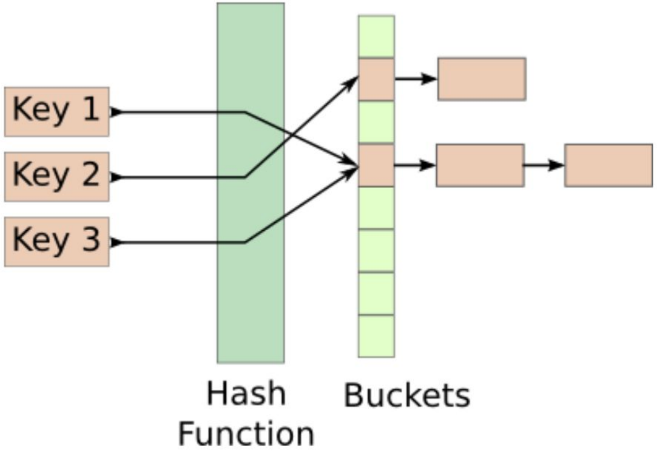


Image: https://vhanda.in/blog

Hash Tables : Complexity

Load Factor = #input size(N) / #buckets

For fixed number of buckets, time complexity grows linearly with input size N.

Can we do better?

- Yes, increase number of buckets dynamically.

Dynamic hashing (Any better ideas?)

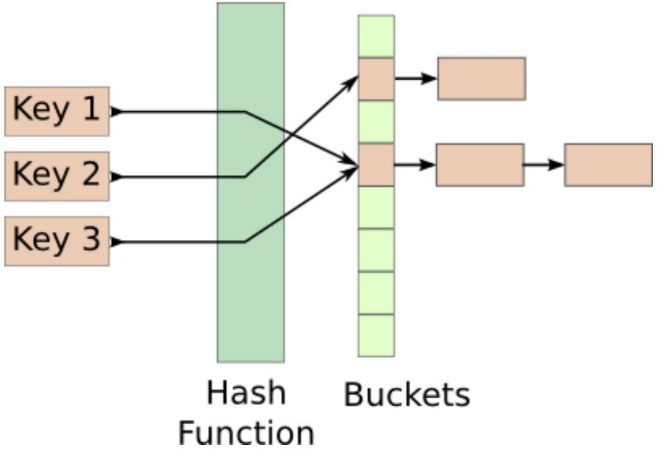


Image: https://vhanda.in/blog

Hash Tables : Complexity

Load Factor = #input size(N) / #buckets

For fixed number of buckets, time complexity grows linearly with input size N.

Can we do better?

- Yes, increase number of buckets dynamically.

Dynamic hashing (Any better ideas?)

- Create a new Hash Table with bigger size, but don’t copy values from old to new.
- Handle copy along with insertion i,e distribute the copying across all new insertions possible.
- Eg: If we want to increase #buckets by 1.5x, along with each new insertion, move 2 more key-values from the old table.

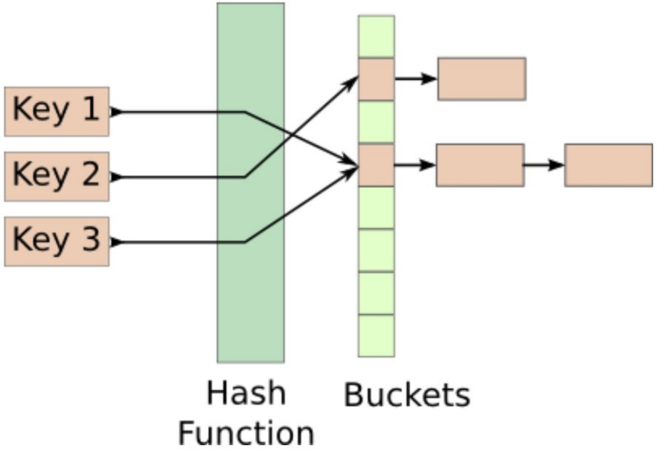


Image: https://vhanda.in/blog

Hash Tables : Complexity Summary

Operations on hash tables with a fixed number of buckets are O(N).

Operations on a hash table with a fixed maximum load factor (so it grows the number of buckets if necessary)

- O(1) on average if a full rehash is done all at once,
- O(1) always if re-hashing is done incrementally. (This assumes a good hash function that produces reasonably uniformly distributed output values.)

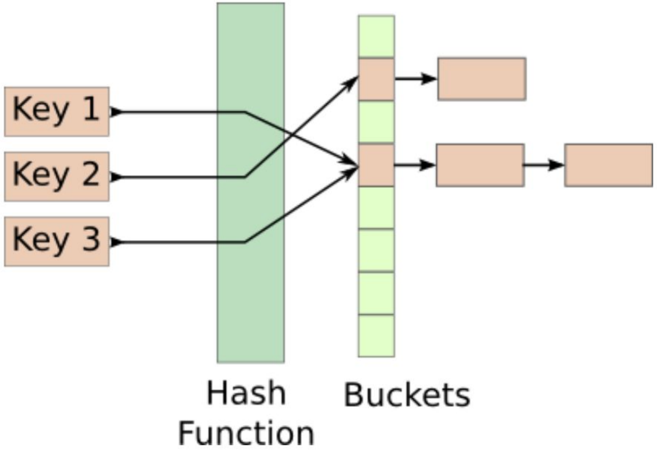


Image: https://vhanda.in/blog

Hash Tables : Implementation

```
class HashTable{
        public :
             HashTable();
             bool find(string key);
             int get(string key);
             bool insert(string key, int value);
             bool delete(string key);
       private:
            int hash(string key);
            vector<list<Node>> m_array;
            int arr_size;
            int m_size;
};

struct Node{
            string key;
            int value;
};
```

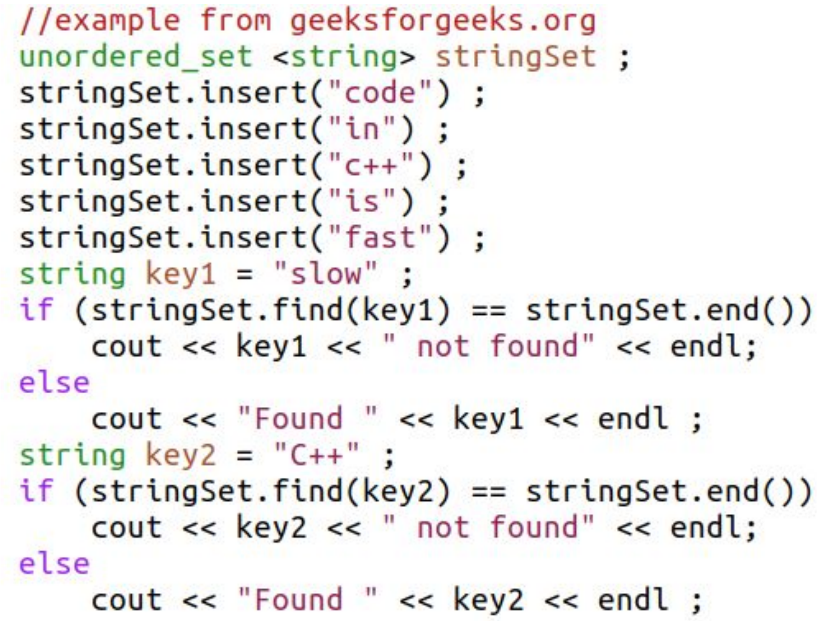
Hash Tables : STL

unordered_set, unordered_map

- Uses hash table
- Unsorted
- No duplicates allowed
- Average case O(1)
- Worst case O(N)

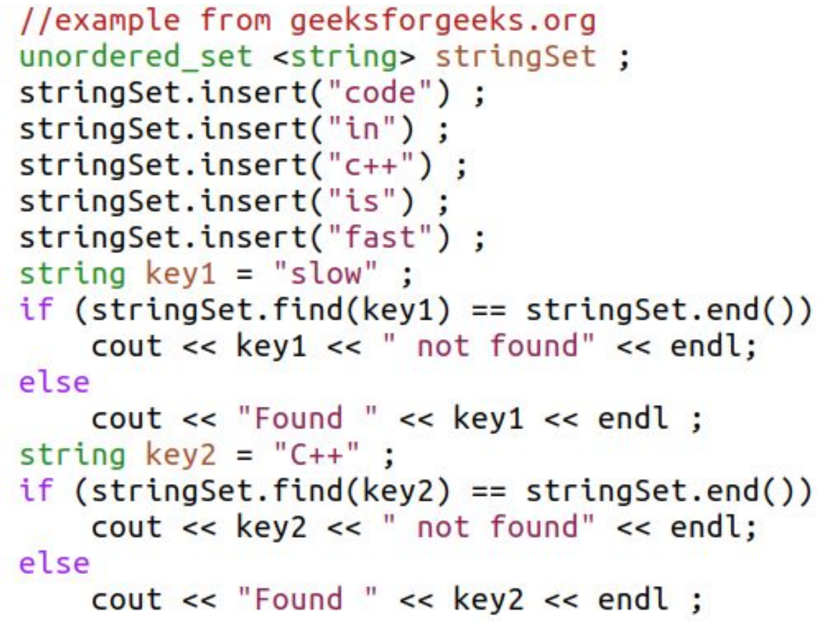
Reference
https://www.cplusplus.com/reference/unordered_set/unordered_set/
https://www.cplusplus.com/reference/unordered_map/unordered_map/

Hash Tables : Examples



```
//example from geeksforgeeks.org
unordered_set <string> stringSet ;
stringSet.insert("code") ;
stringSet.insert("in") ;
stringSet.insert("c++") ;
stringSet.insert("is") ;
stringSet.insert("fast") ;
string key1 = "slow" ;
if (stringSet.find(key1) == stringSet.end())
    cout << key1 << " not found" << endl;
else
    cout << "Found " << key1 << endl ;
string key2 = "C++" ;
if (stringSet.find(key2) == stringSet.end())
    cout << key2 << " not found" << endl;
else
    cout << "Found " << key2 << endl ;
```

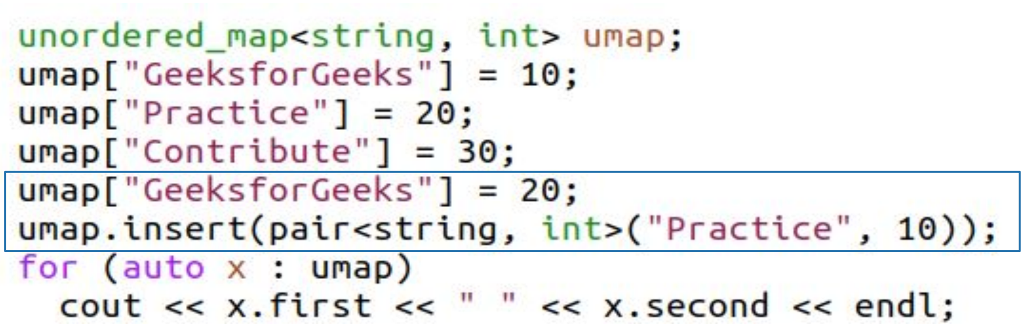
Hash Tables : Examples



```
//example from geeksforgeeks.org
unordered_set <string> stringSet ;
stringSet.insert("code") ;
stringSet.insert("in") ;
stringSet.insert("c++") ;
stringSet.insert("is") ;
stringSet.insert("fast") ;
string key1 = "slow" ;
if (stringSet.find(key1) == stringSet.end())
    cout << key1 << " not found" << endl;
else
    cout << "Found " << key1 << endl ;
string key2 = "C++" ;
if (stringSet.find(key2) == stringSet.end())
    cout << key2 << " not found" << endl;
else
    cout << "Found " << key2 << endl ;
```

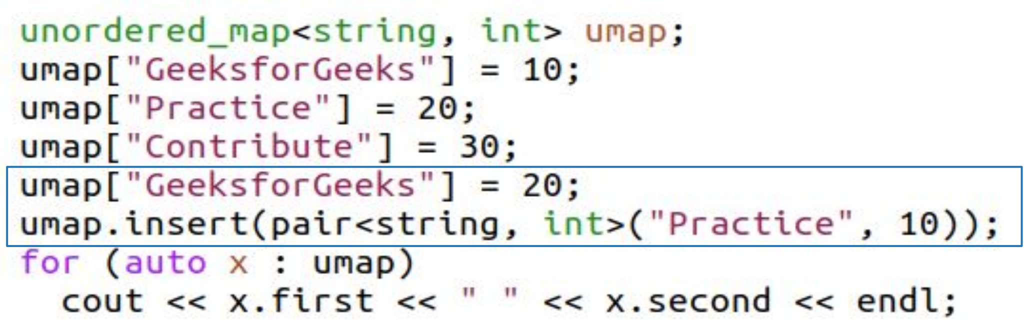
```
slow not found
C++ not found
```

Hash Tables : Examples



```
unordered_map<string, int> umap;
umap["GeeksforGeeks"] = 10;
umap["Practice"] = 20;
umap["Contribute"] = 30;
umap["GeeksforGeeks"] = 20;
umap.insert(pair<string, int>("Practice", 10));
for (auto x : umap)
  cout << x.first << " " << x.second << endl;
```

Hash Tables : Examples



```
unordered_map<string, int> umap;
umap["GeeksforGeeks"] = 10;
umap["Practice"] = 20;
umap["Contribute"] = 30;
umap["GeeksforGeeks"] = 20;
umap.insert(pair<string, int>("Practice", 10));
for (auto x : umap)
  cout << x.first << " " << x.second << endl;
```

```
GeeksforGeeks 20
Practice 20
Contribute 30
```