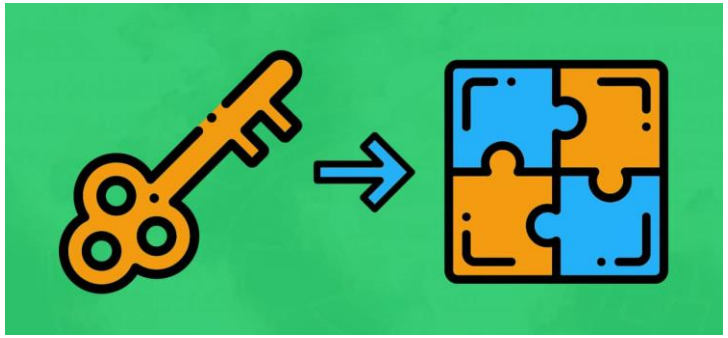
Data Structures and Algorithm

Moazzam Ali Sahi

Lecture # 27-28

Hashing



Last Lecture

- Examine and implement various graph traversal algorithms
- Learn how to implement a shortest path algorithm
- Dijkstra's Algorithm

This Lecture

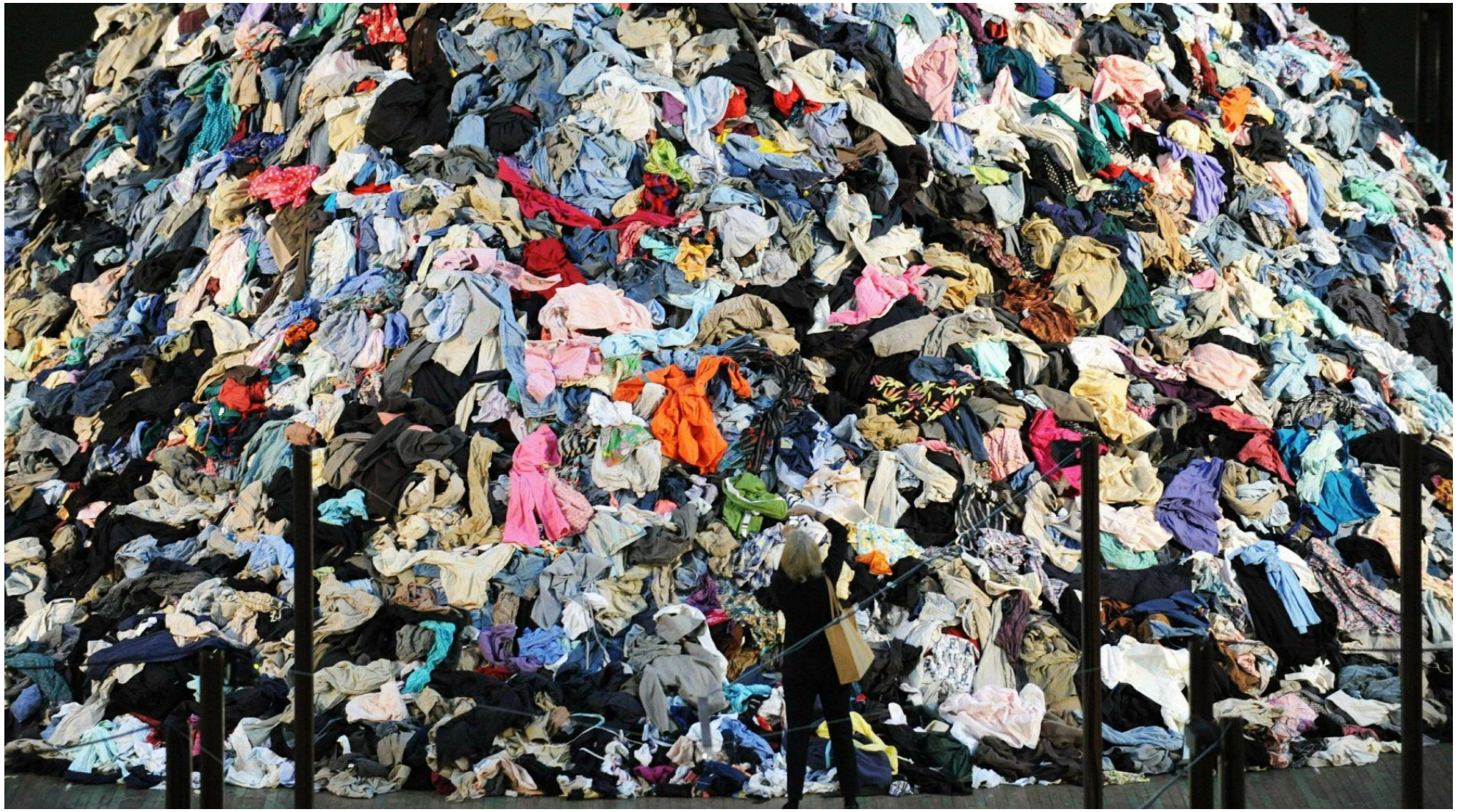
- What is Hashing
- Insertion , Deletion and Searching in Hash Tables
- Collisions and their solutions

Hashing

- Two search algorithms:
  - sequential and
  - binary.
- In a binary search, the data must be sorted;
- in a sequential search, the data does not need to be in any particular order.
- Sequential search is of order n, and a binary search is of order $\log_2 n$, where n is the length of the list.
- The obvious question is: Can we construct a search algorithm that is of order less than $\log_2 n$?
- Recall that both search algorithms, sequential and binary, are comparison-based algorithms.
- We obtained a lower bound on comparison-based search algorithms, which shows that comparison-based search algorithms are at least of order $\log_2 n$.
- Therefore, if we want to construct a search algorithm that is of order less than $\log_2 n$, it cannot be comparison based.
- We want an algorithm that, on average, is of order 1.

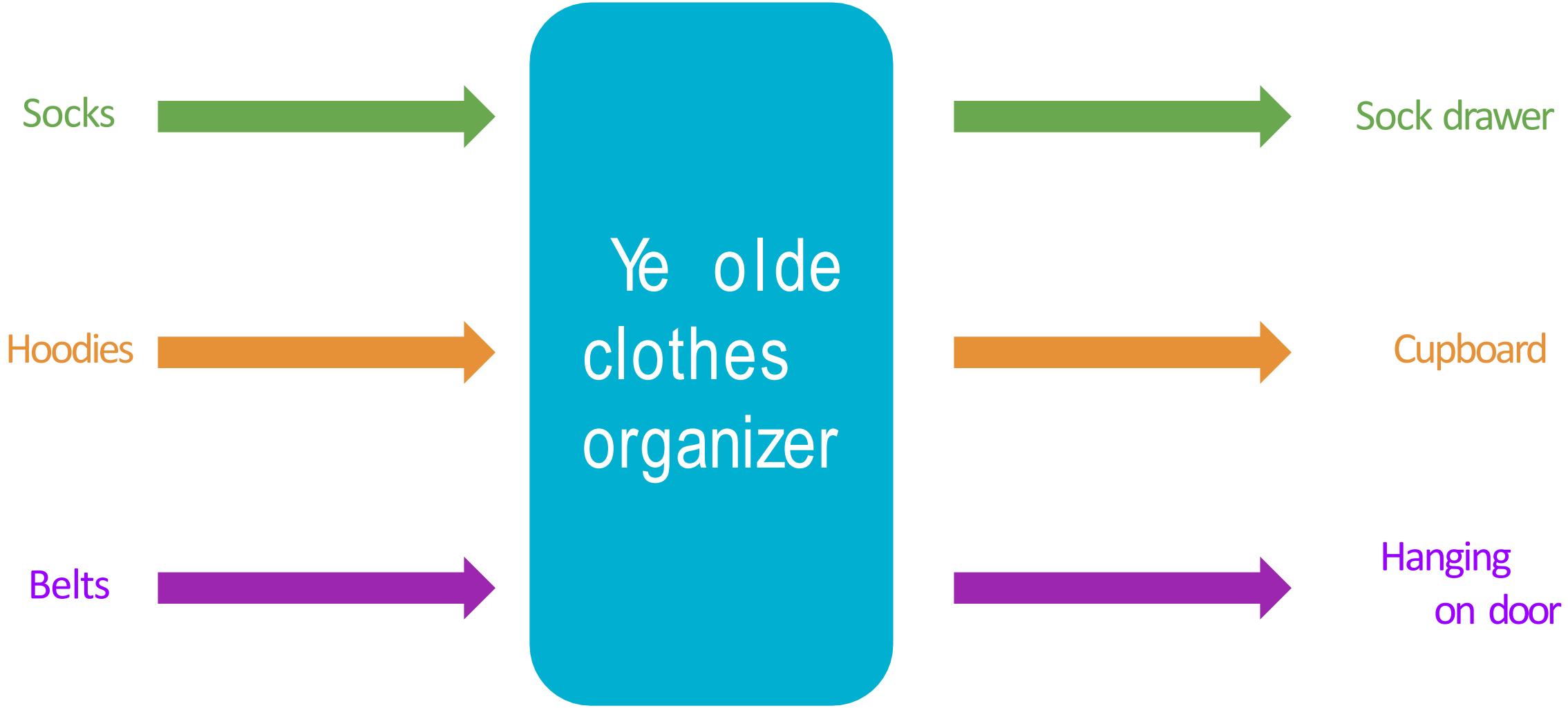
My laundry

My laundry on steroids



My laundry if I were a functional adult

A mapping from clothes to storage locations



How does this mapping help?

I can go directly to where the clothes would be

Lookup is improved

Insertion is improved

Removal is improved

Assuming I have N clothes, operations go from O(N) to O(1) I i

How does this mapping help?

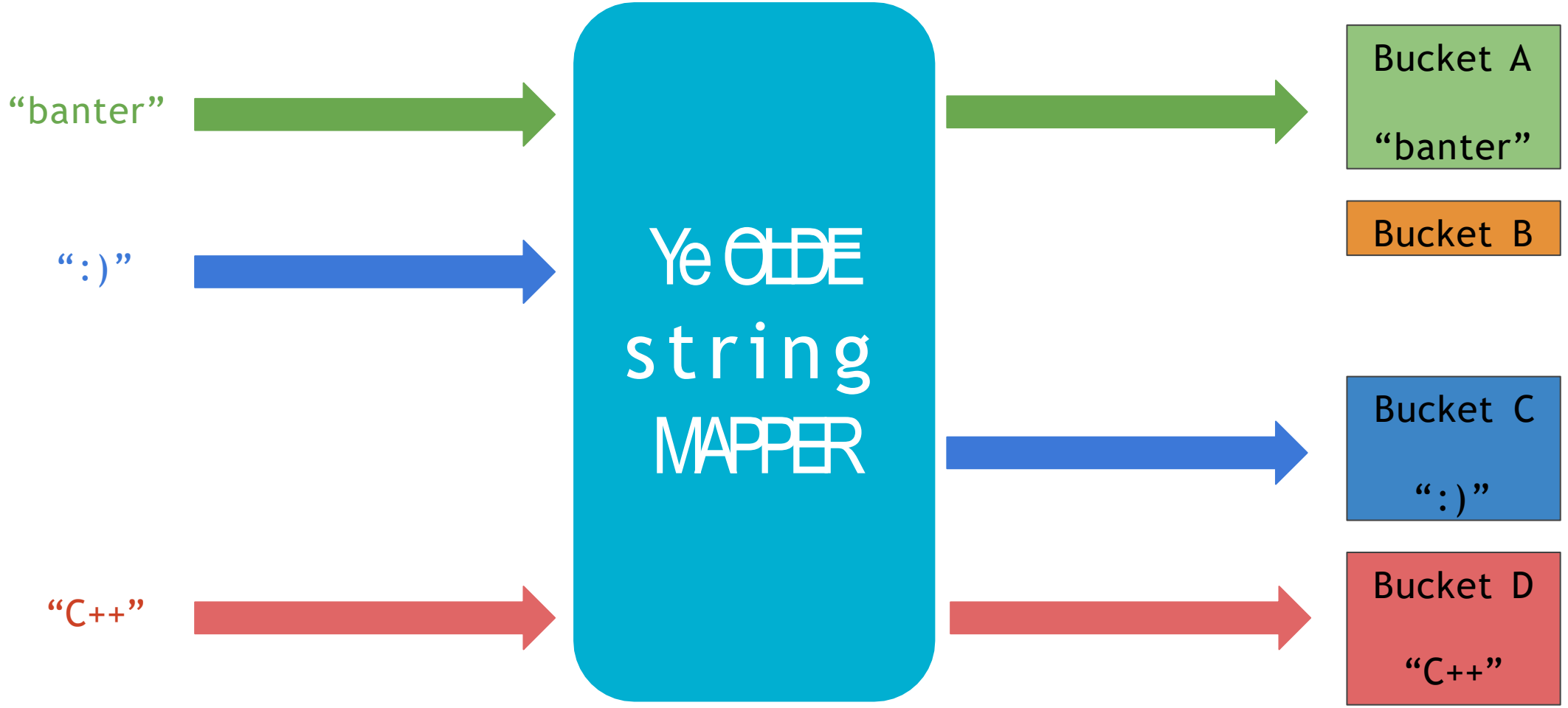
Lookup is O(1)

Insertion is O(1)

Removal is O(1)

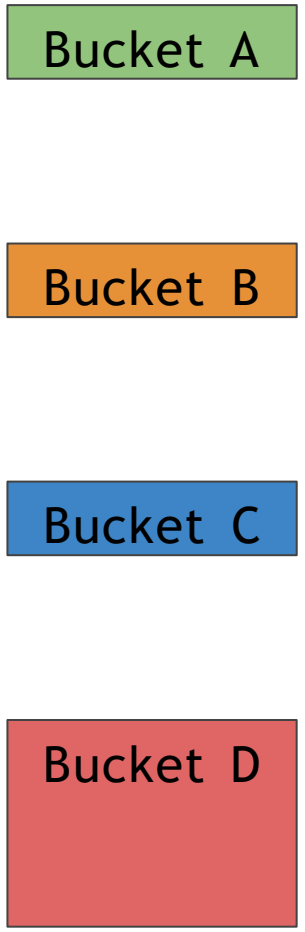
} Could we use this in a data structure?

How a mapped data structure might look



The last piece of the puzzle

How do we formalize the mapping between strings and buckets?



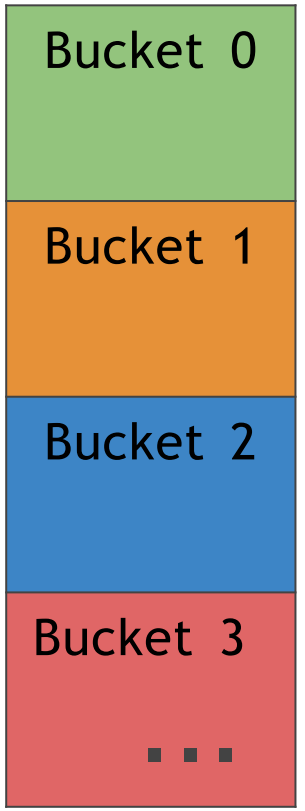
The last piece of the puzzle

How do we formalize the mapping between `string`s and buckets?

Step 1: Turn the buckets into an array

Ye OLDE `string` MAPPER

```
string *buckets = new string[nBuckets];
```

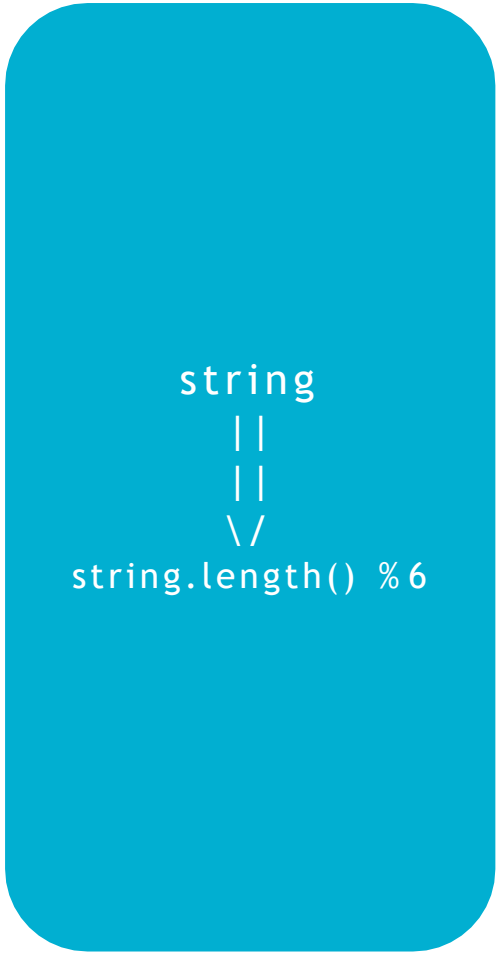


The last piece of the puzzle

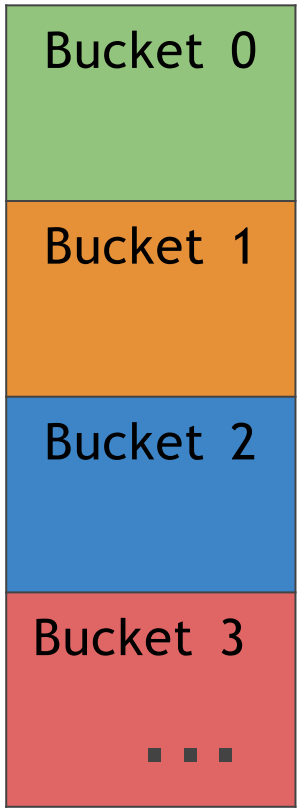
How do we formalize the mapping between `string`s and buckets?

Step 1: Turn the buckets into an array

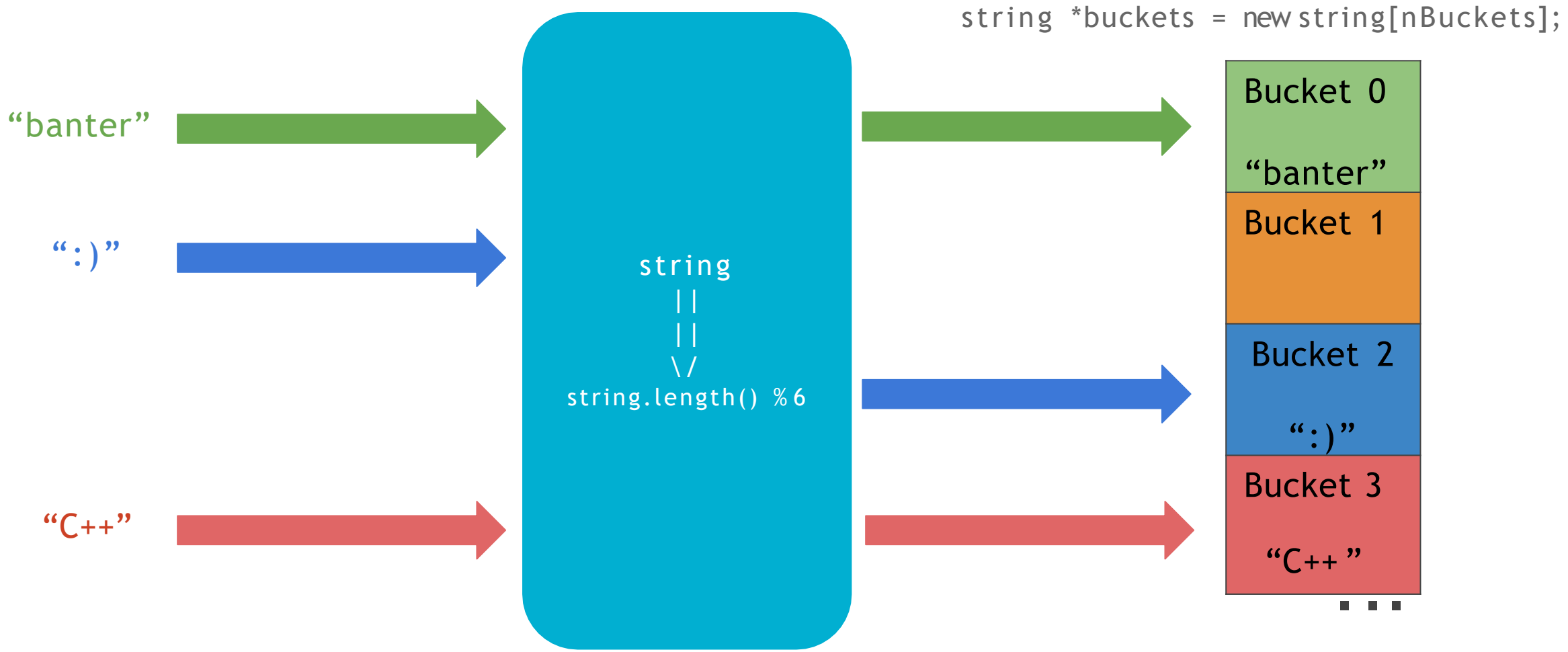
Step 2: Define a function from a `string` to the index of a bucket in the array



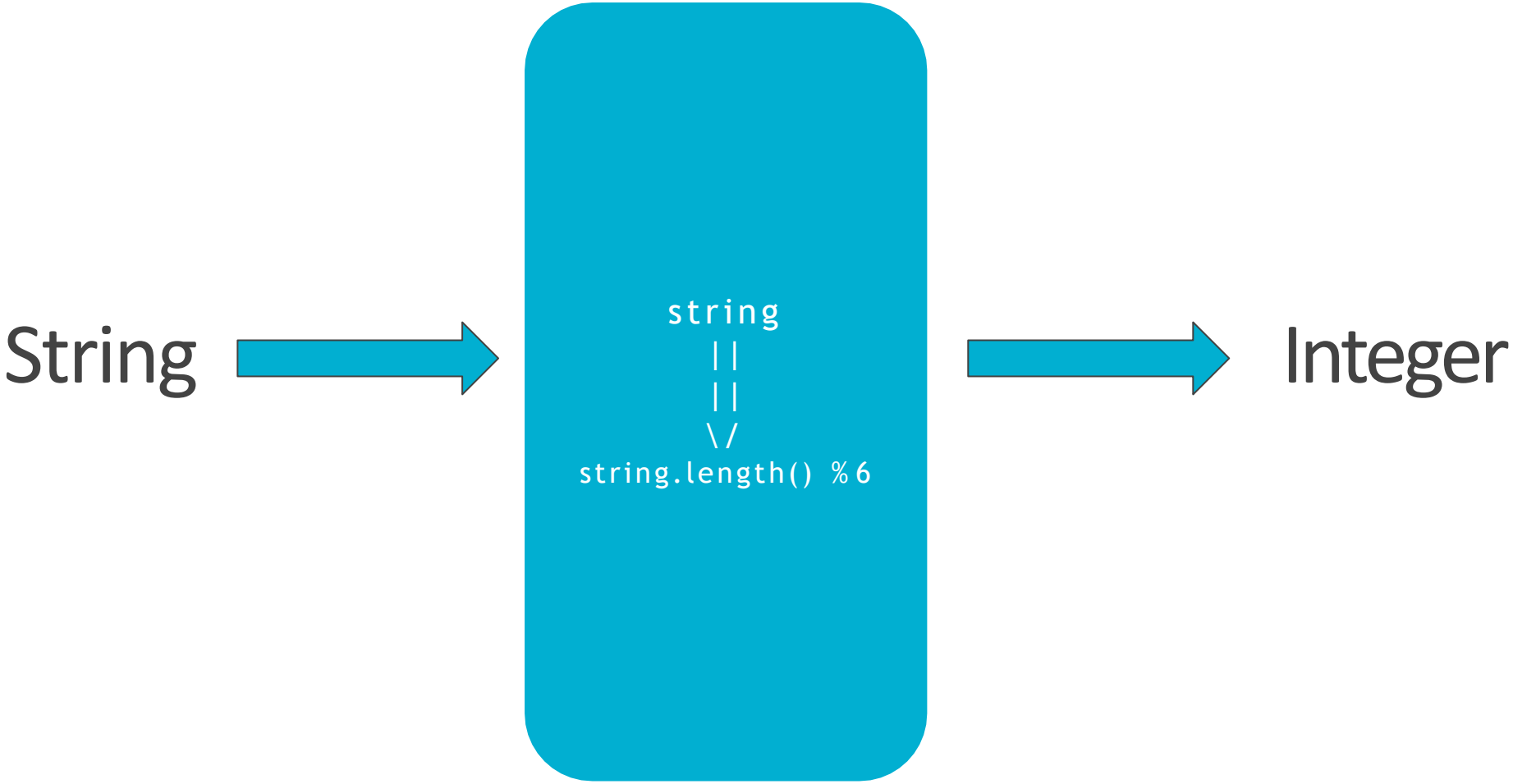

```
string *buckets = new string[nBuckets];
```



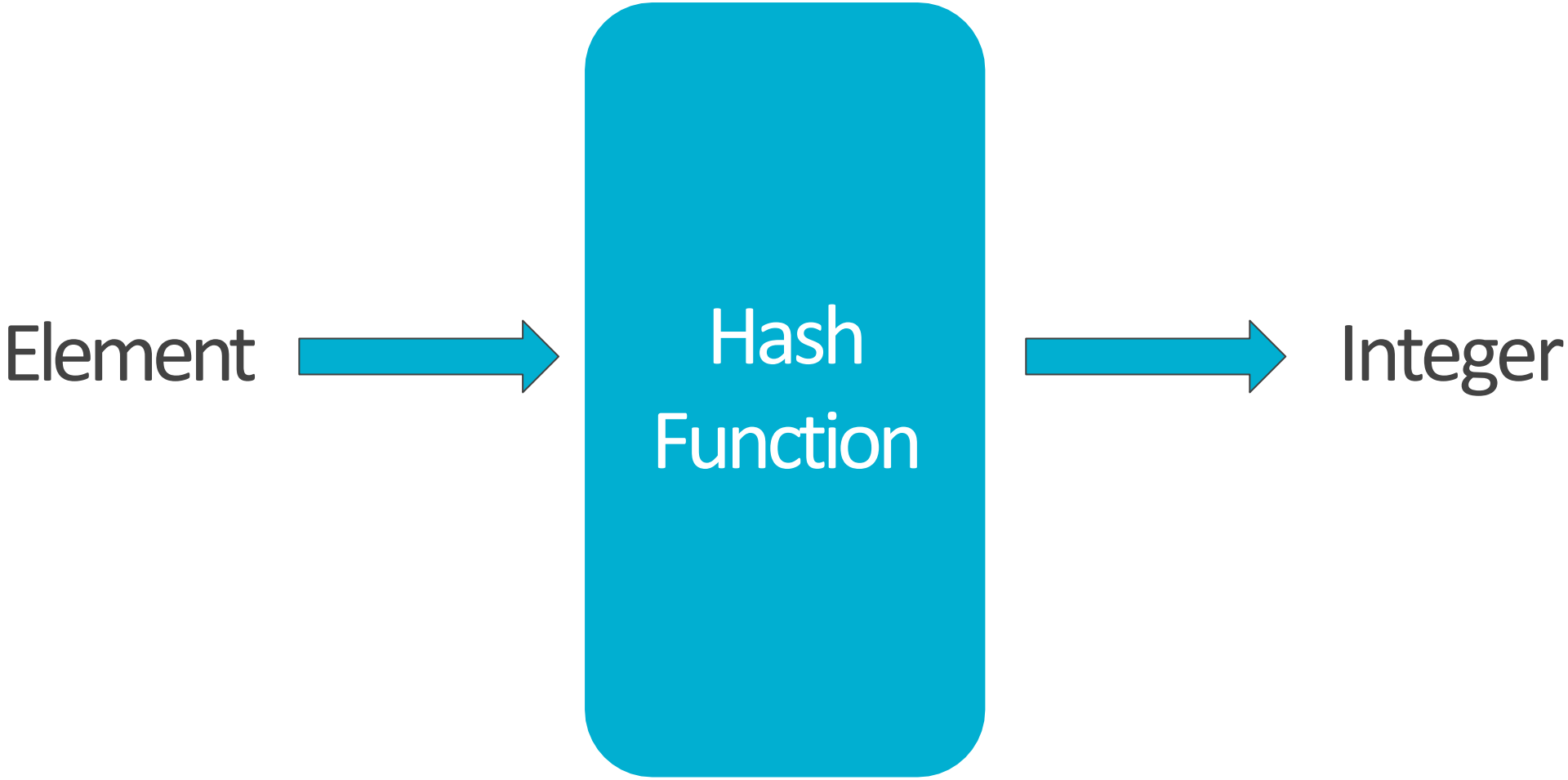
Putting it all together



Our Hash Function



The General Hash Function



Hash Table

- In hashing, the data is organized with the help of a table, called the hash table, denoted by HT, and the hash table is stored in an array.
- To determine whether a particular item with a key, say X, is in the table, we apply a function h, called the hash function, to the key X; that is, we compute h(X), read as h of X.
- The function h is typically an arithmetic function and h(X) gives the address of the item in the hash table.
- Suppose that the size of the hash table, HT, is m. Then $0 \leq h(X) < m$.
- Thus, to determine whether the item with key X is in the table, we look at the entry HT[h(X)] in the hash table.
- Because the address of an item is computed with the help of a function, it follows that the items are stored in no particular order.
- Before continuing with this discussion, let us consider the following questions:
  - How do we choose a hash function?
  - How do we organize the data with the help of the hash table?

Data Organization in Hash Table

- There are two ways that data is organized with the help of the hash table.
- Array
  - The data is stored within the hash table, that is, in an array.
- Linked List
- The data is stored in linked lists and the hash table is an array of pointers to those linked lists.
- Each approach has its own advantages and disadvantages.
- Some more terminology
  - The hash table HT is, usually, divided into, say b buckets HT[0], HT[1], . . ., HT[b – 1].
  - Each bucket is capable of holding, say r items.
  - Thus, it follows that br = m, where m is the size of HT.
- Generally, r = 1 and so each bucket can hold one item.
- The hash function h maps the key X onto an integer t, that is, h(X) ¼ t, such that

  0 ≤h(X) ≤b – 1.

Example

- Suppose there are six students a1, a2, a3, a4, a5, a6 in the Data Structures class and their IDs are
  - a1: 197354863 → k1
  - a2: 933185952 → k2
  - a3: 132489973 → k3
  - a4: 134152056 → k4
  - a5: 216500306 → k5
  - a6. 106500306. → k6
- Suppose that HT denotes the hash table and HT is of size 13 indexed 0, 1, 2, . . ., 12.
- Define the function h: {k1, k2, k3, k4, k5, k6} ! {0, 1, 2, . . ., 12} by $h(k_i) = k_i \% 13$.

Example

$h(k_1) = h(197354863) = 197354863 \% 13 = 4$	$h(k_4) = h(134152056) = 134152056 \% 13 = 12$
$h(k_2) = h(933185952) = 933185952 \% 13 = 10$	$h(k_5) = h(216500306) = 216500306 \% 13 = 9$
$h(k_3) = h(132489973) = 132489973 \% 13 = 5$	$h(k_6) = h(106500306) = 106500306 \% 13 = 3$

[Solution]

H[0]	
H[1]	
H[2]	
H[3]	106500306,a6
H[4]	197354863, a1
H[5]	132488973, a3
H[6]	
H[7]	
H[8]	
H[9]	216500306,a5
H[10]	933185952, a2
H[11]	
H[12]	134152056,a4

Example 2

[Your Turn]

- Suppose there are eight students in the class in a college and their IDs are
  - 197354864 → k1
  - 933185952 → k2
  - 132489973 → k3
  - 134152056 → k4
  - 216500306 → k5
  - 106500306 → k6
  - 216510306 → k7
  - 197354865. → k8

Example

[Solution]

$h(k_1)$ =197354864 % 13 = 5	$h(k_4)$ = 134152056 % 13 = 12	$h(k_7)$ = 216510306 % 13 = 12
$h(k_2)$ = 933185952 % 13 = 10	$h(k_5)$ = 216500306 % 13 = 9	$h(k_8)$ = 197354865 % 13 = 6
$h(k_3)$ = 132489973 % 13 = 5	$h(k_6)$ = 106500306 % 13 = 3	

H[0]	
H[1]	
H[2]	
H[3]	
H[4]	
H[5]	
H[6]	
H[7]	
H[8]	
H[9]	
H[10]	
H[11]	
H[12]	

Hash Functions

- Some of the commonly used hash functions.
  - **Mid-Square:** In this method, the hash function, h, is computed by squaring the identifier, and then using the appropriate number of bits from the middle of the square to obtain the bucket address. Because the middle bits of a square usually depend on all the characters, it is expected that different keys will yield different hash addresses with high probability, even if some of the characters are the same.
  - **Folding:** In folding, the key X is partitioned into parts such that all the parts, except possibly the last parts, are of equal length. The parts are then added, in some convenient way, to obtain the hash address.
  - **Division (Modular arithmetic):** In this method, the key X is converted into an integer i_X. This integer is then divided by the size of the hash table to get the remainder, giving the address of X in HT. That is, (in C++) h(X) = i_X % HTSize;

Hash Functions

```
int hashFunction(char *insertKey, int keyLength)
{
    int sum = 0;

    for (int j = 0; j < keyLength; j++)
        sum = sum + static_cast<int>(insertKey[j]);

    return (sum % HTSize);
} // end hashFunction
```

Collisions ?

- Unless I have infinite buckets, I can’t guarantee that everything will have its own bucket
- If two things are hashed into the same bucket, a collision has occurred
- This isn’t necessarily a bad thing

Collision Resolution

- In hashing, we must include algorithms to handle collisions.
- Collision resolution techniques are classified into two categories:
  i. open addressing (also called closed hashing)
  ii. chaining (also called open hashing)
- In open addressing, the data is stored within the hash table.
- In chaining, the data is organized in linked lists and the hash table is an array of pointers to the linked lists.

Chaining

Linear Probing

- Open addressing can be implemented in several ways, but the common ways to implement is using Linear Probing

Example

- Using the hash function ‘key mod 7’, insert the following sequence of keys in the hash table-

  50, 700, 76, 85, 92, 73 and 101

Example

- Suppose there are six workers, in a workshop, with IDs 147, 169, 580, 216, 974, and 124. Suppose hash table, HT, is of the size 13, indexed 0, 1, 2, . . ., 12.
- Show how these workers' IDs, in the order given, are inserted in HT using the hashing function h(k) = k % 13.

  Use Chaining and linear probing to resolve collision