ITS202: Algorithms and Data Structures

Advanced Data Structures

Ms. Sonam Wangmo

Gyalpozhing College of Information Technology
Royal University of Bhutan

November 29, 2020

Hash Tables

Concepts

- A hash table is a data structure that is used to store keys/value pairs.
- Uses hash function to compute an index into an array in which an element will be inserted or searched.
- The best case time complexity required to search, insert and delete for an element in a hash table is O(1). Fast Retrieval of data no matter how much data is.

Hash Tables

Hash Functions

- Hash function Maps each key k to an integer in the range [0, N-1].
- Calculation applied to a key to transform into an address.
- For numeric keys, divide the key by the number of available addresses, n, and take the remainder.
  Address = Key Mode n
- For alphanumeric keys, divide the sum of ASCII codes in key by the number of available addresses, n, and take the remainder.

Hash Tables

hash code: an int between $-2^{(}32)$ and $2^{(}32)-1$

```
private int hash(Key key)
{  return key.hashCode() % M;  }
```

bug

```
private int hash(Key key)
{  return Math.abs(key.hashCode()) % M;  }
```

1-in-a-billion bug

hashCode() of "polygenelubricants" is -2^{31}

```
private int hash(Key key)
{  return (key.hashCode() & 0x7fffffff) % M;  }
```

correct

Hash Tables

Collisions

If there are two or more keys with the same hash value, we say a collision occurred.

Hash Tables

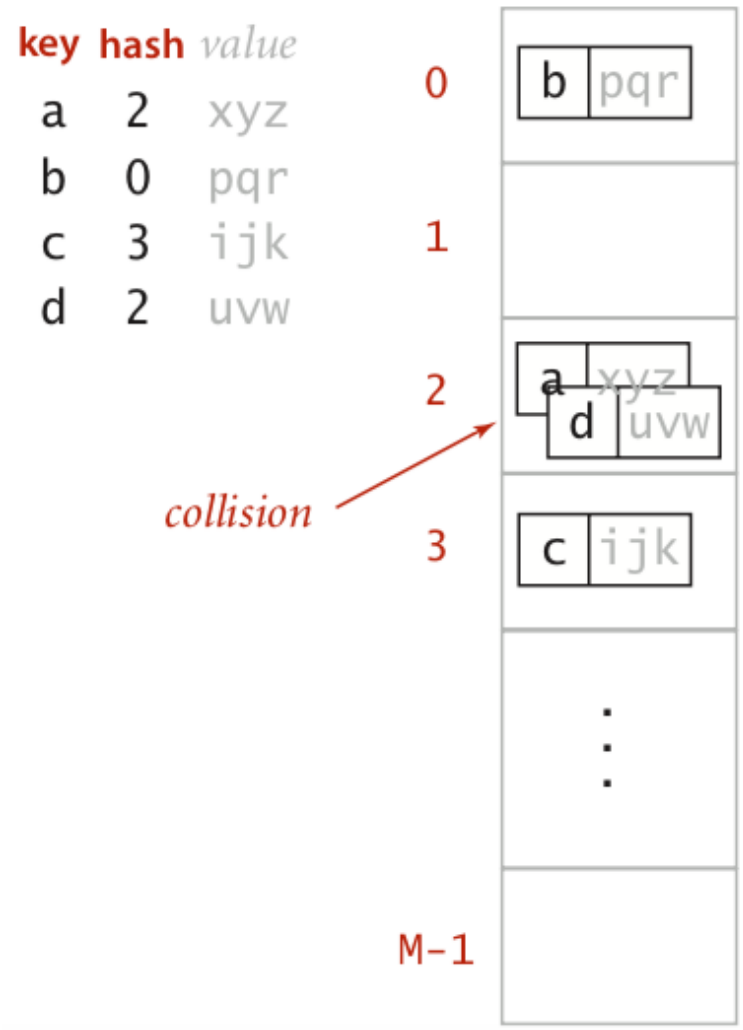


Figure 1: Collision in Hash Table

Hash Tables

Methods to deal with collision:

1. Separate Chaining or Closed-addressing Hashing method
2. Linear Probing or Open-addressing Hashing method

Hash Tables

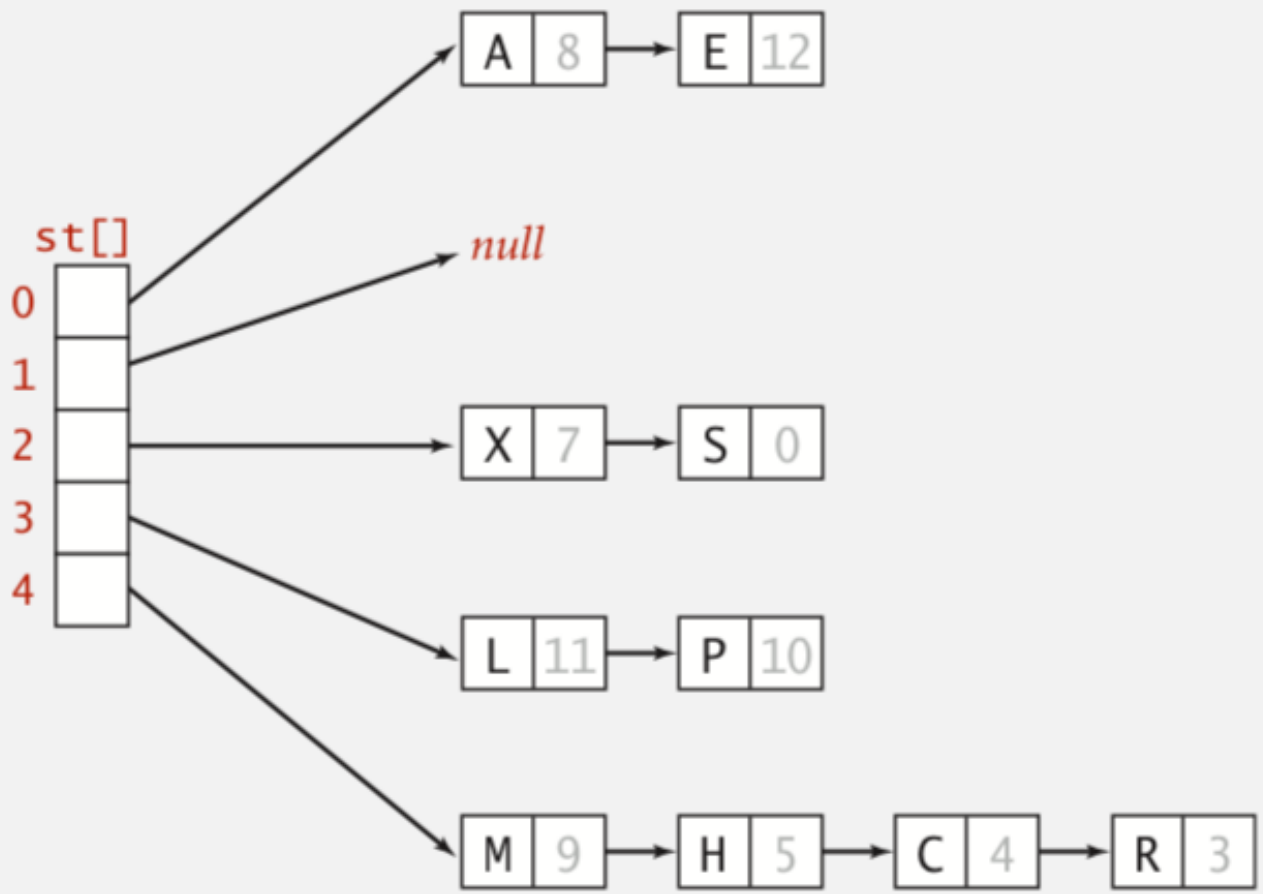
Methods to deal with collision: Separate Chaining

A method by which linked lists of values are built in association with each location within the hash table when the collision occurs.

- Hash: map key to integer i between 0 and M - 1.
- Insert: put at front of ith chain (if not already there).
- Search: need to search only ith chain.

Hash Tables: Seperate Chaining

key	hash	value
S	2	0
E	0	1
A	0	2
R	4	3
C	4	4
H	4	5
E	0	6
X	2	7
A	0	8
M	4	9
P	3	10
L	3	11
E	0	12

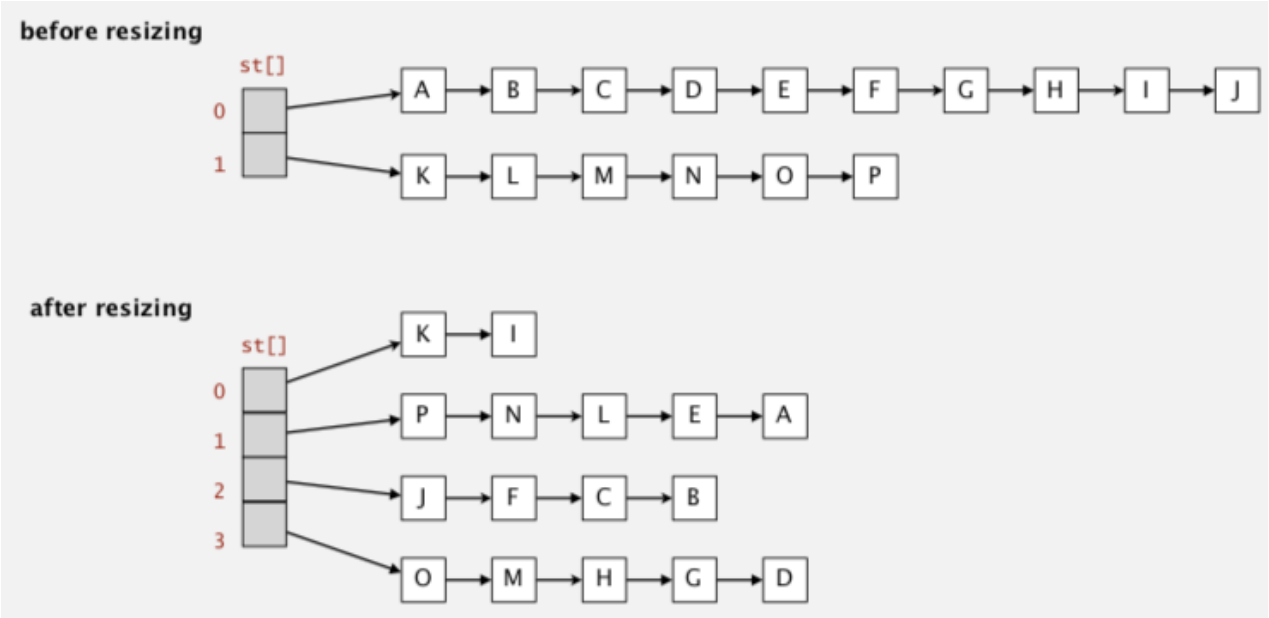


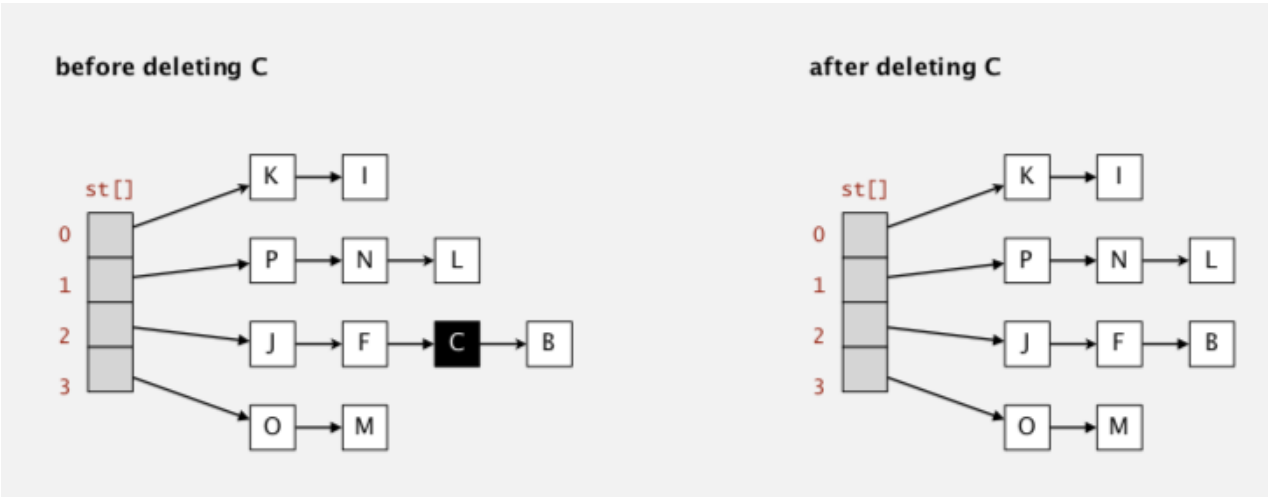
Hash Tables: Seperate Chaining: Java implementation

```
public Value get(Key key) {
   int i = hash(key);
   for (Node x = st[i]; x != null; x = x.next)
      if (key.equals(x.key)) return (Value) x.val;
   return null;
}
```

```
public void put(Key key, Value val) {
   int i = hash(key);
   for (Node x = st[i]; x != null; x = x.next)
      if (key.equals(x.key)) { x.val = val; return; }
   st[i] = new Node(key, val, st[i]);
}
```

Hash Tables: Seperate Chaining: Resizing and deletion





Hash Tables: Seperate Chaining: Resizing and deletion

Double size of array M when N / M >= 8. Halve size of array M when N / M =< 2. Need to rehash all keys when resizing.

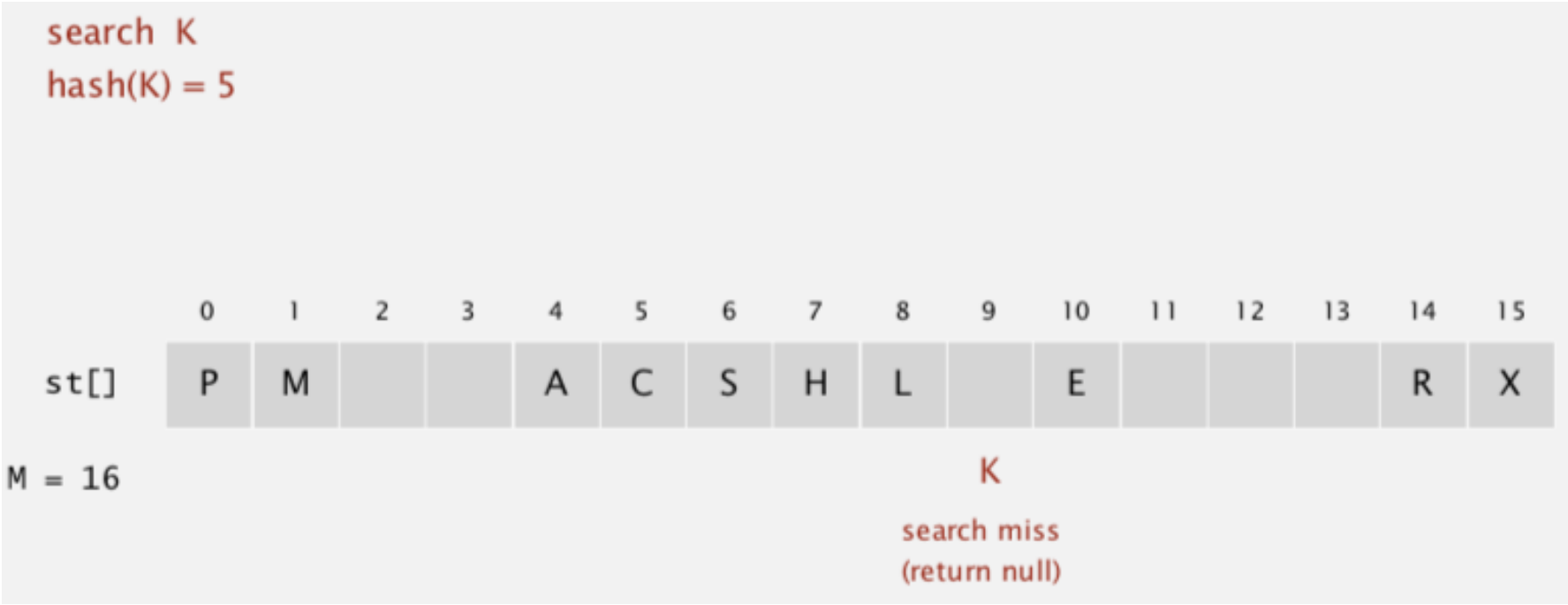
Hash Tables: Linear Probing

Methods to deal with collision: Linear Probing

A strategy for resolving collision or keys that maps to the same index in a hash table such that If the index spot contains the value, use the next index spot and continue until the free spot is available. Also, when the end of the array is reached, go back to the front of the array.

- Hash:Map key to integer i between 0 and M-1.
- Insert: Put at table index i if free; if not try i+1, i+2, etc.
- Search: Search table index i; if occupied but no match, try i+1, i+2, etc.

Hash Tables:Linear Probing



Hash Tables:Performance

Both Linear Probing and Seperate Chaining take O(n) time complexity though it takes 0(1) in best case time complexity.