



Data Structures (Spring 2020)

Linear Hashing (7th Lab)

2020.05.01

Seoul National University
Database Systems Lab

Today's Lab

- Linear Hashing Implementation
- Ask questions about Programming Assignment!



Linear Hashing

- Linear Hashing
 - Linear hashing is a kind of dynamic, open hashing.
- Why Linear Hashing?
 - *Static Hashing*. Long overflow chains can develop and degrade performance.
 - handles duplicates!
 - Since buckets are split round-robin, long overflow chains don't develop!

Linear Hashing

- Insert: 20, 15, 5, 9, 16, 8, 13, 4
- Initial empty hash table of 4 entries
- hash funcs: $h_i(K) = K \bmod 2^i$ for $i \geq 2$.

- Examples

0	16 8
1	9
2	
3	
4	20 4
5	5 13
6	
7	15

- 9 causes the first split, $splitindex = 1$.
- $h_i(16) = 0 < splitindex$. Insert 16 to $h_{i+1}(16) = 0$.
- $h_i(8) = 0 < splitindex$. Insert 8 to $h_{i+1}(8) = 0$, causing the second split, $splitindex = 2$.
- $h_i(13) = 1 < splitindex$. Insert 13 to $h_{i+1}(13) = 5$, causing the third split, $splitindex = 3$.
- $h_i(4) = 0 < splitindex$. Insert 4 to $h_{i+1}(4) = 4$, causing the fourth split, $splitindex = 0$.
- Ready for the next round.

Exercises

- Fill the blank of codes
 - Update your code in LinearHash.java ("// TODO: " section)
 - Modify size(), h(), insert(), search(), remove(), etc..
 - Input argument (such as size()) can be modified



Project Structure

```
public class Main {  
    // main point.  
    public static void main(String[] args) {  
        int size = Integer.parseInt(args[0]);  
        LinearHash<Integer, Integer> T = new LinearHash<Integer, Integer>(size);  
        System.out.println("Hash table size: " + T.size());  
  
        // Insert test  
        for (int i = 1; i < args.length; i++) {  
            T.insert(Integer.parseInt(args[i]), Integer.parseInt(args[i]));  
        }  
  
        // Search test  
        for (int i = 0; i < args.length; i++) {  
            T.search(Integer.parseInt(args[i]));  
        }  
    }  
}
```

Main.java

Exercises

- Result

```
$ java Main 2 20 15 5 9 16 8 13 4
Hash table size: 4
inserted: <0, 20>
inserted: <3, 15>
inserted: <1, 5>
inserted: <1, 9>, split caused: <0, 20> -> <4, 20>
inserted: <0, 16>
inserted: <0, 8>, split caused: <1, 5> -> <5, 5>
inserted: <5, 13>
inserted: <4, 4>, split caused: <3, 15> -> <7, 15>
search 20: <4, 20>
search 15: <7, 15>
search 5: <5, 5>
search 9: <1, 9>
search 16: <0, 16>
search 8: <0, 8>
search 13: <5, 13>
search 4: <4, 4>
```

0	16 8
1	9
2	
3	
4	20 4
5	5 13
6	
7	15