Rehashing

- As more keys are inserted into the hashtable, the performance degrades. Why?
- The solution to this problem is *rehashing*. A new hashtable is created, and all keys are rehashed to the new table.

Q1: When is a good time to rehash?
Q2: What is a good size for the new hashtable?
When to rehash?

- In arrays, we used to expand the array when it is full. Should we do the same with hashtables?

- What happens if we wait until the hashtable is full?
  - with linear probing
  - with quadratic probing

- Load Factor: $\lambda = \frac{\# \text{ of keys}}{\# \text{ of buckets}}$

- $0 \leq \lambda \leq 1$
Load Factor

We would better stay on this side

\( \lambda = 0.5 \)
Rehashing

- When $\lambda > 0.5$
- New size is roughly double the old side

```cpp
Rehash() {
    T* newHashtable = new T[new_size];
    for (i = 0 to old_size) {
        if (bucket #i is occupied) {
            Insert the key at bucket #i into the new table;
        }
    }
    replace the old table with the new one;
}
```
Rehashing Example

0
1
2
3
4
5
6

\[ h = x \% 7 \]

Insert \{37, 8, 3, 16, 26\}
Rehashing Example

\[ h = x \% 7 \]

Insert \{30, 8, 3, 16, 26\}

\[ h = x \% 13 \]
Rehashing Example

$$h = x \% 7$$

Insert \{37, 8, 3, 16, 26\}

$$h = x \% 13$$
Application to Hashtables

- Bloom filter
- Stores a set of keys
- Answers one question: Is the key $x$ in the set or not?
- Application: Used as a prefilter to avoid costly searches when the key is not there
  - e.g., BST search, hashtable search, ordered list search, unordered list search
Bloom Filter

Initialize: Create a bit vector all set to zeros

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Bloom Filter

Insert(x) → hash(x)

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Bloom Filter

Insert(x) → hash(x)

0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Bloom Filter

Search(x) → hash(x) → Return

0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Implementation

Initialize(m) {
    b = new bit vector[m];
}

Insert(x) {
    b[hash(x)] = 1;
}

Search(x) {
    return b[hash(x)];
}
Collisions

- What to do with collisions?
- Nothing!!
- What are the consequences of this?
- False positives
- How to support deletions?
- Deletions are not supported
Multiple Hash Functions

Insert(x)

hash1(x)  hash2(x)  hash3(x)

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Multiple Hash Functions

\[ \text{Insert}(x) \]

\[ \text{hash}_1(x) \quad \text{hash}_2(x) \quad \text{hash}_3(x) \]

![Diagram of multiple hash functions with a binary tree structure and data values at leaf nodes.](image)
Multiple Hash Functions

Search(x)

hash1(x) \quad hash2(x) \quad hash3(x)

\[ \begin{array}{cccccccccccccccccc}
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array} \]

AND

\downarrow

Return