CS261 Data Structures

Hash Tables

Buckets/Chaining

Hash Tables: Resolving Collisions

There are two general approaches to resolving collisions:

1. Open address hashing: if a spot is full, probe for next empty spot

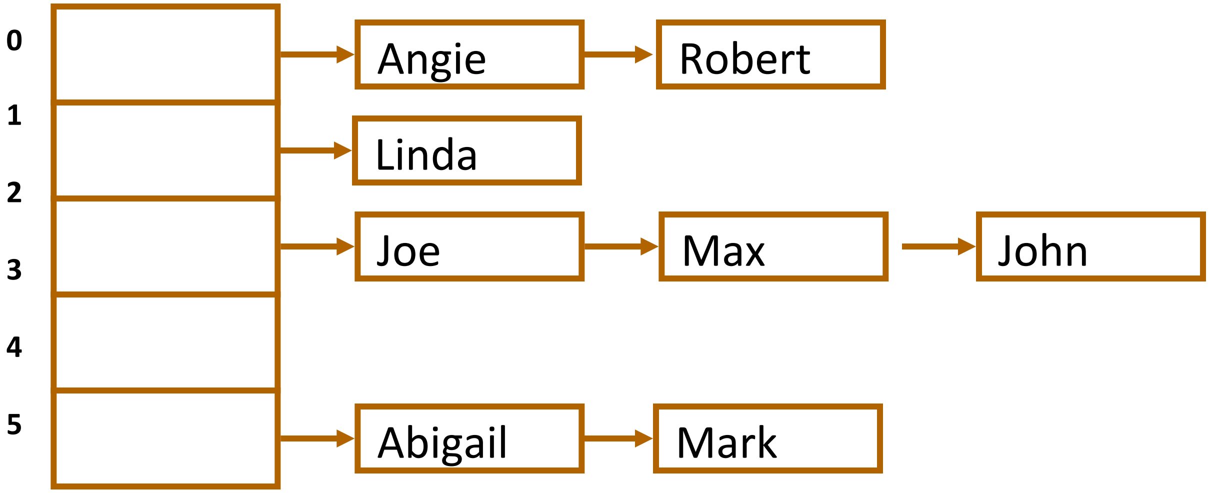
2. Chaining (or buckets): keep a collection at each table entry

Resolving Collisions: Chaining / Buckets

Maintain a collection (typically a BAG ADT) at each table entry:

Each collection is called a 'bucket' or a 'chain'

solved collision problem by allowing multiple things to be stored at each location



Hash Table Implementation: Initialization

```
struct HashTable {
  struct Linked List **table;    /* Hash table → Array of Lists. */
  int  capacity;
  int count;
}

void initHashTable(struct HashTable *ht, int size) {
  int i;

  ht->capacity  = size;
  ht->count   = 0;
  ht->table = malloc(ht->capacity * sizeof(struct LinkedList *));
  assert(ht->table != 0);
  for(i = 0; i < ht->capacity; i++) ht->table[i] = newList();
}
```

Hash Table Implementation: Add

```
void addHashTable(struct HashTable *ht, TYPE val) {
  /* Compute hash table bucket index. */
  int idx = hash(val) % ht->capacity;
  if (idx < 0) idx += ht->capacity;
```

takes care of any negative index values

```
  /* Add to bucket. */
  addList(ht->table[idx], val);
  ht->count++;
```

use index to choose correct value in table

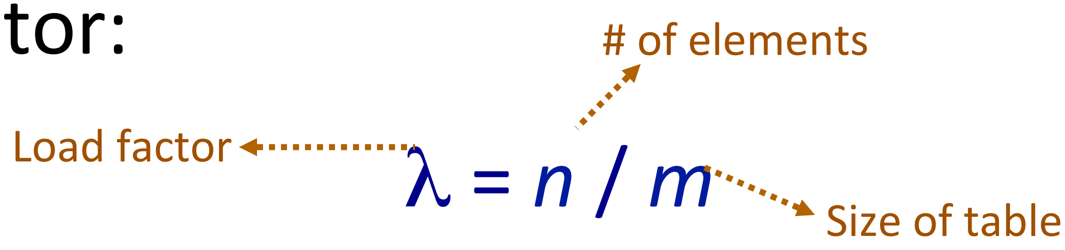
```
  /* Next step: Reorganize if load factor to large.   More on
this later! */
}
```

Hash Table: Contains & Remove

- Contains: find correct bucket using the hash function, then checks to see if element is in the linked list
- Remove: if element is in the table, remove it and decrement the count

Hash Table Size

- Load factor:



# of elements

Load factor

$\lambda = n / m$

Size of table

  - Load factor represents average number of elements in each bucket
  - **For chaining, load factor can be greater than 1**

- As in open address hashing: if load factor becomes larger than some fixed limit (say, 8) → double table size

Hash Table

- Load factor:

# of elements

Load factor ← $\lambda = n / m$ → Size of table

  - The average number of links traversed in successful searches, S, and unsuccessful searches, U, is

$$S \approx 1 + \frac{\lambda}{2} \qquad U \approx \lambda$$

  - If load factor becomes larger than some fixed limit (say, 8) → double table size

Hash Tables: Algorithmic Complexity

- Assuming:
  - Time to compute hash function is constant
  - Chaining uses a linked list
  - Worst case analysis → All values hash to same position
  - Best case analysis → Hash function uniformly distributes the values and we have no collisions

- Contains operation:
  - Worst case for open addressing → O(n)
  - Worst case for chaining → O(n)
  - Best case for open addressing → O(1)
  - Best case for chaining → O(1)

Hash Tables With Chaining: Average Case

- Assume hash function distributes elements uniformly (a BIG if)
- And we have collisions
- Average case for all operations: $O(\lambda)$
- Want to keep the load factor relatively small
- Resize table (doubling its size) if load factor is larger than some fixed limit (e.g., 8)
  - Only improves things *IF* hash function distributes values uniformly
  - How do we handle a resize?

Design Decisions

- Implement the Map interface to store values with keys (ie. implement a dictionary)
- Rather than store linked lists, build the linked lists directly
  - Link **hashTable;

Your Turn

- Worksheet 38: Hash Tables using Buckets