DS221: Introduction to Scalable Systems

Topic: Algorithms and Data Structures

L4 (Extension)
Analysis of Hash Tables

[Recall] Complexity Of Dictionary Operations find(), insert()

- Given **n** elements in the dictionary

Data Structure	Worst Case	Expected
Hash Table (linear probing)	O(n)	O(1) *
Binary Search Tree	O(n)	O(log n)
Balanced Binary Search Tree	O(log n)	O(log n)

* Assumptions: (i) Each key's hash is uniform and independent over the 'b' buckets and (ii) the *load factor* (= n/b) is a constant strictly less than one

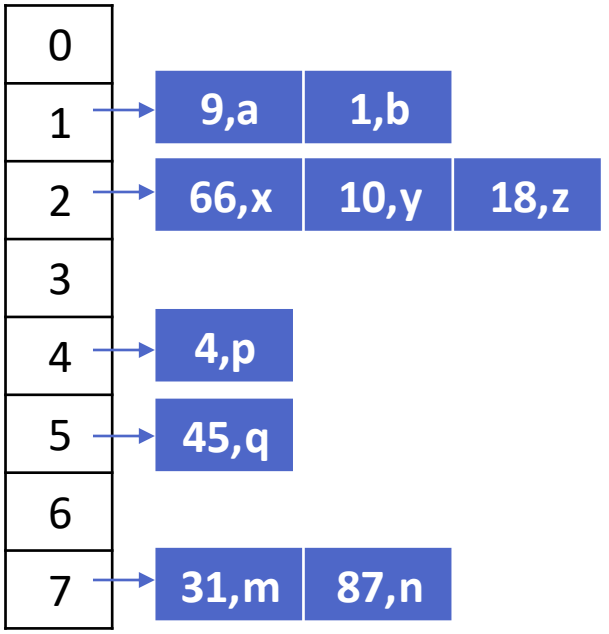
[Recall] Hash Table

- Uses a 1D array (or table) table[0:b-1]
  - Each position of this array is a **bucket**
  - Number of buckets is b
  - A bucket can normally hold only one dictionary pair: <key, value>
    - But larger capacity allowed per bucket as well
    - Bucket sizes can be unbounded as well
- Uses a hash function h that converts each key k into an index in the range [0, b-1].
  - h(k) is the "home bucket" for key k.
- Every dictionary pair is stored in its home bucket

  table[h(item.key)] = item

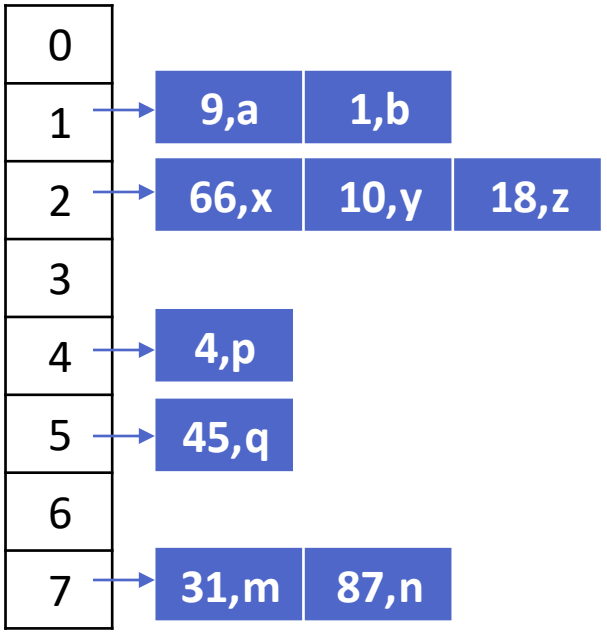
[Recall] Hash Table with Chaining

- Buckets with unbounded capacity
  - Bucket as a linked list
- Hash function gives array index
- Array contains pointer to head of linked list
  - Items are <key,value> pairs
- Traverse list to lookup element
- What if key not present?
- Time complexity for Insert? Lookup?



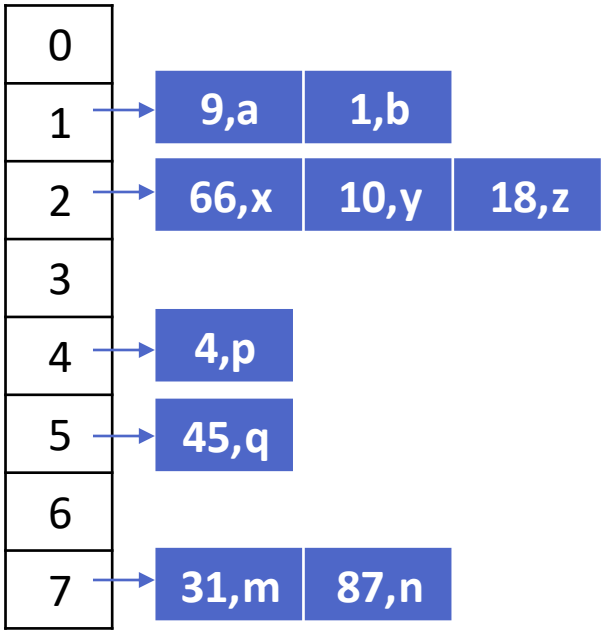
Analysis of hashing with chaining

- **Definition** (load factor)
  - Given a hash table T with b buckets and n elements stored in it, the ***load factor α*** for T is n/b
- The worst-case behaviour of hash table with chaining is terrible: (when all n keys hash to the same bucket)
- Worst case time for searching is θ(n) + the time to compute hash function
  - This is as bad as if we had used one linked list for all elements ☹



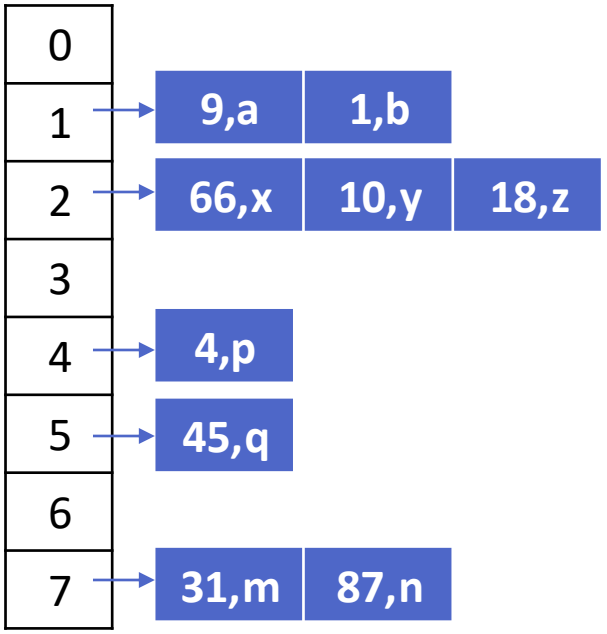
Analysis of hashing with chaining

- Let us analyze the performance with an assumption of *simple uniform hashing*
- **Definition** (Simple uniform hashing)
  - Any given key is equally likely to hash into any of the b buckets, independently of where any other element has hashed to
  - Formally, If $x \neq y$, then $Pr[h(x)=h(y)] = 1/b$
- We will also assume that computing h(k) takes O(1) time



Analysis of hashing with chaining

- Suppose keys $k_1, k_2, \ldots, k_n$ exist in the hash table
- **Claim 1**: The expected length of any list is equals the load factor α under the assumption of simple uniform hashing.



Analysis of hashing with chaining

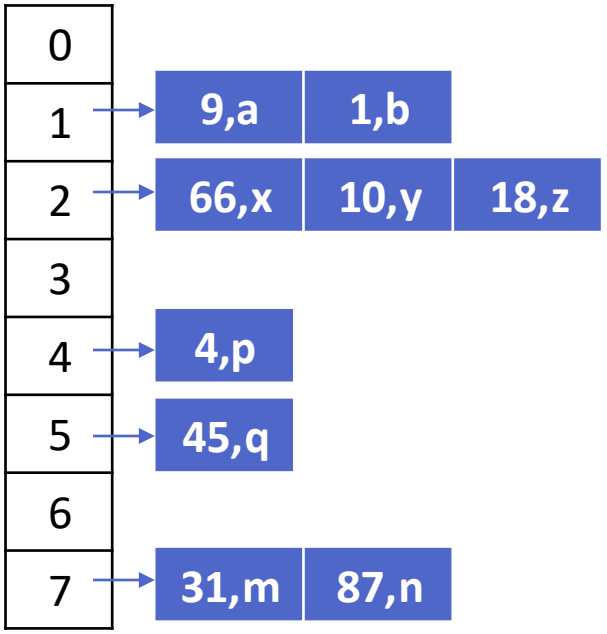
- Suppose keys $k_1, k_2, \ldots, k_n$ exist in the hash table
- **Claim 1**: The expected length of any list is equals the load factor α under the assumption of simple uniform hashing.

- Consider any bucket Z in the table. For key k_i, define the indicator random variable X_i that is 1 if k_i hashes to Z and 0 otherwise.
- $Pr(X_i = 1) = 1/b$
- Length of list at bucket Z equals $\sum_1^n X_i$
- Expected length of list at bucket Z

$= E(\sum_1^n X_i)$

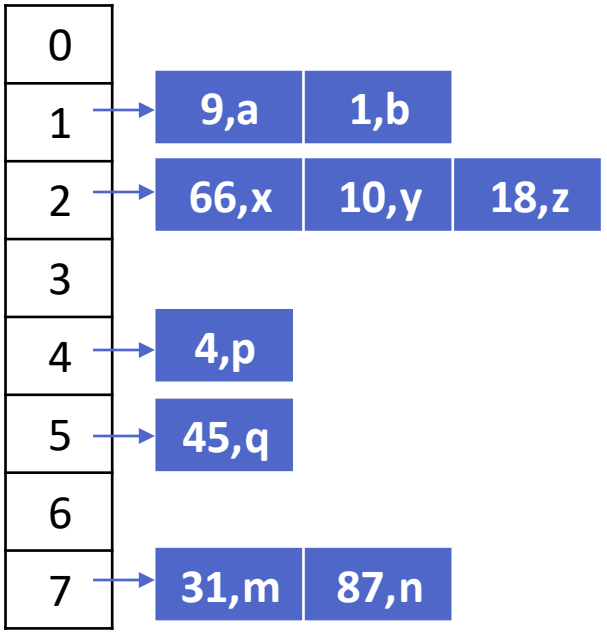
$= \sum_1^n E(X_i)$ [using linearity of expectation]

$= n/b$



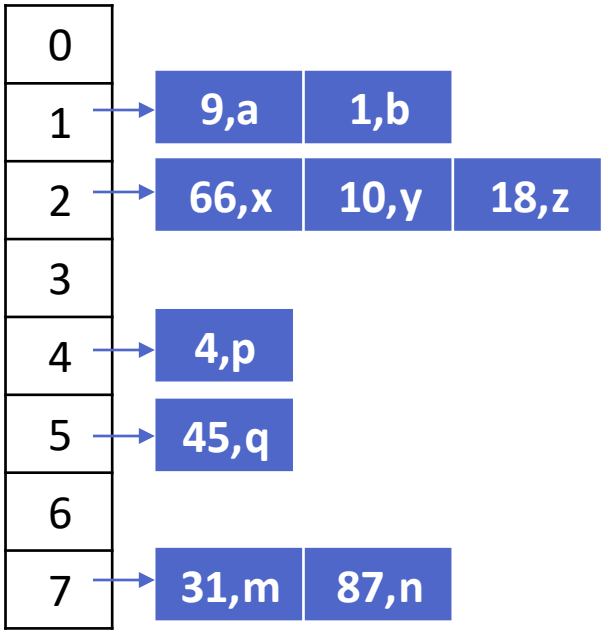
Analysis of hashing with chaining

- Suppose keys $k_1, k_2, \ldots, k_n$ exist in the hash table
- **Claim 2**: The expected time required for an **unsuccessful** search operation is $\theta(1+\alpha)$, under the assumption of simple uniform hashing.



Analysis of hashing with chaining

- Suppose keys $k_1, k_2, \ldots, k_n$ exist in the hash table
- **Claim 2**: The expected time required for an **unsuccessful** search operation is $\theta(1+\alpha)$, under the assumption of simple uniform hashing.



- Proof: Any key k not already in the table is equally likely to hash to any of the b buckets
- The expected time is determined the size of list in bucket h(k), which has expected length α

Analysis of hashing with chaining

- Suppose keys $k_1, k_2, \ldots, k_n$ exist in the hash table
- What about a **successful** search operation?

Analysis of hashing with chaining

- Suppose keys $k_1, k_2, \ldots, k_n$ exist in the hash table
- The situation for a **successful search** is slightly different, since each list is **not** equally likely to be searched. The probability that a list is searched is proportional to the number of keys it contains.

Analysis of hashing with chaining

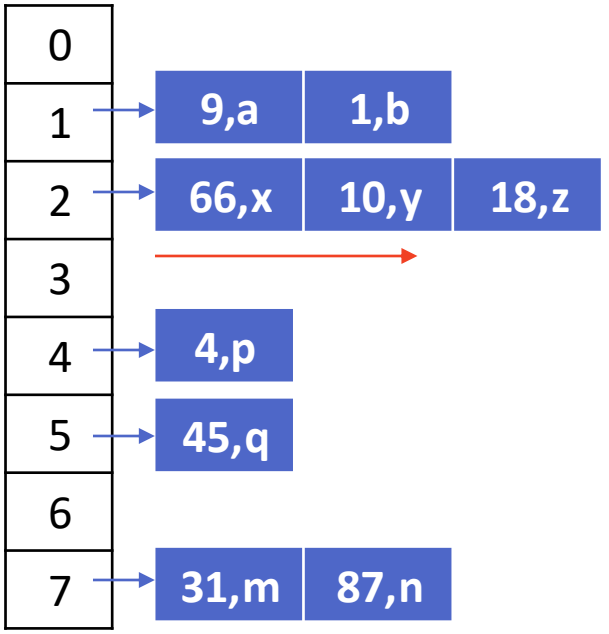
- Suppose keys $k_1, k_2, \ldots, k_n$ exist in the hash table
- The situation for a **successful search** is slightly different, since each list is **not** equally likely to be searched. The probability that a list is searched is proportional to the number of keys it contains.
- Nonetheless, the expected search time still turns out be $\theta(1+\alpha)$. {We'll prove in the next slide}
- **Claim 3**: The expected time required for a **successful** search operation is $\theta(1+\alpha)$, under the assumption of simple uniform hashing. Assume that the key being searched for is equally likely to be any of the existing keys in the table.

Analysis of hashing with chaining

Observations:

- The number of keys examined during a successful search for a key k is one more than the number of keys that **appear before** k in the list at bucket h(k)

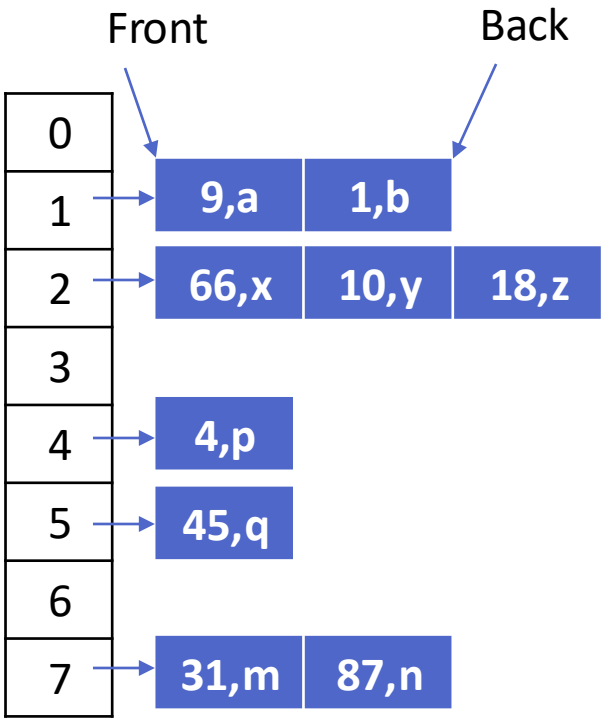
Suppose k=10 and h(k) = 2



Analysis of hashing with chaining

Observations:

- The number of keys examined during a successful search for a key k is one more than the number of keys that appear **before** k in the list at bucket h(k)
- Also recall that **new keys in a bucket are placed at the front** of the list



Analysis of hashing with chaining

Claim 3: The expected time required for a **successful** search operation is $\theta(1+\alpha)$, under the assumption of simple uniform hashing. Assume that the key being searched for is equally likely to be any of the existing keys in the table.

Analysis of hashing with chaining

Claim 3: The expected time required for a **successful** search operation is $\theta(1+\alpha)$, under the assumption of simple uniform hashing. Assume that the key being searched for is equally likely to be any of the existing keys in the table.

[Hint to prove]
Let k_i denote the i^{th} key inserted into the table

Step 1: Check the expected count of keys among $\{k_{i+1}, k_{i+2}, \ldots k_n\}$ that are hashed into same bucket as k_i. This would give us the expected time to search k_i

Step 2: Compute an average, that is, add the above expression for all $1 \leq i \leq n$ and divide by n

Analysis of hashing with chaining

Claim 3: The expected time required for a **successful** search operation is $\theta(1+\alpha)$, under the assumption of simple uniform hashing. Assume that the key being searched for is equally likely to be any of the existing keys in the table.

Let k_i denote the i^{th} key inserted into the table

For keys k_i and k_j, define a random variable X_{ij}

$X_{ij} = 1$ if $h(k_i) = h(k_j)$, $X_{ij} = 0$ otherwise

$Pr(X_{ij} = 1) = 1/b$. Therefore $E(X_{ij}) = 1/b$

Expected #keys examined in searching $k_i = 1 + E[\sum_{j=i+1}^{n} X_{ij}]$

Expected #keys examined in a successful search =

$\frac{1}{n}\sum_{i=1}^{n}(1 + E[\sum_{j=i+1}^{n} X_{ij}])$

Analysis of hashing with chaining

Simplifying the expression:

$$= \frac{1}{n} \sum_{i=1}^{n} (1 + E[\sum_{j=i+1}^{n} X_{ij}])$$

$$= \frac{1}{n} \sum_{i=1}^{n} (1 + \sum_{j=i+1}^{n} E(Xij))$$

$$= \frac{1}{n} (n + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{1}{b})$$

$$= \frac{1}{n} (n + \frac{n(n-1)}{2b})$$

$$= 1 + \frac{(n-1)}{2b}$$

$$= 1 + \frac{\alpha}{2} - \frac{\alpha}{2n}$$

Self-study (Optional, not in syllabus)

How to analyse open-address hashing?

With open addressing, we require that for every key k, the probe sequence h(k,0), h(k,1), …., h(k,b-1) is a permutation of ⟨0, 1,…,b-1⟩

Assume **uniform hashing** where the sequence h(k,0), h(k,1), …., h(k,b-1) used to insert or search for each key k is equally likely to be any permutation of ⟨0, 1,…,b-1⟩

Intuition: For each key k, its probe sequence is fixed after throwing a fair die that has b! faces

Question: Given an open-address hash table with load factor $\alpha = n/b < 1$, what is the expected number of probes in an unsuccessful search?

Tasks

- Self study
  - Chapter 11 of the textbook “Introduction to Algorithms” (CLRS)