CS 457, Fall 2019

Drexel University, Department of Computer Science

Lecture 8

Randomized Selection Algorithm

Randomized-Select(A, p, r, i)

1. if $p == r$
2. return $A[p]$
3. q = Randomized-Partition(A, p, r)
4. $k = q - p + 1$
5. if $i == k$
6. return $A[q]$
7. else if $i \le k$
8. Randomized-Select$(A,\ p,\ q - 1,\ i)$
9. else
10. Randomized-Select$(A,\ q + 1,\ r,\ i - k)$

Randomized-Partition (A, p, r)

1. i = Random(p, r)
2. Exchange $A[r]$ with $A[i]$
3. return Partition(A, p, r)

Running Time

- Indicator variable $\mathbb{I}\{E\}$ is 1 if event E occurs and 0 o/w (see page 118)
- Consider an array $A[p, \dots, r]$ with n elements
- If $X_k = \mathbb{I}\{$the subarray $A[p, \dots, q]$ has exactly k elements$\}$, then

$$T(n) \le \sum_{k=1}^{n} X_k\,(T(\max\{k-1, n-k\}) + O(n))$$

$$= \sum_{k=1}^{n} X_k\,(T(\max\{k-1, n-k\})) + \sum_{k=1}^{n} X_k\,O(n)$$

$$= \sum_{k=1}^{n} X_k\,(T(\max\{k-1, n-k\})) + O(n)$$

Expected Running Time

- $X_k = \mathbb{I}\{\text{the subarray } A[p, \ldots, q] \text{ has exactly } k \text{ elements}\}$. Since $A[p, \ldots, r]$ has n elements and q is chosen uniformly at random, we have $\mathbb{E}[X_k] = \frac{1}{n}$, so

$$
\begin{aligned}
\mathbb{E}[T(n)] &\leq \mathbb{E}\left[\sum_{k=1}^{n} X_k \left(T(\max\{k-1, n-k\})\right) + O(n)\right] \\
&= \sum_{k=1}^{n} \mathbb{E}[X_k(T(\max\{k-1, n-k\}))] + O(n) \\
&= \sum_{k=1}^{n} \mathbb{E}[X_k]\, \mathbb{E}[T(\max\{k-1, n-k\})] + O(n) \\
&= \sum_{k=1}^{n} \frac{1}{n}\, \mathbb{E}[T(\max\{k-1, n-k\})] + O(n)
\end{aligned}
$$

Expected Running Time

Thus, $\mathbb{E}[T(n)] \leq \sum_{k=1}^{n} \frac{1}{n} \mathbb{E}[T(\max\{k-1, n-k\})] + O(n)$,

and $\max\{k-1, n-k\}) = \begin{cases} k-1 \text{ if } k > \lceil n/2 \rceil \\ n-k \text{ if } k \leq \lceil n/2 \rceil \end{cases}$

This is a recurrence
that we can now solve

So, $\mathbb{E}[T(n)] \leq \frac{2}{n} \sum_{k=\left\lfloor \frac{n}{2} \right\rfloor}^{n-1} \mathbb{E}[T(k)] + O(n)$

Expected Running Time

We use substitution in order to solve: $\mathbb{E}[T(n)] \le \frac{2}{n}\sum_{k=\lfloor\frac{n}{2}\rfloor}^{n-1}\mathbb{E}[T(k)] + O(n)$

We guess that $\mathbb{E}[T(n)] = O(n)$

Hence, we assume $\mathbb{E}[T(n')] \le cn'$ for some constant c and every $n' < n$,

and we show that $\mathbb{E}[T(n)] \le cn$ for that same constant c. Upon substitution:

$$\mathbb{E}[T(n)] \le \frac{2}{n}\sum_{k=\lfloor\frac{n}{2}\rfloor}^{n-1} ck + an \quad \le \frac{2c}{n}\left(\frac{n^2-n}{2} - \frac{n^2/4 - 3n/2 + 2}{2}\right) + an$$

which leads to $\mathbb{E}[T(n)] \le cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right)$

Expected Running Time

We use substitution in order to solve: $\mathbb{E}[T(n)] \le \frac{2}{n} \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} \mathbb{E}[T(k)] + O(n)$

We guess that $\mathbb{E}[T(n)] = O(n)$

Hence, we assume $\mathbb{E}[T(n')] \le cn'$ for some constant c and every $n' < n$,

and we show that $\mathbb{E}[T(n)] \le cn$ for that same constant c. Upon substitution:

$$\mathbb{E}[T(n)] \le \frac{2}{n} \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} ck + an \le \frac{2c}{n}\left(\frac{n^2-n}{2} \cdots\right) + an$$

for $\boldsymbol{c > 4a}$,
and $\boldsymbol{n \ge \frac{2c}{c-4a}}$

which leads to $\mathbb{E}[T(n)] \le cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right) \le cn$

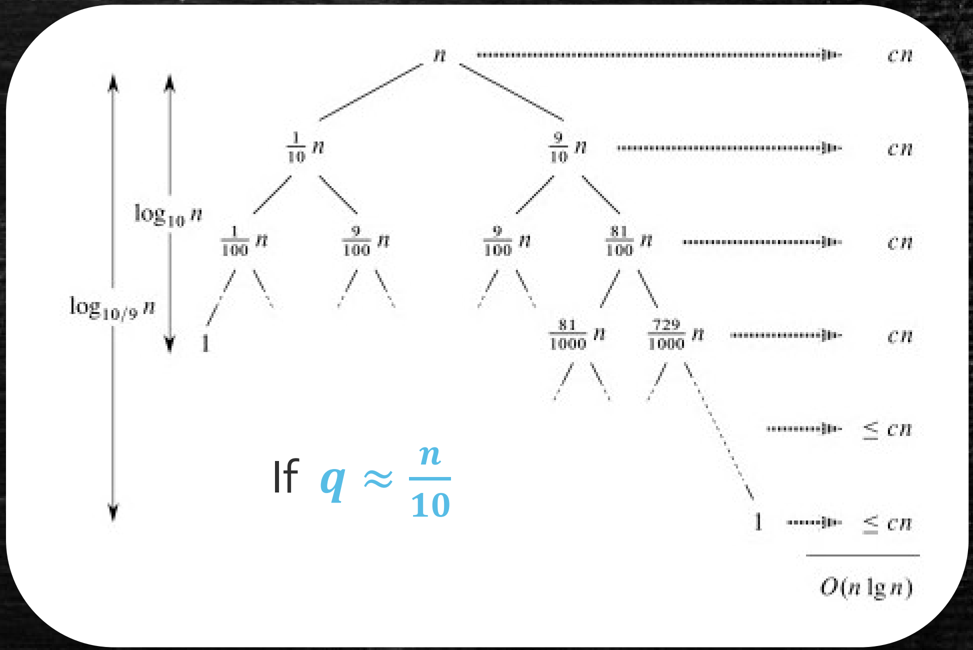
Quicksort (Running Time)

QUICKSORT (A, p, r)

```
if p < r                              // Check for base case
    q = PARTITION(A, p, r)            // Divide step
    QUICKSORT (A, p, q - 1)           // Conquer step
    QUICKSORT (A, q + 1, r)           // Conquer step
```

$$\boldsymbol{T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(q) + T(n-q-1) + \Theta(n) & \text{otherwise} \end{cases}}$$

- $T(n) \le \max\limits_{0 \le q \le n-1} (T(q) + T(n-q-1)) + \Theta(n)$
- This leads to $O(n^2)$ in the worst case
- But, if $q = cn$ for some constant c, it is $O(n \log n)$



Quicksort (Running Time)

How many times is Partition called overall?

QUICKSORT (A, p, r)

1. **if** $p < r$
2. q = PARTITION(A, p, r)
3. QUICKSORT (A, p, q - 1)
4. QUICKSORT (A, q + 1, r)

PARTITION (A, p, r)

1. x = A[r]
2. i = p – 1
3. **for** j = p **to** r – 1
4. **if** A[j] $\leq$ x
5. i = i + 1
6. exchange A[i] with A[j]
7. exchange A[i+1] with A[r]
8. **return** i+1

What is the worst case value of X?

Lemma: Let $\boldsymbol{X}$ be the number of comparisons performed in line 4 of PARTITION over the entire execution of QUICKSORT on an n-element array. Then the running time of QUICKSORT is $\boldsymbol{O(n+X)}$.

Quicksort (Running Time)

RANDOMIZED-QUICKSORT (A, p, r)

```
if p < r
    q = RANDOMIZED-PARTITION(A, p, r)
    RANDOMIZED-QUICKSORT (A, p, q - 1)
    RANDOMIZED-QUICKSORT (A, q + 1, r)
```

RANDOMIZED-PARTITION (A, p, r)

```
i = RANDOM(p, r)
exchange A[r] with A[i]
return PARTITION (A, p, r)
```

What is the expected value of X if we choose the pivot element randomly?

Lemma: Let X be the number of comparisons performed in line 4 of PARTITION over the entire execution of QUICKSORT on an n-element array. Then the running time of QUICKSORT is $O(n + X)$.

Randomized Quicksort (Running Time)

Check out Appendix C

- Denote the sorted elements of the array by $z_1, z_2, \ldots, z_n$
- Let $X_{ij} = I\{z_i \text{ is compared to } z_j\}$
- Then, $X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$ and $E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Pr\{z_i \text{ is compared to } z_j\}$

- Let $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$
- $Pr\{z_i \text{ is compared to } z_j\} = Pr\{z_i \text{ or } z_j \text{ is first pivot chosen from } Z_{ij}\} = \frac{2}{j-i+1}$
- So, $E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} < \sum_{i=1}^{n-1} \sum_{k=1}^{n-1} \frac{2}{k} = \sum_{i=1}^{n-1} O(\log n) = O(n \log n)$

Today's Lecture

- More probabilistic analysis and randomized algorithms
  - Hash tables
  - Bucket Sort
  - Binary Trees

Dynamic Set Data Structures

- Dynamic set K with keys drawn from the universe $U = \{0, 1, \ldots, m\}$
- Support dictionary operations given set K and key k :
  - INSERT(K, k)
  - SEARCH(K, k)
  - DELETE(K, k)
- What data structure should we use in order to store these keys?
  - Stack?
  - Queue?
  - Linked list?
- What would the running time of each operation be?

Dynamic Set Data Structures

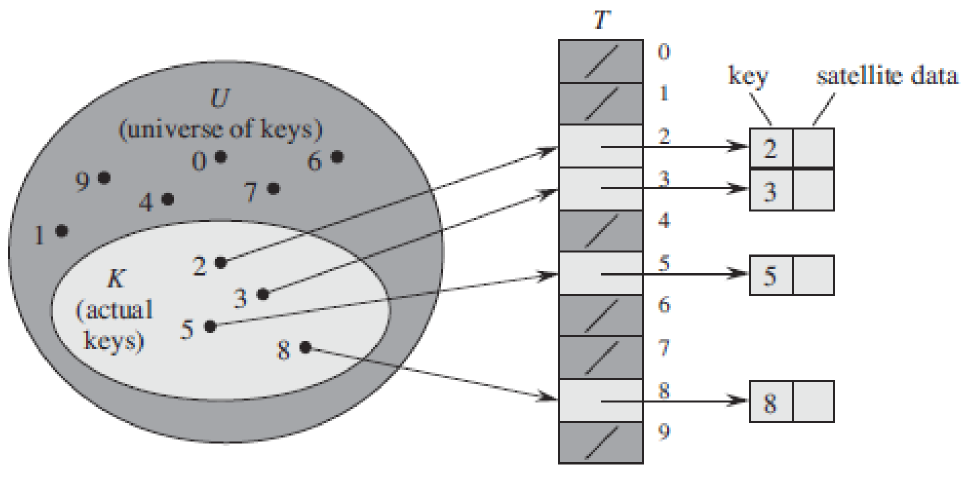
- Support dictionary operations:
  - INSERT(K, k)
  - SEARCH(K, k)
  - DELETE(K, k)

DIRECT-ADDRESS-SEARCH(T, k)
  return $T[k]$

DIRECT-ADDRESS-INSERT(T, x)
  $T[x.key] = x$

DIRECT-ADDRESS-DELETE(T, x)
  $T[x.key] =$ NIL

- Direct addressing into table T?
- How fast are the operations?
  - They all take $O(1)$ time!
- What issues may arise?
  - Size of array needs to be $|U|$!



Dynamic Set Data Structures

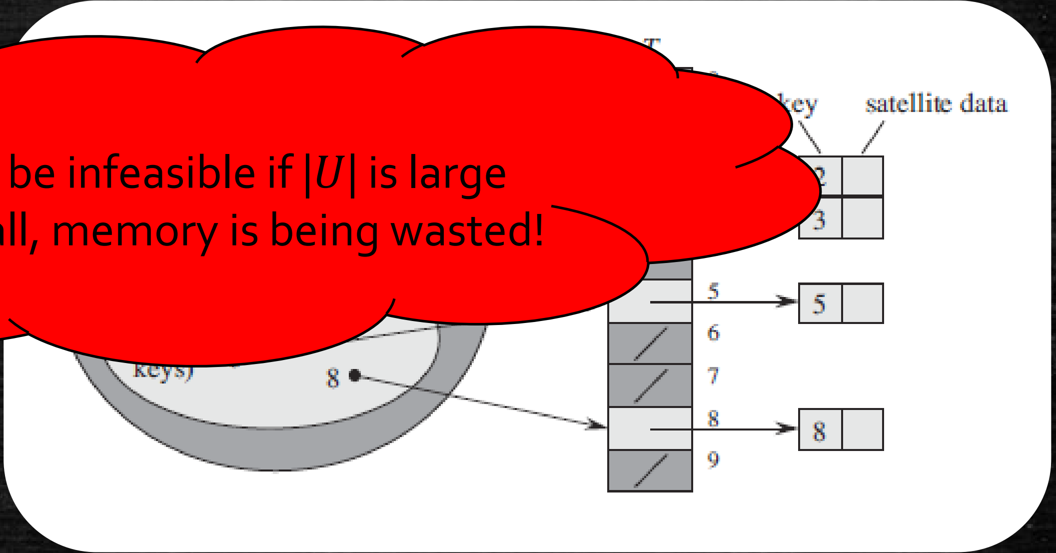
- Support dictionary operations:
  - INSERT(K, k)
  - SEARCH(K, k)
  - DELETE(K, k)

DIRECT-ADDRESS-SEARCH(T, k)
1 **return** $T[k]$

DIRECT-ADDRESS-INSERT(T, x)
1 $T[x.key] = x$

DIRECT-ADDRESS-DELETE(T, x)
1 $T[x.key] =$ NIL

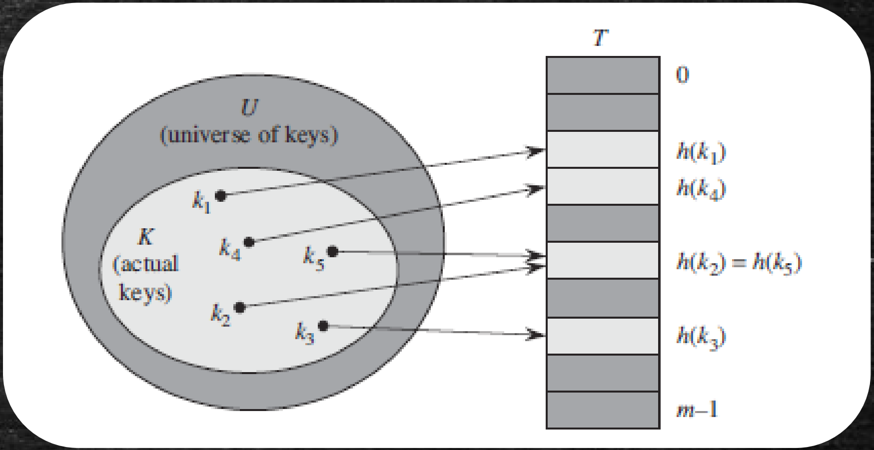
- Direct addressing into table T?
- How fast are the o
  - They all take $O(1)$ t
- What issues may arise?
  - Size of array needs to be $|U|$!



1. This may be infeasible if $|U|$ is large
2. If $|K|$ is small, memory is being wasted!

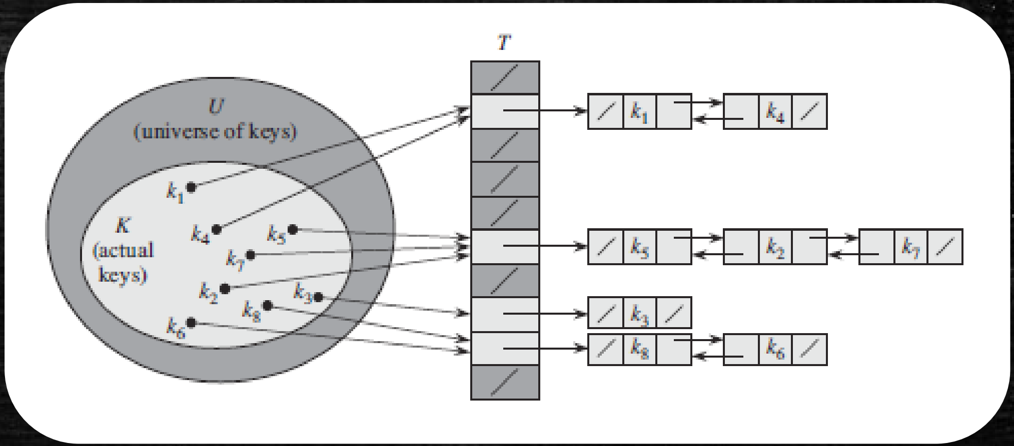
Hash Tables

- Use a hash function h
  - Function h maps U to slots of hash table
  - Given key k, compute the slot $h(k)$
  - This reduces the required table size



- But what if we get a collision?
  - Two distinct keys could be mapped to the same slot
  - How can we try to avoid this?
  - The function needs to be deterministic

- We can address that using chaining
  - Place colliding keys to same linked list
  - How does this affect the running time?



Hash Tables

- Use a hash function h
  - Function h maps U to slots of hash table
  - Given key k, compute the slot $h(k)$
  - This reduces the required table size

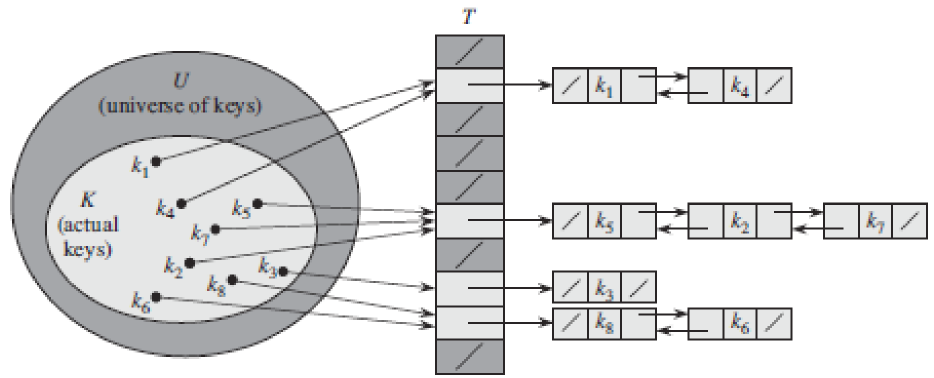
```
CHAINED-HASH-INSERT(T, x)
  insert x at the head of list T[h(x.key)]

CHAINED-HASH-SEARCH(T, k)
  search for an element with key k in list T[h(k)]

CHAINED-HASH-DELETE(T, x)
  delete x from the list T[h(x.key)]
```

- But what if we get a collision?
  - Two distinct keys could be mapped to the same slot
  - How can we try to avoid this?
  - The function needs to be deterministic

- We can address that using chaining
  - Place colliding keys to same linked list
  - How does this affect the running time?



Hash Tables (Running Time)

- Given a hash table with m slots that stores n elements:
  - Worst case running time for searching is $\Theta(n)$ plus time to compute hash function
  - This is no better than the time achieved by a single linked list…
  - Simple uniform hashing: any element is equally likely to hash into any of the slots
- What about the average-case running time for search?
  - Let n/m be the load factor α for hash table T
  - Let n_j be the length of the list $T[j]$ for $j \in \{0, 1, \dots, m-1\}$
  - The expected value of n_j for uniform hashing is $E[n_j] = \alpha$
  - Assume that computing the hash value $h(k)$ takes $O(1)$ time

Theorem 11.1
In a hash table in which collisions are resolved by chaining, an $\boxed{\text{unsuccessful}}$ search takes average-case time $\Theta(1+\alpha)$, under the assumption of simple uniform hashing.

Hash Tables (Running Time)

- Given a hash table with m slots that stores n elements:
  - Worst case running time for searching is $\Theta(n)$ plus time to compute hash function
  - This is no better than the time achieved by a single linked list…
  - Simple uniform hashing: any element is equally likely to hash into any of the slots
- What about the average-case running time for search?
  - Let n/m

Number of examined elements is: $X = \sum_{j=1}^{m} \frac{1}{m}(1 + n_j)$

So, $\mathrm{E}[X] = \sum_{j=1}^{m} \frac{1}{m}(1 + \mathrm{E}[n_j]) = \sum_{j=1}^{m} \frac{1}{m}\left(1 + \frac{n}{m}\right) = 1 + \frac{n}{m}$

Theorem 11.1

In a hash table in which collisions are resolved by chaining, an unsuccessful search takes average-case time $\Theta(1+\alpha)$, under the assumption of simple uniform hashing.

Hash Tables (Running Time)

- Given a hash table with m slots that stores n elements:
  - Worst case running time for searching is $\Theta(n)$ plus time to compute hash function
  - This is no better than the time achieved by a single linked list…
  - Simple uniform hashing: any element is equally likely to hash into any of the slots
- What about the average-case running time for search?
  - Let n/m be the load factor α for hash table T
  - Let n_j be the length of the list $T[j]$ for $j \in \{0, 1, \dots, m-1\}$
  - The expected value of n_j for uniform hashing is $E[n_j] = \alpha$
  - Assume that computing the hash value $h(k)$ takes $O(1)$ time

Theorem 11.2

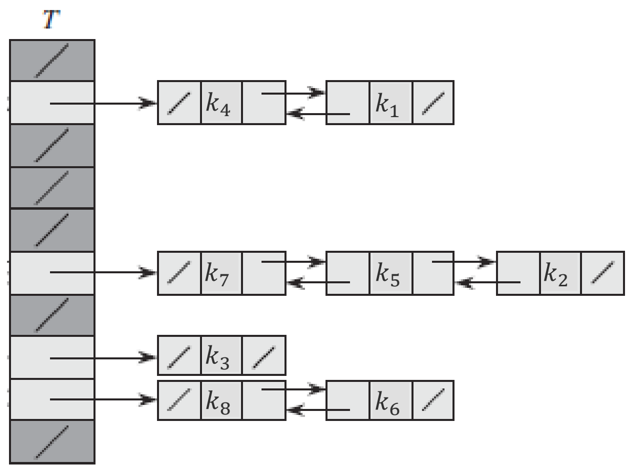
In a hash table in which collisions are resolved by chaining, a $\boxed{\text{successful}}$ search takes average-case time $\Theta(1+\alpha)$, under the assumption of simple uniform hashing.

Hash Tables (Running Time)

- For keys k_i and k_j we define indicator variable $X_{ij} = \mathbb{I}\{h(k_i) = h(k_j)\}$
- For simple uniform hashing, we get $\Pr\{h(k_i) = h(k_j)\} = 1/m$
- Assume that element being searched for is equally likely to be any of the n elements
- Expected number of elements examined in a successful search is:

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

Verify this for the following instance:



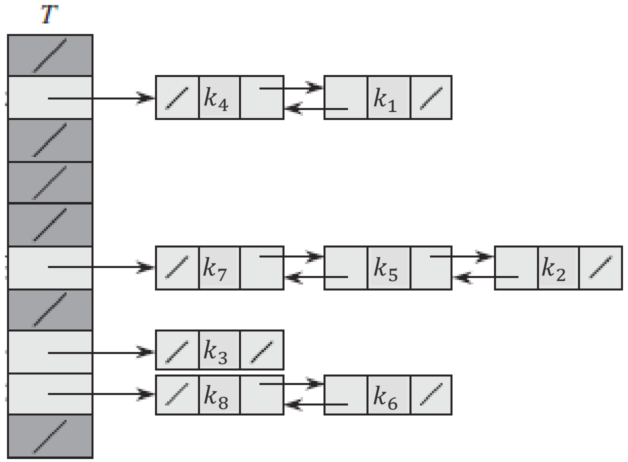
Hash Tables (Running Time)

- For keys k_i and k_j we define indicator variable $X_{ij} = \mathbb{I}\{h(k_i) = h(k_j)\}$
- For simple unifor $\} = 1/m$
- Assume that to be any of the n elements
- Expected num essful search is:

For simplicity, this assumes that k_i is the key of the i-th element to be added to the hash table!

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

Verify this for the following instance:



Hash Tables (Running Time)

- For keys k_i and k_j we define indicator variable $X_{ij} = \mathbb{I}\{h(k_i) = h(k_j)\}$
- For simple uniform hashing, we get $\Pr\{h(k_i) = h(k_j)\} = 1/m$
- Assume that element being searched for is equally likely to be any of the n elements
- Expected number of elements examined in a successful search is:

$$\begin{aligned}
\mathrm{E}\left[\frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n} X_{ij}\right)\right]
&= \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n} \mathrm{E}\left[X_{ij}\right]\right) \quad \text{(by linearity of expectation)} \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n} \frac{1}{m}\right) \\
&= 1+\frac{1}{nm}\sum_{i=1}^{n}(n-i)
\end{aligned}$$

Hash Tables (Running Time)

$$
\begin{aligned}
\mathrm{E}\left[\frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}X_{ij}\right)\right] \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}\mathrm{E}\left[X_{ij}\right]\right) && \text{(by linearity of expectation)} \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}\frac{1}{m}\right) \\
&= 1+\frac{1}{nm}\sum_{i=1}^{n}(n-i) \\
&= 1+\frac{1}{nm}\left(\sum_{i=1}^{n}n-\sum_{i=1}^{n}i\right) \\
&= 1+\frac{1}{nm}\left(n^2-\frac{n(n+1)}{2}\right) && \text{(by equation (A.1))} \\
&= 1+\frac{n-1}{2m} \\
&= 1+\frac{\alpha}{2}-\frac{\alpha}{2n}\,.
\end{aligned}
$$