# UMC 201: Data Structures & Algorithms

## Lecture 8: Hash tables; Universal hashing; Pairwise independent hash functions

Indian Institute of Science

# Recap

- In the last lecture, we discussed several elementary data structures such as *arrays*, *matrices*, *stacks*, *queues*, *linked lists*, *rooted trees*.
- These data structures help us store dynamic sets and support dictionary operations such as *insert*, *delete* and *search*.

# Recap

- In the last lecture, we discussed several elementary data structures such as *arrays*, *matrices*, *stacks*, *queues*, *linked lists*, *rooted trees*.
- These data structures help us store dynamic sets and support dictionary operations such as *insert*, *delete* and *search*.

| | Insert | Delete | Search |
|---|---|---|---|
| Unsorted array | $\Theta(1)$ | $\Theta(n)$ | $\Theta(n)$ |
| Sorted array | $\Theta(n)$ | $\Theta(n)$ | $\Theta(\log n)$ |
| Doubly Linked list | $\Theta(1)$ | $\Theta(1)$ | $\Theta(n)$ |

# Recap

- In the last lecture, we discussed several elementary data structures such as *arrays*, *matrices*, *stacks*, *queues*, *linked lists*, *rooted trees*.
- These data structures help us store dynamic sets and support dictionary operations such as *insert*, *delete* and *search*.
- Question. Is there a data structure that supports insertion, deletion, search – *all in* $\Theta(1)$ *time*?

# Recap

- In the last lecture, we discussed several elementary data structures such as *arrays*, *matrices*, *stacks*, *queues*, *linked lists*, *rooted trees*.
- These data structures help us store dynamic sets and support dictionary operations such as *insert*, *delete* and *search*.
- Question. Is there a data structure that supports insertion, deletion, search – *all in* $\Theta(1)$ *time*?
- A direct-address table, and more generally, a hash table provide such effective data structures.

# Today's lecture

- In today's lecture, we'll discuss *hash tables* and hash functions, which are used to implement hash tables.
- Although search for an element in a hash table can take $\Theta(n)$ *in the worst case*, the *expected (average) time* to search for an element is $\Theta(1)$ if "nice" hash functions are used to implement hash tables.

# Today's lecture

- In today's lecture, we'll discuss *hash tables* and hash functions, which are used to implement hash tables.
- Although search for an element in a hash table can take $\Theta(n)$ *in the worst case*, the *expected (average) time* to search for an element is $\Theta(1)$ if "nice" hash functions are used to implement hash tables.
- Indeed, hashing performs *extremely well in practice*. The built-in dictionaries of Python are implemented using hash tables.

# Direct-address tables

# Direct-address tables

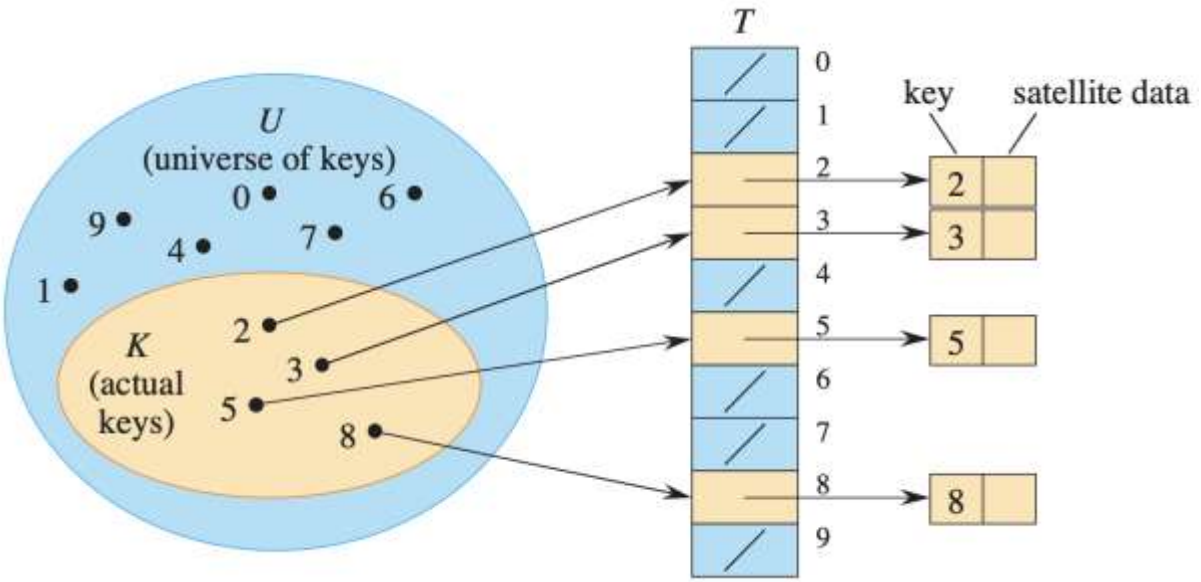
- Direct addressing is a simple technique that works well when the universe $U$ of keys is *small*.
- Suppose that an application needs a dynamic set in which each element has a *distinct key* from the universe $U = \{0,1, \ldots, m-1\}$, where $m$ is *small*.

# Direct-address tables

- Direct addressing is a simple technique that works well when the universe $U$ of keys is *small*.
- Suppose that an application needs a dynamic set in which each element has a *distinct key* from the universe $U = \{0,1, \ldots, m-1\}$, where $m$ is *small*.
- To represent the dynamic set, we can use an array, called direct-address table, in which every position, or slot, corresponds to a key in the universe $U$.

# Direct-address tables

- Direct addressing is a simple technique that works well when the universe $U$ of keys is *small*.
- Suppose that an application needs a dynamic set in which each element has a *distinct key* from the universe $U = \{0,1, \ldots, m-1\}$, where $m$ is *small*.



# Direct-address tables

- Direct addressing is a simple technique that works well when the universe $U$ of keys is *small*.
- Suppose that an application needs a dynamic set in which each element has a *distinct key* from the universe $U = \{0,1, \ldots, m-1\}$, where $m$ is *small*.

```
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
```

- Each of the dictionary operations takes $\Theta(1)$ time.

# Direct-address tables: Downside

- The downside of direct addressing is apparent: if the universe $U$ is *large*, storing a table $T$ of size $|U|$ may be *impractical*.
- Furthermore, the set $K$ of keys *actually stored may be so small* relative to $U$ that most of the space allocated for $T$ would be *wasted*.

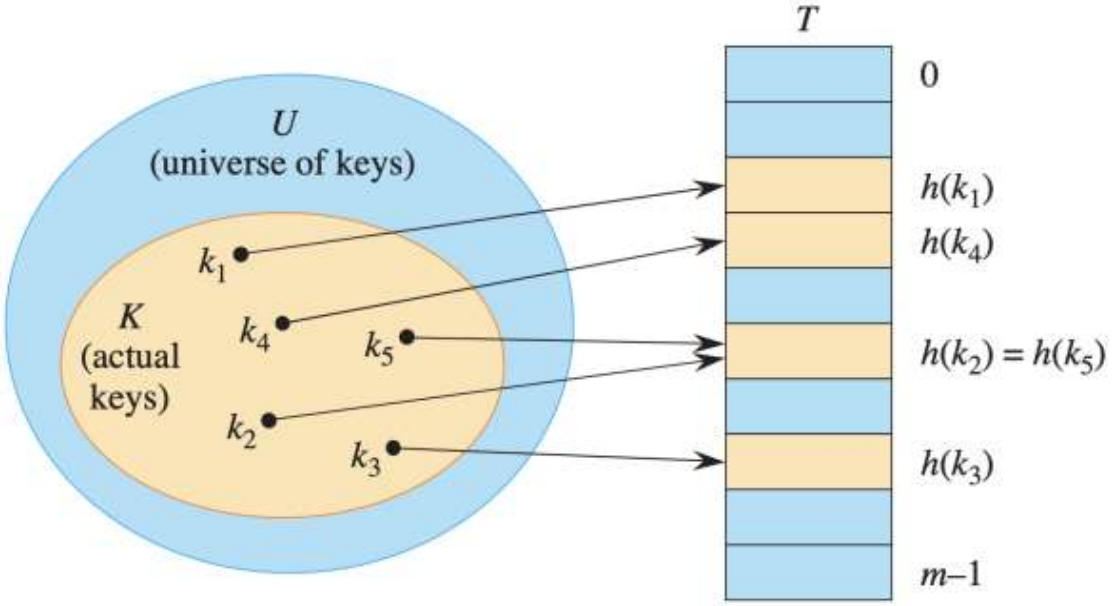
# Direct-address tables: Downside

- The downside of direct addressing is apparent: if the universe $U$ is *large*, storing a table $T$ of size $|U|$ may be *impractical*.
- Furthermore, the set $K$ of keys *actually stored may be so small* relative to $U$ that most of the space allocated for $T$ would be *wasted*.
- When the $|K|$ is small relative to $|U|$, hash tables become an *effective alternative* to direct-addressing.
- The storage requirement reduces to $\Theta(|K|)$ and a search takes $O(1)$ time *in the expected (average) case*.

# Hash tables

# Hash tables

- With direct addressing, an element with key $k$ is stored in slot $k$, but with hashing, we use a hash function $h: U \to \{0,1, \ldots, m-1\}$ to *compute the slot number* from the key $k$, so that the element goes into slot $h(k)$ of a hash table $T[0: m-1]$.



# Hash tables

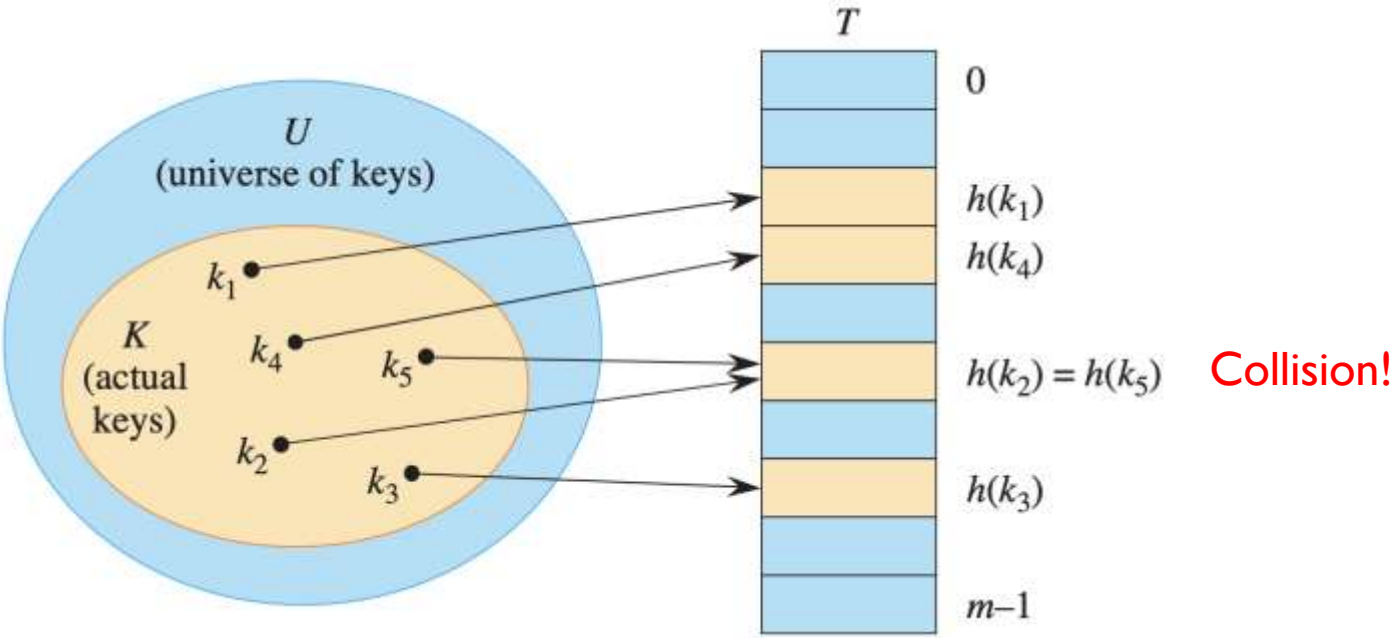
- With direct addressing, an element with key $k$ is stored in slot $k$, but with hashing, we use a hash function $h: U \to \{0,1, \ldots, m-1\}$ to *compute the slot number* from the key $k$, so that the element goes into slot $h(k)$ of a hash table $T[0: m-1]$.
- Typically, $m$ is *much less than* $|U|$. We say that an element with key $k$ hashes to slot $h(k)$; in other words, $h(k)$ is the hash value of key $k$.

# Hash tables

- With direct addressing, an element with key $k$ is stored in slot $k$, but with hashing, we use a hash function $h: U \to \{0,1, \ldots, m-1\}$ to *compute the slot number* from the key $k$, so that the element goes into slot $h(k)$ of a hash table $T[0: m-1]$.
- Typically, $m$ is *much less than* $|U|$. We say that an element with key $k$ hashes to slot $h(k)$; in other words, $h(k)$ is the hash value of key $k$.
- Example. A simple, but *not particularly good*, example of a hash function is $h(k) = k \mod m$.

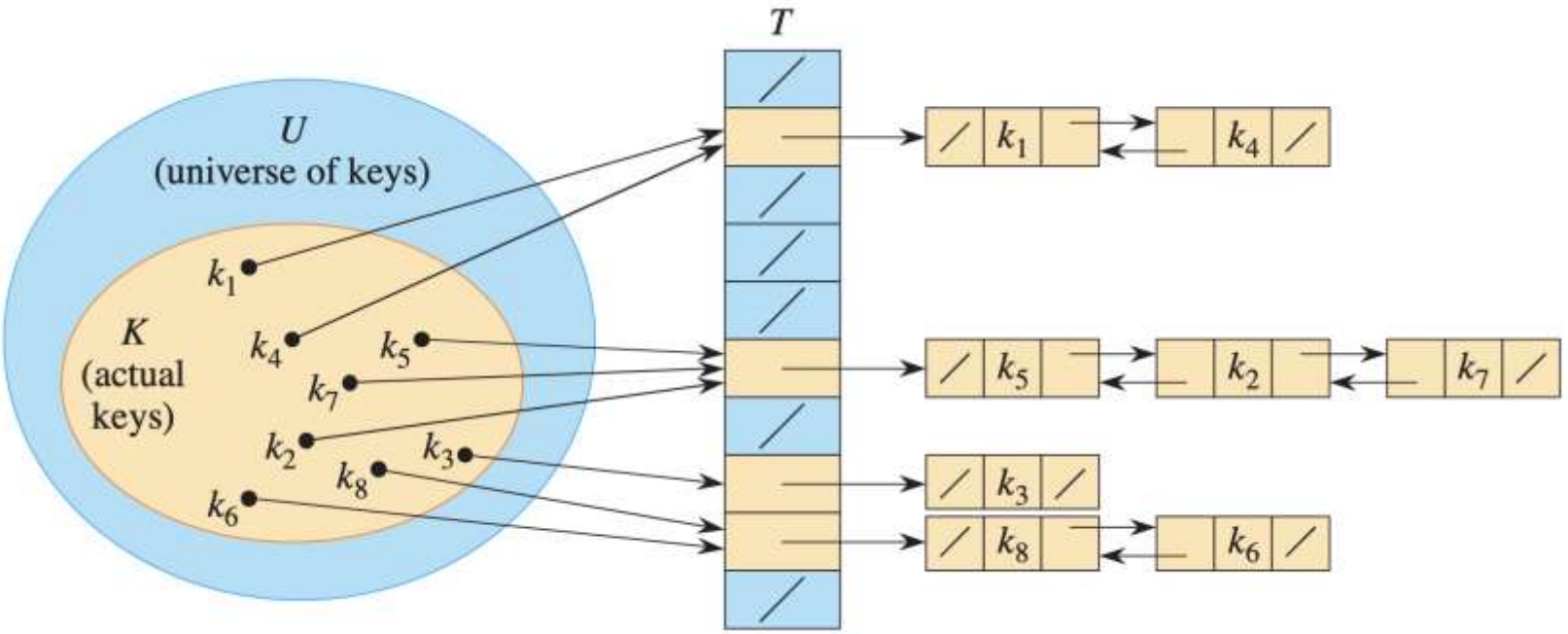
# Hash tables: Collisions

- There is *one problem*: Two keys may hash to the *same slot*. We call this situation collision.



# Hash tables: Collisions

- There is *one problem*: Two keys may hash to the *same slot*. We call this situation collision.
- A simple way to *resolve collision* is called chaining.



# Hash tables: Collisions

- There is *one problem*: Two keys may hash to the *same slot*. We call this situation collision.
- A simple way to *resolve collision* is called chaining.
- Idea. Each nonempty slot *points to a linked list*, and all the elements that hash to the *same slot* go into the slot's linked list.

# Hash tables: Operations

- There is *one problem*: Two keys may hash to the *same slot*. We call this situation collision.
- A simple way to *resolve collision* is called chaining.

```
CHAINED-HASH-INSERT(T, x)
1  LIST-PREPEND(T[h(x.key)], x)

CHAINED-HASH-SEARCH(T, k)
1  return LIST-SEARCH(T[h(k)], k)

CHAINED-HASH-DELETE(T, x)
1  LIST-DELETE(T[h(x.key)], x)
```

Obs. The worst-case running time for an *insertion* is $\Theta(1)$.

# Hash tables: Operations

- There is *one problem*: Two keys may hash to the *same slot*. We call this situation collision.
- A simple way to *resolve collision* is called chaining.

```
CHAINED-HASH-INSERT(T, x)
1  LIST-PREPEND(T[h(x.key)], x)

CHAINED-HASH-SEARCH(T, k)
1  return LIST-SEARCH(T[h(k)], k)

CHAINED-HASH-DELETE(T, x)
1  LIST-DELETE(T[h(x.key)], x)
```

Obs. The worst-case running time for a *deletion* is $\Theta(1)$.

# Hash tables: Operations

- There is *one problem*: Two keys may hash to the *same slot*. We call this situation collision.
- A simple way to *resolve collision* is called chaining.

```
CHAINED-HASH-INSERT(T, x)
1  LIST-PREPEND(T[h(x.key)], x)

CHAINED-HASH-SEARCH(T, k)
1  return LIST-SEARCH(T[h(k)], k)

CHAINED-HASH-DELETE(T, x)
1  LIST-DELETE(T[h(x.key)], x)
```

Obs. The worst-case runtime for a search is *proportional to the length of the linked list* pointed to by the slot $h(k)$.

# Hash tables: Operations

- There is *one problem*: Two keys may hash to the *same slot*. We call this situation collision.
- A simple way to *resolve collision* is called chaining.

```
CHAINED-HASH-INSERT(T, x)
1  LIST-PREPEND(T[h(x.key)], x)

CHAINED-HASH-SEARCH(T, k)
1  return LIST-SEARCH(T[h(k)], k)

CHAINED-HASH-DELETE(T, x)
1  LIST-DELETE(T[h(x.key)], x)
```

Question. How do we control the length of a linked list?

# Hash tables: Operations

- There is *one problem*: Two keys may hash to the *same slot*. We call this situation collision.
- A simple way to *resolve collision* is called chaining.

```
CHAINED-HASH-INSERT(T, x)
1  LIST-PREPEND(T[h(x.key)], x)

CHAINED-HASH-SEARCH(T, k)
1  return LIST-SEARCH(T[h(k)], k)

CHAINED-HASH-DELETE(T, x)
1  LIST-DELETE(T[h(x.key)], x)
```

Question. How do we control the length of a linked list?

Answer. By *choosing the hash function $h$ carefully*, we can ensure that the *expected length* of a linked list is *small*.

# Hash functions

- Question. What makes a hash function *good* or *useful*?
- Well, a hash function is good or useful if the *lengths of the linked lists*, attached to the slots of the hash table, are *small*.

# Hash functions

- Question. What makes a hash function *good* or *useful*?
- Well, a hash function is good or useful if the *lengths of the linked lists*, attached to the slots of the hash table, are *small*.
- Question. How do we get such hash functions?

# Hash functions

- Question. What makes a hash function *good* or *useful*?
- Well, a hash function is good or useful if the *lengths of the linked lists*, attached to the slots of the hash table, are *small*.
- Question. How do we get such hash functions?
- Issue. We want $m = \Theta(|K|) \ll |U|$ to make a hash table *space efficient*. If we use a *fixed* (static) hash function, then (by a simple averaging argument) an adversary can choose a $K$ such that *all the keys in* $K$ *map to the same slot*, thereby making the list size $|K|$.

# Hash functions

- Question. What makes a hash function *good* or *useful*?
- Well, a hash function is good or useful if the *lengths of the linked lists*, attached to the slots of the hash table, are *small*.
- Question. How do we get such hash functions?
- Idea. Intuitively, if a hash function maps the keys *uniformly and independently at random* among the slots of the hash table, then we *expect* the lengths of the linked lists to be small.

# Hash functions

- Question. What makes a hash function *good* or *useful*?
- Well, a hash function is good or useful if the *lengths of the linked lists*, attached to the slots of the hash table, are *small*.
- Question. How do we get random hash functions?

# Hash functions

- Question. What makes a hash function *good* or *useful*?
- Well, a hash function is good or useful if the *lengths of the linked lists*, attached to the slots of the hash table, are *small*.
- Question. How do we get random hash functions?
- Idea. If we work with a *family of hash functions*, then by picking a hash function $h$ *randomly* from the family we can hope that $h$ will keep the *expected size* of a list small, thereby keeping search time under control.

# Hash function family

- Definition. Let $\mathcal{H}$ be a finite family (set) of hash functions that map a universe $U$ of keys into the range $\{0,1,\ldots,m-1\}$. The family $\mathcal{H}$ is called
  - uniform if for *any* key $k \in U$ and for *any* slot $q \in \{0,1,\ldots,m-1\}$, $\Pr_{h\in\mathcal{H}}(h(k)=q)=1/m$.

# Hash function family

- Definition. Let $\mathcal{H}$ be a finite family (set) of hash functions that map a universe $U$ of keys into the range $\{0,1,\dots,m-1\}$. The family $\mathcal{H}$ is called
  - uniform if for *any* key $k \in U$ and for *any* slot $q \in \{0,1,\dots,m-1\}$, $\Pr_{h\in\mathcal{H}}(h(k)=q)=1/m$.
  - universal if for *any two distinct keys* $k_1, k_2 \in U$, $\Pr_{h\in\mathcal{H}}\big(h(k_1)=h(k_2)\big)\le 1/m$.

# Hash function family

- Definition. Let $\mathcal{H}$ be a finite family (set) of hash functions that map a universe $U$ of keys into the range $\{0,1,\dots,m-1\}$. The family $\mathcal{H}$ is called
  - uniform if for *any* key $k \in U$ and for *any* slot $q \in \{0,1,\dots,m-1\}$, $\Pr_{h\in\mathcal{H}}(h(k)=q)=1/m$.
  - universal if for *any two distinct keys* $k_1,k_2 \in U$, $\Pr_{h\in\mathcal{H}}\big(h(k_1)=h(k_2)\big)\le 1/m$.
  - $d$-independent if for any *distinct keys* $k_1,\dots,k_d \in U$ and for any slots $q_1,\dots,q_d$ (*not necessarily distinct*) $\Pr_{h\in\mathcal{H}}(h(k_i)=q_i,\forall i\in[d])=1/m^d$.

# Hash function family

- Obs. Let $\mathcal{H}$ be a $d$-independent family of hash functions for $d \geq 2$, i.e., for any *distinct keys* $k_1, \ldots, k_d \in U$ and for any slots $q_1, \ldots, q_d$, $\Pr_{h \in \mathcal{H}}(h(k_i) = q_i, \forall i \in [d]) = 1/m^d$. Then, $\mathcal{H}$ is both a uniform and universal family of hash functions.

# Hash function family

- Obs. Let $\mathcal{H}$ be a $d$-independent family of hash functions for $d \geq 2$, i.e., for any *distinct keys* $k_1, \ldots, k_d \in U$ and for any slots $q_1, \ldots, q_d$, $\Pr_{h \in \mathcal{H}}(h(k_i) = q_i, \forall i \in [d]) = 1/m^d$. Then, $\mathcal{H}$ is both a uniform and universal family of hash functions.

- Thus, 2-independent (a.k.a. pairwise independent) hash function families are both *uniform* and *universal*, i.e., $\Pr(h(k) = q)) = 1/m$, and for $k_1 \neq k_2$,

$$\Pr\big(h(k_1) = h(k_2)\big) = 1/m.$$

# Hash function family

- Obs. Let $\mathcal{H}$ be a $d$-independent family of hash functions for $d \geq 2$, i.e., for any *distinct keys* $k_1, \ldots, k_d \in U$ and for any slots $q_1, \ldots, q_d$, $\Pr_{h \in \mathcal{H}}(h(k_i) = q_i, \forall i \in [d]) = 1/m^d$. Then, $\mathcal{H}$ is both a uniform and universal family of hash functions.

- Let us now discuss the *usefulness* of uniform, universal and $d$-independent hash function families.

# Uniform hash function family

- Let $|K| = n$ be the number of keys being stored in a hash table with $m$ slots. If $\mathcal{H}$ is a *uniform* hash function family, then for *any* key $k \in K$ and for *any* slot $q \in \{0,1,\ldots,m-1\}$, $\Pr_{h \in \mathcal{H}}(h(k) = q) = 1/m$.

# Uniform hash function family

- Let $|K| = n$ be the number of keys being stored in a hash table with $m$ slots. If $\mathcal{H}$ is a *uniform* hash function family, then for *any* key $k \in K$ and for *any* slot $q \in \{0,1,\ldots,m-1\}$, $\Pr_{h \in \mathcal{H}}(h(k) = q) = 1/m$.
- For a slot $q$, let $n_q$ be the *size of the corresponding list*. For a key $k$, let the *indicator variable* $x_k = 1$ *iff* $h(k) = q$.

# Uniform hash function family

- Let $|K| = n$ be the number of keys being stored in a hash table with $m$ slots. If $\mathcal{H}$ is a *uniform* hash function family, then for *any* key $k \in K$ and for *any* slot $q \in \{0,1,\dots,m-1\}$, $\Pr_{h\in\mathcal{H}}(h(k) = q) = 1/m$.
- For a slot $q$, let $n_q$ be the *size of the corresponding list*. For a key $k$, let the *indicator variable* $x_k = 1$ *iff* $h(k) = q$. Then, $n_q = \sum_{k\in K} x_k$, and $E[n_q] = n/m$ (by *linearity of expectation*) as $E[x_k] = \Pr(x_k = 1) = 1/m$.

# Uniform hash function family

- Let $|K| = n$ be the number of keys being stored in a hash table with $m$ slots. If $\mathcal{H}$ is a *uniform* hash function family, then for *any* key $k \in K$ and for *any* slot $q \in \{0,1,\dots,m-1\}$, $\Pr_{h\in\mathcal{H}}(h(k) = q) = 1/m$.
- For a slot $q$, let $n_q$ be the *size of the corresponding list*. For a key $k$, let the *indicator variable* $x_k = 1$ *iff* $h(k) = q$. Then, $n_q = \sum_{k\in K} x_k$, and $E[n_q] = n/m$ (by *linearity of expectation*) as $E[x_k] = \Pr(x_k = 1) = 1/m$. The ratio $n/m$ is called the load factor.

# Uniform hash function family

- Let $|K| = n$ be the number of keys being stored in a hash table with $m$ slots. If $\mathcal{H}$ is a *uniform* hash function family, then for *any* key $k \in K$ and for *any* slot $q \in \{0,1,\ldots,m-1\}$, $\Pr_{h \in \mathcal{H}}(h(k) = q) = 1/m$.

- Typically, we choose $m > n$ and hence the load factor $n/m$, which is the expected length of a list, is *less than* $1$ if the hash function family $\mathcal{H}$ is uniform.

# Universal hash function family

- If $\mathcal{H}$ is a *universal* hash function family, then for *any two distinct* $k_1, k_2 \in U$, $\Pr_{h \in \mathcal{H}}\big(h(k_1) = h(k_2)\big) \leq 1/m$.
- For an *arbitrarily fixed* $k \in K$, let $n_k$ be the size of the list that $k$ belongs to. For $\ell \in K \backslash \{k\}$, let the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$.

# Universal hash function family

- If $\mathcal{H}$ is a *universal* hash function family, then for *any two distinct* $k_1, k_2 \in U$, $\Pr_{h \in \mathcal{H}}\big(h(k_1) = h(k_2)\big) \leq 1/m$.
- For an *arbitrarily fixed* $k \in K$, let $n_k$ be the size of the list that $k$ belongs to. For $\ell \in K \backslash \{k\}$, let the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$.
- Then, $n_k = \sum_{\ell \in K \backslash \{k\}} y_\ell + 1$, and $E[n_k] \leq (n-1)/m + 1$ as $E[y_\ell] = \Pr(y_\ell = 1) \leq 1/m$.

# Universal hash function family

- If $\mathcal{H}$ is a *universal* hash function family, then for *any two distinct* $k_1, k_2 \in U$, $\Pr_{h \in \mathcal{H}}\big(h(k_1) = h(k_2)\big) \leq 1/m$.
- For an *arbitrarily fixed* $k \in K$, let $n_k$ be the size of the list that $k$ belongs to. For $\ell \in K \backslash \{k\}$, let the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$.
- Then, $n_k = \sum_{\ell \in K \backslash \{k\}} y_\ell + 1$, and $E[n_k] \leq (n-1)/m + 1$ as $E[y_\ell] = \Pr(y_\ell = 1) \leq 1/m$.
- Thus, for any arbitrarily fixed $k$, the *expected length* of the list that $k$ belongs to is $\Theta(1)$ if the hash family $\mathcal{H}$ is universal and $m > n$.

# Universal hash function family

- Suppose $\mathcal{H}$ is a *universal* hash function family and $m = 2n$. Then, $E[n_k] < 1.5$ for an arbitrarily fixed $k$.
- Then, by Markov's inequality, $\Pr(n_k \geq 150) \leq \frac{1}{100}$.

- Markov's inequality. *(recall)* If $X$ is a *nonnegative* random variable and $a > 0$, then $\Pr(X \geq a) \leq \frac{E[X]}{a}$.

# Universal hash function family

- Suppose $\mathcal{H}$ is a *universal* hash function family and $m = 2n$. Then, $E[n_k] < 1.5$ for an arbitrarily fixed $k$.
- Then, by Markov's inequality, $\Pr(n_k \geq 150) \leq \frac{1}{100}$.
- Suppose, the *indicator variable* $z_k = 1$ *iff* $n_k \geq 150$. Then, $E[z_k] = \Pr(z_k = 1) \leq \frac{1}{100}$.

# Universal hash function family

- Suppose $\mathcal{H}$ is a *universal* hash function family and $m = 2n$. Then, $E[n_k] < 1.5$ for an arbitrarily fixed $k$.
- Then, by Markov's inequality, $\Pr(n_k \geq 150) \leq \frac{1}{100}$.
- Suppose, the *indicator variable* $z_k = 1$ *iff* $n_k \geq 150$. Then, $E[z_k] = \Pr(z_k = 1) \leq \frac{1}{100}$.
- Let $B$ be the number of $k \in K$ such that $n_k \geq 150$. Then, $B = \sum_{k \in K} z_k$. Hence, $E[B] \leq \frac{n}{100}$ as $|K| = n$.

# Universal hash function family

- Suppose $\mathcal{H}$ is a *universal* hash function family and $m = 2n$. Then, $E[n_k] < 1.5$ for an arbitrarily fixed $k$.
- Then, by Markov's inequality, $\Pr(n_k \geq 150) \leq \frac{1}{100}$.
- Suppose, the *indicator variable* $z_k = 1$ *iff* $n_k \geq 150$. Then, $E[z_k] = \Pr(z_k = 1) \leq \frac{1}{100}$.
- Let $B$ be the number of $k \in K$ such that $n_k \geq 150$. Then, $B = \sum_{k \in K} z_k$. Hence, $E[B] \leq \frac{n}{100}$ as $|K| = n$.
- By Markov's inequality, $\Pr\left(B \geq \frac{n}{10}\right) \leq \frac{1}{10}$.

# Universal hash function family

- Suppose $\mathcal{H}$ is a *universal* hash function family and $m = 2n$. Then, $E[n_k] < 1.5$ for an arbitrarily fixed $k$.
- Then, by Markov's inequality, $\Pr(n_k \geq 150) \leq \frac{1}{100}$.

- This means, *with probability at least* $0.9$ over the *random choice of* $h \in \mathcal{H}$, more than $90\%$ of the keys $k \in K$ *belong to lists of length at most* $150$.

# Universal hash function family

- Suppose $\mathcal{H}$ is a *universal* hash function family and $m = 2n$. Then, $E[n_k] < 1.5$ for an arbitrarily fixed $k$.
- Then, by Markov's inequality, $\Pr(n_k \geq 150) \leq \frac{1}{100}$.

- This means, *with probability at least* $0.9$ over the *random choice of* $h \in \mathcal{H}$, more than $90\%$ of the keys $k \in K$ *belong to lists of length at most* $150$.
- Question. Can we ensure that with high probability *all* keys belong to lists of constant size?

# $d$-independent hash function family

- Let $\mathcal{H}$ be a 3-independent hash function family, i.e., for any *distinct keys* $k_1, k_2, k_3 \in U$ and for any slots $q_1, q_2, q_3$ , $\Pr_{h \in \mathcal{H}}(h(k_i) = q_i, \forall i \in [3]) = 1/m^3$ . Suppose, $m = n^2$, where $n = |K|$.

# $d$-independent hash function family

- Let $\mathcal{H}$ be a 3-independent hash function family, i.e., for any *distinct keys* $k_1, k_2, k_3 \in U$ and for any slots $q_1, q_2, q_3$ , $\Pr_{h\in\mathcal{H}}(h(k_i) = q_i, \forall i \in [3]) = 1/m^3$ . Suppose, $m = n^2$, where $n = |K|$.
- For an *arbitrarily fixed* $k \in K$, let $n_k$ be the size of the list that $k$ belongs to. For $\ell \in K\backslash\{k\}$, let the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$. Let $Y_k = n_k - 1$.
- Then, $Y_k = \sum_{\ell\in K\backslash\{k\}} y_\ell$ , and $E[Y_k] = (n-1)/m$ as $E[y_\ell] = \Pr(y_\ell = 1) = 1/m$. (3-independent ⇒ universal)

# $d$-independent hash function family

- Let $\mathcal{H}$ be a 3-independent hash function family, i.e., for any *distinct keys* $k_1, k_2, k_3 \in U$ and for any slots $q_1, q_2, q_3$ , $\Pr_{h\in\mathcal{H}}(h(k_i) = q_i, \forall i \in [3]) = 1/m^3$ . Suppose, $m = n^2$, where $n = |K|$.
- For an *arbitrarily fixed* $k \in K$, let $n_k$ be the size of the list that $k$ belongs to. For $\ell \in K\backslash\{k\}$, let the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$. Let $Y_k = n_k - 1$.
- Then, $Y_k = \sum_{\ell\in K\backslash\{k\}} y_\ell$ , and $E[Y_k] = (n-1)/m$ as $E[y_\ell] = \Pr(y_\ell = 1) = 1/m$.
- We'll show that $\Pr(Y_k \geq 5) \leq 1/(10n)$.

# $d$-independent hash function family

- Let $\mathcal{H}$ be a 3-independent hash function family, i.e., for any *distinct keys* $k_1, k_2, k_3 \in U$ and for any slots $q_1, q_2, q_3$ , $\Pr_{h\in\mathcal{H}}(h(k_i) = q_i, \forall i \in [3]) = 1/m^3$ . Suppose, $m = n^2$, where $n = |K|$.
- For an *arbitrarily fixed* $k \in K$, let $n_k$ be the size of the list that $k$ belongs to. For $\ell \in K\backslash\{k\}$, let the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$. Let $Y_k = n_k - 1$.
- Then, $Y_k = \sum_{\ell\in K\backslash\{k\}} y_\ell$ , and $E[Y_k] = (n-1)/m$ as $E[y_\ell] = \Pr(y_\ell = 1) = 1/m$.
- By union bound, $\Pr(\exists k \in K\ s.t.\ Y_k \geq 5) \leq 1/10$.

# $d$-independent hash function family

- Let $\mathcal{H}$ be a 3-independent hash function family, i.e., for any *distinct keys* $k_1, k_2, k_3 \in U$ and for any slots $q_1, q_2, q_3$ , $\Pr_{h\in\mathcal{H}}(h(k_i) = q_i, \forall i \in [3]) = 1/m^3$ . Suppose, $m = n^2$, where $n = |K|$.
- For an *arbitrarily fixed* $k \in K$, let $n_k$ be the size of the list that $k$ belongs to. For $\ell \in K\backslash\{k\}$, let the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$. Let $Y_k = n_k - 1$.
- Then, $Y_k = \sum_{\ell\in K\backslash\{k\}} y_\ell$ , and $E[Y_k] = (n-1)/m$ as $E[y_\ell] = \Pr(y_\ell = 1) = 1/m$.
- By union bound, $\Pr(\exists k \in K\ s.t.\ n_k \geq 6) \leq 1/10$.

# $d$-independent hash function family

- Let $\mathcal{H}$ be a 3-independent hash function family, i.e., for any *distinct keys* $k_1, k_2, k_3 \in U$ and for any slots $q_1, q_2, q_3$ , $\Pr_{h\in\mathcal{H}}(h(k_i) = q_i, \forall i \in [3]) = 1/m^3$ . Suppose, $m = n^2$, where $n = |K|$.
- For an *arbitrarily fixed* $k \in K$, let $n_k$ be the size of the list that $k$ belongs to. For $\ell \in K\backslash\{k\}$, let the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$. Let $Y_k = n_k - 1$.
- Then, $Y_k = \sum_{\ell\in K\backslash\{k\}} y_\ell$ , and $E[Y_k] = (n-1)/m$ as $E[y_\ell] = \Pr(y_\ell = 1) = 1/m$.
- With probability at least $0.9$ *all list sizes* are $\leq 6$.

# Bounding $\Pr(Y_k \geq 5)$

- For $\ell \in K \backslash \{k\}$, the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$, $Y_k = \sum_{\ell \in K \backslash \{k\}} y_\ell$, & $E[Y_k] = (n-1)/m$ as $E[y_\ell] = \Pr(y_\ell = 1) = 1/m$. Recall, $m = n^2$.

- Chebyshev's inequality. Let $Y$ be a random variable. Then, for any $a > 0$, $\Pr(|Y - E[Y]| \geq a) \leq \frac{Var(Y)}{a^2}$.

# Bounding $\Pr(Y_k \geq 5)$

- For $\ell \in K \backslash \{k\}$, the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$, $Y_k = \sum_{\ell \in K \backslash \{k\}} y_\ell$, & $E[Y_k] = (n-1)/m$ as $E[y_\ell] = \Pr(y_\ell = 1) = 1/m$. Recall, $m = n^2$.

- Chebyshev's inequality. Let $Y$ be a random variable. Then, for any $a > 0$, $\Pr(|Y - E[Y]| \geq a) \leq \frac{Var(Y)}{a^2}$.
- Let $Y = Y_k$, and $a = 4n.E[Y_k]$. Then,

$$\Pr(Y_k \geq (1+4n)E[Y_k]) \leq \frac{Var(Y_k)}{16n^2 E[Y_k]^2}.$$

# Bounding $\Pr(Y_k \geq 5)$

- For $\ell \in K \backslash \{k\}$, the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$, $Y_k = \sum_{\ell \in K \backslash \{k\}} y_\ell$, & $E[Y_k] = (n-1)/m$ as $E[y_\ell] = \Pr(y_\ell = 1) = 1/m$. Recall, $m = n^2$.

- Chebyshev's inequality. Let $Y$ be a random variable. Then, for any $a > 0$, $\Pr(|Y - E[Y]| \geq a) \leq \frac{Var(Y)}{a^2}$.
- Let $Y = Y_k$, and $a = 4n.E[Y_k]$. Then,

$$\Pr(Y_k \geq 5) \leq \frac{Var(Y_k)}{16n^2 E[Y_k]^2}.$$

# Bounding $Var(Y_k)$

- For $\ell \in K \backslash \{k\}$, the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$, $Y_k = \sum_{\ell \in K \backslash \{k\}} y_\ell$, & $E[Y_k] = (n-1)/m$. Note that $Var(Y_k) = E\left[Y_k^2\right] - E[Y_k]^2$.

# Bounding $Var(Y_k)$

- For $\ell \in K\backslash\{k\}$, the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$, $Y_k = \sum_{\ell \in K\backslash\{k\}} y_\ell$, & $E[Y_k] = (n-1)/m$. Note that $Var(Y_k) = E\left[Y_k^2\right] - E[Y_k]^2$.
- $E\left[Y_k^2\right] = E[(\sum_\ell y_\ell)^2]$

# Bounding $Var(Y_k)$

- For $\ell \in K \backslash \{k\}$, the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$, $Y_k = \sum_{\ell \in K \backslash \{k\}} y_\ell$, & $E[Y_k] = (n-1)/m$. Note that $Var(Y_k) = E\left[Y_k^2\right] - E[Y_k]^2$.
- $E\left[Y_k^2\right] = E[(\sum_\ell y_\ell)^2]$

$$= \sum_\ell E\left[y_\ell^2\right] + 2 \sum_{\ell, j; \ell \neq j} E\left[y_\ell y_j\right]$$

# Bounding $Var(Y_k)$

- For $\ell \in K\backslash\{k\}$, the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$, $Y_k = \sum_{\ell \in K\backslash\{k\}} y_\ell$, & $E[Y_k] = (n-1)/m$. Note that $Var(Y_k) = E\left[Y_k^2\right] - E[Y_k]^2$.
- $E\left[Y_k^2\right] = E[(\sum_\ell y_\ell)^2]$

  $= \sum_\ell E\left[y_\ell^2\right] + 2\sum_{\ell,j;\ell\neq j} E\left[y_\ell y_j\right]$

  $= \sum_\ell E[y_\ell] + 2\sum_{\ell,j;\ell\neq j} E[y_\ell]E\left[y_j\right]$
- As $y_\ell$ is a Boolean variable, $y_\ell^2 = y_\ell$.

# Bounding $Var(Y_k)$

- For $\ell \in K\backslash\{k\}$, the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$, $Y_k = \sum_{\ell \in K\backslash\{k\}} y_\ell$, & $E[Y_k] = (n-1)/m$.
  Note that $Var(Y_k) = E\left[Y_k^2\right] - E[Y_k]^2$.
- $E\left[Y_k^2\right] = E[(\sum_\ell y_\ell)^2]$
  $= \sum_\ell E\left[y_\ell^2\right] + 2\sum_{\ell,j;\ell\neq j} E\left[y_\ell y_j\right]$
  $= \sum_\ell E[y_\ell] + 2\sum_{\ell,j;\ell\neq j} E[y_\ell]E\left[y_j\right]$
- $E\left[y_\ell y_j\right] = \Pr\left(h(\ell) = h(j) = h(k)\right) = 1/m^2$
  $= \Pr\left(h(\ell) = h(k)\right)\Pr\left(h(j) = h(k)\right)$,
  as $\mathcal{H}$ is a 3-independent hash function family.

# Bounding $Var(Y_k)$

- For $\ell \in K\backslash\{k\}$, the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$, $Y_k = \sum_{\ell \in K\backslash\{k\}} y_\ell$, & $E[Y_k] = (n-1)/m$. Note that $Var(Y_k) = E\left[Y_k^2\right] - E[Y_k]^2$.
- $E\left[Y_k^2\right] = E[(\sum_\ell y_\ell)^2]$
  $= \sum_\ell E\left[y_\ell^2\right] + 2\sum_{\ell,j;\ell\neq j} E\left[y_\ell y_j\right]$
  $= \sum_\ell E[y_\ell] + 2\sum_{\ell,j;\ell\neq j} E[y_\ell]E\left[y_j\right]$
- $E\left[y_\ell y_j\right] = \Pr\left(h(\ell) = h(j) = h(k)\right) = 1/m^2$
  $= \Pr\left(h(\ell) = h(k)\right)\Pr\left(h(j) = h(k)\right)$
  $= E[y_\ell]E\left[y_j\right]$

# Bounding $Var(Y_k)$

- For $\ell \in K \backslash \{k\}$, the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$, $Y_k = \sum_{\ell \in K \backslash \{k\}} y_\ell$, & $E[Y_k] = (n-1)/m$. Note that $Var(Y_k) = E\left[Y_k^2\right] - E[Y_k]^2$.
- $E\left[Y_k^2\right] = \sum_\ell E[y_\ell] + 2\sum_{\ell, j; \ell \neq j} E[y_\ell] E\left[y_j\right]$
- $E[Y_k]^2 = \left(\sum_\ell E[y_\ell]\right)^2$ (by linearity of expectation)

# Bounding $Var(Y_k)$

- For $\ell \in K\backslash\{k\}$, the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$, $Y_k = \sum_{\ell \in K\backslash\{k\}} y_\ell$, & $E[Y_k] = (n-1)/m$. Note that $Var(Y_k) = E\left[Y_k^2\right] - E[Y_k]^2$.
- $E\left[Y_k^2\right] = \sum_\ell E[y_\ell] + 2\sum_{\ell,j;\ell\neq j} E[y_\ell]E\left[y_j\right]$
- $E[Y_k]^2 = (\sum_\ell E[y_\ell])^2$ (by linearity of expectation)

  $= \sum_\ell E[y_\ell]^2 + 2\sum_{\ell,j;\ell\neq j} E[y_\ell]E\left[y_j\right]$

# Bounding $Var(Y_k)$

- For $\ell \in K\backslash\{k\}$, the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$, $Y_k = \sum_{\ell \in K\backslash\{k\}} y_\ell$, & $E[Y_k] = (n-1)/m$. Note that $Var(Y_k) = E\left[Y_k^2\right] - E[Y_k]^2$.
- $E\left[Y_k^2\right] = \sum_\ell E[y_\ell] + 2\sum_{\ell,j;\ell\neq j} E[y_\ell]E\left[y_j\right]$
- $E[Y_k]^2 = (\sum_\ell E[y_\ell])^2$ (by linearity of expectation)
  $= \sum_\ell E[y_\ell]^2 + 2\sum_{\ell,j;\ell\neq j} E[y_\ell]E\left[y_j\right]$
- $E\left[Y_k^2\right] - E[Y_k]^2 = \sum_\ell (E[y_\ell] - E[y_\ell]^2)$

# Bounding $Var(Y_k)$

- For $\ell \in K\backslash\{k\}$, the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$, $Y_k = \sum_{\ell \in K\backslash\{k\}} y_\ell$, & $E[Y_k] = (n-1)/m$. Note that $Var(Y_k) = E\left[Y_k^2\right] - E[Y_k]^2$.
- $E\left[Y_k^2\right] = \sum_\ell E[y_\ell] + 2\sum_{\ell,j;\ell\neq j} E[y_\ell]E\left[y_j\right]$
- $E[Y_k]^2 = (\sum_\ell E[y_\ell])^2$ (by linearity of expectation)
  $= \sum_\ell E[y_\ell]^2 + 2\sum_{\ell,j;\ell\neq j} E[y_\ell]E\left[y_j\right]$
- $E\left[Y_k^2\right] - E[Y_k]^2 = \sum_\ell (E[y_\ell] - E[y_\ell]^2)$
  $= (n-1)\cdot(1/m - 1/m^2)$
  $= E[Y_k]\cdot(1-1/m)$

# Bounding $Var(Y_k)$

- For $\ell \in K \backslash \{k\}$, the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$, $Y_k = \sum_{\ell \in K \backslash \{k\}} y_\ell$, & $E[Y_k] = (n-1)/m$. Note that $Var(Y_k) = E\left[Y_k^2\right] - E[Y_k]^2$.
- Hence, $Var(Y_k) = E[Y_k] \cdot (1 - 1/m)$.

# Bounding $\Pr(Y_k \geq 5)$

- For $\ell \in K\backslash\{k\}$, the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$, $Y_k = \sum_{\ell \in K\backslash\{k\}} y_\ell$, & $E[Y_k] = (n-1)/m$. Note that $Var(Y_k) = E\left[Y_k^2\right] - E[Y_k]^2$.
- Hence, $Var(Y_k) = E[Y_k] \cdot (1 - 1/m)$.
- Therefore, $\Pr(Y_k \geq 5) \leq \frac{Var(Y_k)}{16n^2 E[Y_k]^2} = \frac{m-1}{16n^2 E[Y_k] m}$

# Bounding $\Pr(Y_k \geq 5)$

- For $\ell \in K\backslash\{k\}$, the *indicator variable* $y_\ell = 1$ *iff* $h(\ell) = h(k)$, $Y_k = \sum_{\ell \in K\backslash\{k\}} y_\ell$, & $E[Y_k] = (n-1)/m$. Note that $Var(Y_k) = E\left[Y_k^2\right] - E[Y_k]^2$.
- Hence, $Var(Y_k) = E[Y_k] \cdot (1 - 1/m)$.
- Therefore, $\Pr(Y_k \geq 5) \leq \frac{Var(Y_k)}{16n^2 E[Y_k]^2} = \frac{m-1}{16n^2 E[Y_k] m}$

$$= \frac{m-1}{16n^2(n-1)} = \frac{n+1}{16n^2} \text{ (as } m = n^2\text{)}$$

$$\leq \frac{1}{10n} \text{ (assuming } n \geq 2\text{).}$$

# Pairwise independent hash functions

# Pairwise independent hash functions

- We'll now see an *explicit construction* of a $2$-independent (*pairwise independent*) hash function family. The idea behind this construction *can be generalized* to construct a $d$-independent hash function family for *any* $d \geq 2$.

# Pairwise independent hash functions

- Definition. A family $\mathcal{H}_{s,t}$ of hash functions from $\{0,1\}^s$ to $\{0,1\}^t$ is *pairwise independent* if for every *distinct* $x, x' \in \{0,1\}^s$ and for every $y, y' \in \{0,1\}^t$,

$$\Pr_{h \in \mathcal{H}_{s,t}}(h(x) = y \text{ and } h(x') = y') = 2^{-2t}.$$

# Pairwise independent hash functions

- Definition. A family $\mathcal{H}_{s,t}$ of hash functions from $\{0,1\}^s$ to $\{0,1\}^t$ is *pairwise independent* if for every *distinct* $x, x' \in \{0,1\}^s$ and for *every* $y, y' \in \{0,1\}^t$,

$$\Pr_{h \in \mathcal{H}_{s,t}}(h(x) = y \text{ and } h(x') = y') = 2^{-2t}.$$

- Obs. Let $\mathcal{H}_{s,t}$ be a pairwise independent hash function family. Then, for every $x \in \{0,1\}^s$ and $y \in \{0,1\}^t$, $\Pr_{h \in \mathcal{H}_{s,t}}(h(x) = y) = 2^{-t}$.
- That is, $\mathcal{H}_{s,t}$ is a *uniform* family.

# Pairwise independent hash functions

- Definition. A family $\mathcal{H}_{s,t}$ of hash functions from $\{0,1\}^s$ to $\{0,1\}^t$ is *pairwise independent* if for every *distinct* $x, x' \in \{0,1\}^s$ and for every $y, y' \in \{0,1\}^t$,

$$\Pr_{h \in \mathcal{H}_{s,t}}(h(x) = y \text{ and } h(x') = y') = 2^{-2t}$$

$$= \Pr_{h \in \mathcal{H}_{s,t}}(h(x) = y) \cdot \Pr_{h \in \mathcal{H}_{s,t}}(h(x') = y') \ .$$

# Pairwise independent hash functions

- Definition. A family $\mathcal{H}_{s,t}$ of hash functions from $\{0,1\}^s$ to $\{0,1\}^t$ is *pairwise independent* if for every *distinct* $x, x' \in \{0,1\}^s$ and for *every* $y, y' \in \{0,1\}^t$,

$$\Pr_{h\in\mathcal{H}_{s,t}}(h(x) = y \text{ and } h(x') = y') = 2^{-2t}$$

$$= \Pr_{h\in\mathcal{H}_{s,t}}(h(x) = y) \cdot \Pr_{h\in\mathcal{H}_{s,t}}(h(x') = y') .$$

- Example. Let $\ell > 0$ and $\mathbb{F}$ be the *finite field* of size $2^\ell$. We can identify $\mathbb{F}$ with $\{0,1\}^\ell$ as elements of $\mathbb{F}$ are $\ell$-bit strings. For $a, b \in \mathbb{F}$, define the function $h_{a,b}$ as $h_{a,b}(x) = ax + b$ for every $x \in \mathbb{F}$. Then, $\mathcal{H}_{\ell,\ell} = \{h_{a,b} : a, b \in \mathbb{F}\}$ is a *pairwise independent hash family*.

# Pairwise independent hash functions

- Example. Let $\ell > 0$ and $\mathbb{F}$ be the *finite field* of size $2^\ell$. We can identify $\mathbb{F}$ with $\{0,1\}^\ell$ as elements of $\mathbb{F}$ are $\ell$-bit strings. For $a, b \in \mathbb{F}$, define the function $h_{a,b}$ as $h_{a,b}(x) = ax + b$ for every $x \in \mathbb{F}$. Then, $\mathcal{H}_{\ell,\ell} = \{h_{a,b} : a, b \in \mathbb{F}\}$ is a *pairwise independent hash family*.
- *Proof.* Let $x, x' \in \mathbb{F}$ be distinct and $y, y' \in \mathbb{F}$. Then, $h_{a,b}(x) = y$ & $h_{a,b}(x') = y'$ *iff* $a = (y - y')/(x - x')$ and $b = (xy' - x'y)/(x - x')$.

# Pairwise independent hash functions

- Example. Let $\ell > 0$ and $\mathbb{F}$ be the *finite field* of size $2^\ell$. We can identify $\mathbb{F}$ with $\{0,1\}^\ell$ as elements of $\mathbb{F}$ are $\ell$-bit strings. For $a, b \in \mathbb{F}$, define the function $h_{a,b}$ as $h_{a,b}(x) = ax + b$ for every $x \in \mathbb{F}$. Then, $\mathcal{H}_{\ell,\ell} = \{h_{a,b} : a, b \in \mathbb{F}\}$ is a *pairwise independent hash family*.
- *Proof.* Let $x, x' \in \mathbb{F}$ be distinct and $y, y' \in \mathbb{F}$. Then, $h_{a,b}(x) = y$ & $h_{a,b}(x') = y'$ *iff* $a = (y - y')/(x - x')$ and $b = (xy' - x'y)/(x - x')$. Therefore,

$$\Pr_{a,b\in\mathbb{F}}(h_{a,b}(x) = y \;\&\; h_{a,b}(x') = y')$$

$$= \Pr_{a,b\in\mathbb{F}}(a = (y - y')/(x - x') \;\&\; b = (xy' - x'y)/(x - x'))$$

$$= 2^{-2t}$$ (as $a$ and $b$ are independently chosen).

# Pairwise independent hash functions

- Example. Let $\ell > 0$ and $\mathbb{F}$ be the *finite field* of size $2^\ell$. We can identify $\mathbb{F}$ with $\{0,1\}^\ell$ as elements of $\mathbb{F}$ are $\ell$-bit strings. For $a, b \in \mathbb{F}$, define the function $h_{a,b}$ as $h_{a,b}(x) = ax + b$ for every $x \in \mathbb{F}$. Then, $\mathcal{H}_{\ell,\ell} = \{h_{a,b} : a, b \in \mathbb{F}\}$ is a *pairwise independent hash family*.

- Obs. If $s \geq t$, then we can construct a pairwise independent $\mathcal{H}_{s,t}$ by considering $\mathcal{H}_{s,s}$ as above. *Truncate the output* of a function to the first $t$ bits.

*(Homework)*

# Pairwise independent hash functions

- Example. Let $\ell > 0$ and $\mathbb{F}$ be the *finite field* of size $2^\ell$. We can identify $\mathbb{F}$ with $\{0,1\}^\ell$ as elements of $\mathbb{F}$ are $\ell$-bit strings. For $a, b \in \mathbb{F}$, define the function $h_{a,b}$ as $h_{a,b}(x) = ax + b$ for every $x \in \mathbb{F}$. Then, $\mathcal{H}_{\ell,\ell} = \{h_{a,b} : a, b \in \mathbb{F}\}$ is a *pairwise independent hash family*.

- Obs. If $s \leq t$, then we can construct a pairwise independent $\mathcal{H}_{s,t}$ by considering $\mathcal{H}_{t,t}$ as above. Generate $t$-bit i/p for a function by *padding with* 0.

*(Homework)*

# Practice problems

# Practice Problem 1

The family $\mathcal{H}$ of hash functions is ***$\epsilon$-universal*** if for any distinct keys $k_1$ and $k_2$ in $U$, the probability that $h(k_1) = h(k_2)$ is at most $\epsilon$. Therefore, a universal family of hash functions is also $1/m$-universal.[2]

Let $U$ be the set of $d$-tuples of values drawn from $\mathbb{Z}_p$, and let $Q = \mathbb{Z}_p$, where $p$ is prime. Define the hash function $h_b : U \to Q$ for $b \in \mathbb{Z}_p$ on an input $d$-tuple $\langle a_0, a_1, \ldots, a_{d-1} \rangle$ from $U$ as

$$h_b(\langle a_0, a_1, \ldots, a_{d-1} \rangle) = \left( \sum_{j=0}^{d-1} a_j b^j \right) \bmod p \ ,$$

and let $\mathcal{H} = \{h_b : b \in \mathbb{Z}_p\}$. Argue that $\mathcal{H}$ is $\epsilon$-universal for $\epsilon = (d-1)/p$. (*Hint:*

# Practice Problem 2

Let $\mathcal{H}$ be a family of hash functions in which each hash function $h \in \mathcal{H}$ maps the universe $U$ of keys to $\{0, 1, \ldots, m-1\}$.

***a.*** Show that if the family $\mathcal{H}$ of hash functions is 2-independent, then it is universal.

***b.*** Suppose that the universe $U$ is the set of $n$-tuples of values drawn from $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$, where $p$ is prime. Consider an element $x = \langle x_0, x_1, \ldots, x_{n-1} \rangle \in U$. For any $n$-tuple $a = \langle a_0, a_1, \ldots, a_{n-1} \rangle \in U$, define the hash function $h_a$ by

$$h_a(x) = \left( \sum_{j=0}^{n-1} a_j x_j \right) \bmod p \, .$$

Let $\mathcal{H} = \{h_a : a \in U\}$. Show that $\mathcal{H}$ is universal, but not 2-independent.

# Practice Problem 2 (contd.)

Suppose that we modify $\mathcal{H}$ slightly from part (b): for any $a \in U$ and for any $b \in \mathbb{Z}_p$, define

$$h'_{ab}(x) = \left( \sum_{j=0}^{n-1} a_j x_j + b \right) \bmod p$$

and $\mathcal{H}' = \{h'_{ab} : a \in U \text{ and } b \in \mathbb{Z}_p\}$. Argue that $\mathcal{H}'$ is 2-independent. (*Hint:*

# Practice Problem 3

Alice and Bob secretly agree on a hash function $h$ from a 2-independent family $\mathcal{H}$ of hash functions. Each $h \in \mathcal{H}$ maps from a universe of keys $U$ to $\mathbb{Z}_p$, where $p$ is prime. Later, Alice sends a message $m$ to Bob over the internet, where $m \in U$. She authenticates this message to Bob by also sending an authentication tag $t = h(m)$, and Bob checks that the pair $(m, t)$ he receives indeed satisfies $t = h(m)$. Suppose that an adversary intercepts $(m, t)$ en route and tries to fool Bob by replacing the pair $(m, t)$ with a different pair $(m', t')$. Argue that the probability that the adversary succeeds in fooling Bob into accepting $(m', t')$ is at most $1/p$, no matter how much computing power the adversary has, even if the adversary knows the family $\mathcal{H}$ of hash functions used.

Assume that $h$ is randomly chosen from $\mathcal{H}$; the probability is over the random choice of $h$. Also, assume that $m \neq m'$.

# Practice Problem 4

~~(10 marks)~~ Let $k \leq n$. Prove that the following family $\mathcal{H}_{n,k}$ is a collection of pairwise independent hash functions from $\{0,1\}^n$ to $\{0,1\}^k$. Identify $\{0,1\}$ with the field $\mathbb{F}_2$. For every $k \times n$ matrix $A$ with entries in $\mathbb{F}_2$, and $\mathbf{b} \in \mathbb{F}_2^k$, $\mathcal{H}_{n,k}$ contains the function $h_{A,\mathbf{b}} : \mathbb{F}_2^n \to \mathbb{F}_2^k$ defined as $h_{A,\mathbf{b}}(\mathbf{x}) = A\mathbf{x} + \mathbf{b}$ for $\mathbf{x} \in \mathbb{F}_2^n$.