Lecture 2
Hash tables

Markus Jalsenius
Dictionaries

In a **dictionary** data structure we store \((key, value)\)-pairs such that for any \(key\) there is at most one pair \((key, value)\) in the dictionary.

Often we want to perform the following three operations:

- **add**\((x, v)\) Add the the pair \((x, v)\).
- **lookup**\((x)\) Return \(v\) if \((x, v)\) is in dictionary, or **NULL** otherwise.
- **delete**\((x)\) Remove pair \((x, v)\) (assuming \((x, v)\) is in dictionary).
Dictionaries

In a dictionary data structure we store (key, value)-pairs such that for any key there is at most one pair (key, value) in the dictionary.

Often we want to perform the following three operations:

- add\((x, v)\) Add the the pair \((x, v)\).
- lookup\((x)\) Return \(v\) if \((x, v)\) is in dictionary, or NULL otherwise.
- delete\((x)\) Remove pair \((x, v)\) (assuming \((x, v)\) is in dictionary).

There are many data structures that will do this job, e.g.:

- Linked lists
- Binary search trees
- B-trees
- Red-black trees
- Skip lists
- More elaborate tree structures, e.g. van Emde Boas trees (which we will cover in this course) that also allow other operations than the three listed here.
Hash tables

- We want to store $n$ elements from the universe in the dictionary.
- Typically $u$ is much, much larger than $n$.

Universe $U$ containing $u$ keys.
Hash tables

- We want to store \( n \) elements from the universe in the dictionary.
- Typically \( u \) is much, much larger than \( n \).

Universe \( U \) containing \( u \) keys. Array \( T \) of size \( m \).

- A hash function \( h : U \rightarrow [m] \) maps a key to a position in \( T \).
  
  **Observe** We write \([m]\) to denote the set \( \{0, \ldots, m - 1\} \).

- \( T \) is referred to as a hash table.
Hash tables

- We want to store $n$ elements from the universe in the dictionary.
- Typically $u$ is much, much larger than $n$.

Universe $U$ containing $u$ keys. Array $T$ of size $m$.

- A hash function $h : U \rightarrow [m]$ maps a key to a position in $T$.
  
  **Observe** We write $[m]$ to denote the set $\{0, \ldots, m - 1\}$.

- $T$ is referred to as a hash table.
Hash tables

- We want to store \( n \) elements from the universe in the dictionary.
- Typically \( u \) is much, much larger than \( n \).

Universe \( U \) containing \( u \) keys. Array \( T \) of size \( m \).

A hash function \( h : U \rightarrow [m] \) maps a key to a position in \( T \).

**Observe** We write \([m]\) to denote the set \( \{0, \ldots, m - 1\} \).

\( T \) is referred to as a hash table.

We want to avoid collisions: \( h(x) = h(y) \) for \( x \neq y \).
Hash tables

- We want to store \( n \) elements from the universe in the dictionary.
- Typically \( u \) is much, much larger than \( n \).

Universe \( U \) containing \( u \) keys. Array \( T \) of size \( m \).

- Collisions can be resolved with **chaining**, i.e. linked list.

- A hash function \( h : U \to [m] \) maps a key to a position in \( T \).

  **Observe** We write \([m]\) to denote the set \( \{0, \ldots, m - 1\} \).

- \( T \) is referred to as a **hash table**.

- We want to avoid **collisions**: \( h(x) = h(y) \) for \( x \neq y \).
Hash tables

- We want to store $n$ elements from the universe in the dictionary.
- Typically $u$ is much, much larger than $n$.

Universe $U$ containing $u$ keys. Array $T$ of size $m$.

A hash function $h : U \rightarrow [m]$ maps a key to a position in $T$.

- Collisions can be resolved with chaining, i.e. linked list.

- $T$ is referred to as a hash table.
- We want to avoid collisions: $h(x) = h(y)$ for $x \neq y$.

Observe: We write $[m]$ to denote the set $\{0, \ldots, m - 1\}$.

Observe: Do not confuse hash functions here with cryptographic hash functions.
Time complexity

- We cannot avoid collisions entirely since \( u > m \); some keys are bound to be mapped to the same position.
- Using chaining, we have the following time complexities:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Worst case time</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>add((x, v))</td>
<td>(O(1))</td>
<td>Simply add item to the list link if necessary.</td>
</tr>
<tr>
<td>lookup((x))</td>
<td>(O(\text{length of linked list}))</td>
<td>We might have to search through the whole list.</td>
</tr>
<tr>
<td>delete((x))</td>
<td>(O(1))</td>
<td>Constant time if we have already located the element in the linked list.</td>
</tr>
</tbody>
</table>
**Theorem**

Consider any $n$ fixed inputs to the hash table, i.e. sequence of add/lookup/delete operations. Pick $h$ uniformly at random from the set of all functions $U \rightarrow [m]$. The expected run-time per operation is $O(1 + \frac{n}{m})$, or simply $O(1)$ if $n = m$. 

**True randomness**
**Theorem**
Consider any $n$ fixed inputs to the hash table, i.e. sequence of add/lookup/delete operations. Pick $h$ uniformly at random from the set of all functions $U \rightarrow [m]$. The expected run-time per operation is $O(1 + \frac{n}{m})$, or simply $O(1)$ if $n = m$.

**Proof**
- Let $x, y$ be two distinct keys from $U$.
- Let indicator r.v. $I_{x,y}$ be 1 iff $h(x) = h(y)$.
- $\Pr(h(x) = h(y)) = \frac{1}{m}$ since $h(x)$ and $h(y)$ are chosen uniformly and independently from $[m]$.
- Thus, $\mathbb{E}(I_{x,y}) = \frac{1}{m}$.
- Let $N_x$ be the number of keys stored in $T$ that are hashed to $h(x)$. Thus, in worst case it takes $N_x$ time to look up $x$ in $T$.
- $N_x = \sum_{y \in T} I_{x,y}$.
- $\mathbb{E}(N_x) = \sum_{y \in T} \mathbb{E}(I_{x,y}) = n \cdot \frac{1}{m} = \frac{n}{m}$ (linearity of expectation).
Specifying the hash function

Problem: how do we specify an *arbitrary* hash function?
Specifying the hash function

**Problem**: how do we specify an *arbitrary* hash function?

For each key in $U$ we need to specify an arbitrary position in $T$, which is a number in $[m]$, hence requires $\log_2 m$ bits. Thus, in total we need $u \log_2 m$ bits, which is a huge amount of space! ($u = |U|$.)
Specifying the hash function

Problem: how do we specify an arbitrary hash function?

- For each key in $U$ we need to specify an arbitrary position in $T$, which is a number in $[m]$, hence requires $\log_2 m$ bits.
  Thus, in total we need $u \log_2 m$ bits, which is a huge amount of space! ($u = |U|$.)

Fixed hashing is vulnerable to bad worst-case behaviour:
Given a fixed hash function $h$, an adversary could pick $n$ keys such that they all map to the same position in $T$. 
Specifying the hash function

- **Problem**: how do we specify an *arbitrary* hash function?
  - For each key in $U$ we need to specify an arbitrary position in $T$, which is a number in $[m]$, hence requires $\log_2 m$ bits.
  - Thus, in total we need $u \log_2 m$ bits, which is a huge amount of space! ($u = |U|$.)

- Fixed hashing is vulnerable to bad worst-case behaviour:
  - Given a fixed hash function $h$, an adversary could pick $n$ keys such that they all map to the same position in $T$.

- Instead of using some specific hash function, we define a whole set, or family, of hash functions: $H = \{h_1, h_2, \ldots\}$.
  - As part of initialising the hash table, we chose the hash function $h$ from $H$ randomly.
Specifying the hash function

**Problem**: how do we specify an *arbitrary* hash function?

- For each key in $U$ we need to specify an arbitrary position in $T$, which is a number in $[m]$, hence requires $\log_2 m$ bits. Thus, in total we need $u \log_2 m$ bits, which is a huge amount of space! ($u = |U|$.)

- Fixed hashing is vulnerable to bad worst-case behaviour: Given a fixed hash function $h$, an adversary could pick $n$ keys such that they all map to the same position in $T$.

- Instead of using some specific hash function, we define a whole set, or family, of hash functions: $H = \{h_1, h_2, \ldots \}$. As part of initialising the hash table, we chose the hash function $h$ from $H$ randomly.

- Again, how do we specify the hash functions in $H$ and how do we pick one at random?
Weakly universal hashing

A set $H$ of hash functions is **weakly universal** if for any two keys $x, y \in U$ (such that $x \neq y$),

$$\Pr (h(x) = h(y)) \leq \frac{1}{m}$$

where $h$ is chosen uniformly at random from $H$. 
Weakly universal hashing

A set $H$ of hash functions is **weakly universal** if for any two keys $x, y \in U$ (such that $x \neq y$),

$$\Pr (h(x) = h(y)) \leq \frac{1}{m}$$

where $h$ is chosen uniformly at random from $H$.

**Observe**
The randomness here comes from the fact that $h$ is picked randomly.
Weakly universal hashing

A set $H$ of hash functions is weakly universal if for any two keys $x, y \in U$ (such that $x \neq y$),

$$
\Pr (h(x) = h(y)) \leq \frac{1}{m}
$$

where $h$ is chosen uniformly at random from $H$.

**Theorem**

Consider any $n$ fixed inputs to the hash table, i.e. sequence of add/lookup/delete operations. Pick $h$ uniformly at random from a weakly universal set $H$ of hash functions. The expected run-time per operation is $O(1)$ if $m \geq n$.

**Observe**

The randomness here comes from the fact that $h$ is picked randomly.
Weakly universal hashing

A set $H$ of hash functions is **weakly universal** if for any two keys $x, y \in U$ (such that $x \neq y$),

$$\Pr(h(x) = h(y)) \leq \frac{1}{m}$$

where $h$ is chosen uniformly at random from $H$.

**Theorem**

Consider any $n$ fixed inputs to the hash table, i.e. sequence of add/lookup/delete operations. Pick $h$ uniformly at random from a weakly universal set $H$ of hash functions. The expected run-time per operation is $O(1)$ if $m \geq n$.

**Proof**

Analogous to the previous proof. Go through it and verify.

**Observe**

The randomness here comes from the fact that $h$ is picked randomly.
Constructing a weakly universal family of hash functions

- Suppose $U = [u]$, i.e. the keys in the universe are integers 0 to $u - 1$.
- Let $p > u$ be any prime.
- For $a, b \in [p]$, let

$$h_{a,b}(x) = (ax + b \mod p) \mod m,$$

$$H_{p,m} = \{h_{a,b} \mid a \in \{1, \ldots, p - 1\}, b \in \{0, \ldots, p - 1\}\}.$$
Constructing a weakly universal family of hash functions

- Suppose \( U = [u] \), i.e. the keys in the universe are integers 0 to \( u - 1 \).
- Let \( p > u \) be any prime.
- For \( a, b \in [p] \), let
  
  \[ h_{a,b}(x) = (ax + b \mod p) \mod m, \]

  \[ H_{p,m} = \{h_{a,b} \mid a \in \{1, \ldots, p - 1\}, b \in \{0, \ldots, p - 1\}\}. \]

**Theorem**

\( H_{p,m} \) is a weakly universal set of hash functions.

**Proof**

See CLRS, Theorem 11.5, page 267.
Constructing a weakly universal family of hash functions

► Suppose $U = [u]$, i.e. the keys in the universe are integers 0 to $u - 1$.
► Let $p > u$ be any prime.
► For $a, b \in [p]$, let

$$h_{a,b}(x) = (ax + b \mod p) \mod m,$$

$$H_{p,m} = \{h_{a,b} \mid a \in \{1, \ldots, p - 1\}, b \in \{0, \ldots, p - 1\}\}.$$ 

**Theorem**

$H_{p,m}$ is a weakly universal set of hash functions.

**Proof**

See CLRS, Theorem 11.5, page 267.

**Observe**

► $ax + b$ is a linear transformation “spreading the keys” over $p$ values when taken modulo $p$. This does not cause any collisions.
► Only when taken modulo $m$ we get collisions.
True randomness vs. weakly universal hashing

For both

- **true randomness** ($h$ is picked uniformly from the set of all possible hash functions) and
- **weakly universal hashing** ($h$ is picked uniformly from a weakly universal set of hash functions),

we have seen on previous slides that when $m = n$ then the expected lookup time in the hash table is $O(1)$. 
True randomness vs. weakly universal hashing

- For both
  - **true randomness** ($h$ is picked uniformly from the set of all possible hash functions) and
  - **weakly universal hashing** ($h$ is picked uniformly from a weakly universal set of hash functions),

we have seen on previous slides that when $m = n$ then the expected lookup time in the hash table is $O(1)$.

- Since constructing a weakly universal set of hash functions seems easier than obtaining true randomness, this is all good news!
  Or…?
True randomness vs. weakly universal hashing

For both

- **true randomness** ($h$ is picked uniformly from the set of all possible hash functions) and
- **weakly universal hashing** ($h$ is picked uniformly from a weakly universal set of hash functions),

we have seen on previous slides that when $m = n$ then the expected lookup time in the hash table is $O(1)$.

Since constructing a weakly universal set of hash functions seems easier than obtaining true randomness, this is all good news!

Or…?

What about the longest chain?
If it is very long, then every now and then a lookup could take very long time.
If $h$ is selected uniformly at random from all functions $U \rightarrow [m]$ then, over $m$ fixed inputs,

$$\Pr \left( \text{any chain has length} \geq 3 \log m \right) \leq \frac{1}{m}.$$
**LEMA**

If $h$ is selected uniformly at random from all functions $U \rightarrow [m]$ then, over $m$ fixed inputs,

$$\Pr \left( \text{any chain has length } \geq 3 \log m \right) \leq \frac{1}{m}.$$

**OBSERVE**

In this lemma we insert $m$ keys, i.e. $n = m$. 
**Lemma**

If \( h \) is selected uniformly at random from all functions \( U \rightarrow [m] \) then, over \( m \) fixed inputs,

\[
\Pr \left( \text{any chain has length} \geq 3 \log m \right) \leq \frac{1}{m}.
\]

**Observe**

In this lemma we insert \( m \) keys, i.e. \( n = m \).

**Proof**

The problem is equivalent to showing that if we randomly throw \( m \) balls into \( m \) bins, the probability of having a bin with at least \( 3 \log m \) balls is at most \( \frac{1}{m} \).
Longest chain – true randomness

**Proof**

continued…

- Let $X_1$ be the number of balls in the first bin.
- On the event $X_1 \geq k$, one can find a subset of size $k$ of the balls such that all these balls are in the first bin. We will choose $k$ shortly.
Let \( X_1 \) be the number of balls in the first bin.

On the event \( X_1 \geq k \), one can find a subset of size \( k \) of the balls such that all these balls are in the first bin. We will choose \( k \) shortly.

For \( k \) given balls, they go into the first bin with probability \( \frac{1}{m^k} \).

So, the union bound gives

\[
\Pr(X_1 \geq k) \leq \binom{m}{k} \cdot \frac{1}{m^k} \leq \frac{1}{k!}.
\]
Longest chain – true randomness

**Proof continued...**

Let $X_1$ be the number of balls in the first bin.

On the event $X_1 \geq k$, one can find a subset of size $k$ of the balls such that all these balls are in the first bin. We will choose $k$ shortly.

For $k$ given balls, they go into the first bin with probability $\frac{1}{m^k}$.

So, the union bound gives

$$Pr(X_1 \geq k) \leq \binom{m}{k} \cdot \frac{1}{m^k} \leq \frac{1}{k!}.$$ 

Using the union bound again, we have

$$Pr(\text{at least one bin receives at least } k \text{ balls}) \leq m \cdot Pr(X_1 \geq k) \leq \frac{m}{k!}.$$
Let $X_1$ be the number of balls in the first bin.

On the event $X_1 \geq k$, one can find a subset of size $k$ of the balls such that all these balls are in the first bin. We will choose $k$ shortly.

For $k$ given balls, they go into the first bin with probability $\frac{1}{m^k}$.

So, the union bound gives

$$\Pr(X_1 \geq k) \leq \binom{m}{k} \cdot \frac{1}{m^k} \leq \frac{1}{k!}.$$  

Using the union bound again, we have

$$\Pr(\text{at least one bin receives at least } k \text{ balls}) \leq m \cdot \Pr(X_1 \geq k) \leq \frac{m}{k!}.$$  

Now we set $k = 3 \log m$ and observe that $\frac{m}{k!} \leq \frac{1}{m}$ for $m \geq 2$, and we are done.
Longest chain – true randomness

**Proof**

continued…

- Let $X_1$ be the number of balls in the first bin.
- On the event $X_1 \geq k$, one can find a subset of size $k$ of the balls such that all these balls are in the first bin. We will choose $k$ shortly.
- For $k$ given balls, they go into the first bin with probability $\frac{1}{m^k}$.
- So, the union bound gives
  \[
  \Pr(X_1 \geq k) \leq \binom{m}{k} \cdot \frac{1}{m^k} \leq \frac{1}{k!}.
  \]
- Using the union bound again, we have
  \[
  \Pr(\text{at least one bin receives at least } k \text{ balls}) \leq m \cdot \Pr(X_1 \geq k) \leq \frac{m}{k!}.
  \]
- Now we set $k = 3 \log m$ and observe that
  \[
  \frac{m}{k!} \leq \frac{1}{m} \text{ for } m \geq 2,
  \]
  and we are done.

As an exercise, prove . Hint: $k! \geq 2^{k-1}$. We have assumed log is in base 2.
The conclusion from previous slides is that with true randomness, the longest chain is very short (at most $3 \log m$) with high probability.
The conclusion from previous slides is that with true randomness, the longest chain is very short (at most $3 \log m$) with high probability.

**Lemma**

If $h$ is picked uniformly at random from a weakly universal set of all functions $U \rightarrow [m]$ then, over $m$ fixed inputs,

$$\Pr \left( \text{any chain has length } \geq 1 + \sqrt{2m} \right) \leq \frac{1}{2}.$$
The conclusion from previous slides is that with true randomness, the longest chain is very short (at most $3 \log m$) with high probability.

**Lemma**

If $h$ is picked uniformly at random from a weakly universal set of all functions $U \rightarrow [m]$ then, over $m$ fixed inputs,

$$\Pr \left( \text{any chain has length} \geq 1 + \sqrt{2m} \right) \leq \frac{1}{2}.$$  

**Observe**

This rubbish upper bound of $\frac{1}{2}$ does not necessarily rule out the possibility that the *tightest* upper bound is indeed very small. However, the upper bound of $\frac{1}{2}$ is in fact tight!
Longest chain – weakly universal hashing

**Proof**

- For any two keys $x, y$, let indicator r.v. $I_{x,y}$ be 1 iff $h(x) = h(y)$. 
Longest chain – weakly universal hashing

**Proof**

- For any two keys $x, y$, let indicator r.v. $I_{x,y}$ be 1 iff $h(x) = h(y)$.
- Let r.v. $C$ be the total number of collisions: $C = \sum_{x,y \in T, x < y} I_{x,y}$. 

Proof

- For any two keys $x, y$, let indicator r.v. $I_{x,y}$ be 1 iff $h(x) = h(y)$.
- Let r.v. $C$ be the total number of collisions: $C = \sum_{x, y \in T, x < y} I_{x,y}$.
- Using linearity of expectation and $\mathbb{E}(I_{x,y}) = \frac{1}{m}$ (weakly universal),

$$
\mathbb{E}(C) = \mathbb{E}\left( \sum_{x, y \in T, x < y} I_{x,y} \right) = \sum_{x, y \in T, x < y} \mathbb{E}(I_{x,y}) = \binom{m}{2} \cdot \frac{1}{m} \leq \frac{m}{2}.
$$
Longest chain – weakly universal hashing

**Proof**

► For any two keys \( x, y \), let indicator r.v. \( I_{x,y} \) be 1 iff \( h(x) = h(y) \).

► Let r.v. \( C \) be the total number of collisions: \( C = \sum_{x,y \in T, x < y} I_{x,y} \).

► Using linearity of expectation and \( \mathbb{E}(I_{x,y}) = \frac{1}{m} \) (weakly universal),

\[
\mathbb{E}(C) = \mathbb{E}\left( \sum_{x,y \in T, x < y} I_{x,y} \right) = \sum_{x,y \in T, x < y} \mathbb{E}(I_{x,y}) = \binom{m}{2} \cdot \frac{1}{m} \leq \frac{m}{2}.
\]

► Markov’s inequality: \( \Pr(C \geq m) \leq \frac{\mathbb{E}(C)}{m} \leq \frac{1}{2} \).
For any two keys $x, y$, let indicator r.v. $I_{x,y}$ be 1 iff $h(x) = h(y)$.

Let r.v. $C$ be the total number of collisions: $C = \sum_{x, y \in T, x < y} I_{x,y}$.

Using linearity of expectation and $\mathbb{E}(I_{x,y}) = \frac{1}{m}$ (weakly universal),

$$\mathbb{E}(C) = \mathbb{E}\left( \sum_{x, y \in T, x < y} I_{x,y} \right) = \sum_{x, y \in T, x < y} \mathbb{E}(I_{x,y}) = \binom{m}{2} \cdot \frac{1}{m} \leq \frac{m}{2}.$$ 

Markov’s inequality: $\Pr(C \geq m) \leq \frac{\mathbb{E}(C)}{m} \leq \frac{1}{2}$.

Let r.v. $L$ be the length of the longest chain. Then $\left( \frac{L}{2} \right) \leq C$. Why?
Longest chain – weakly universal hashing

**Proof**

- For any two keys $x, y$, let indicator r.v. $I_{x,y}$ be 1 iff $h(x) = h(y)$.
- Let r.v. $C$ be the total number of collisions: $C = \sum_{x, y \in T, x < y} I_{x,y}$.
- Using linearity of expectation and $\mathbb{E}(I_{x,y}) = \frac{1}{m}$ (weakly universal),

$$
\mathbb{E}(C) = \mathbb{E}\left(\sum_{x, y \in T, x < y} I_{x,y}\right) = \sum_{x, y \in T, x < y} \mathbb{E}(I_{x,y}) = \left(\begin{array}{c} m \\ 2 \end{array}\right) \cdot \frac{1}{m} \leq \frac{m}{2}.
$$

- Markov’s inequality: $\Pr(C \geq m) \leq \frac{\mathbb{E}(C)}{m} \leq \frac{1}{2}$.

- Let r.v. $L$ be the length of the longest chain. Then $\left(\frac{L}{2}\right) \leq C$. Why?

- Now, $\Pr\left(\frac{(L-1)^2}{2} \geq m\right) \leq \Pr\left(\left(\frac{L}{2}\right) \geq m\right) \leq \Pr\left(C \geq m\right) \leq \frac{1}{2}$. Why?
Longest chain – weakly universal hashing

**Proof**

- For any two keys $x, y$, let indicator r.v. $I_{x,y}$ be 1 iff $h(x) = h(y)$.
- Let r.v. $C$ be the total number of collisions: $C = \sum_{x,y \in T, x < y} I_{x,y}$.
- Using linearity of expectation and $\mathbb{E}(I_{x,y}) = \frac{1}{m}$ (weakly universal),

$$
\mathbb{E}(C) = \mathbb{E}\left( \sum_{x,y \in T, x < y} I_{x,y} \right) = \sum_{x,y \in T, x < y} \mathbb{E}(I_{x,y}) = \binom{m}{2} \cdot \frac{1}{m} \leq \frac{m}{2}.
$$

- Markov’s inequality: $\Pr(C \geq m) \leq \frac{\mathbb{E}(C)}{m} \leq \frac{1}{2}$.
- Let r.v. $L$ be the length of the longest chain. Then $\left( \frac{L}{2} \right) \leq C$. Why?

- Now, $\Pr\left( \frac{(L-1)^2}{2} \geq m \right) \leq \Pr\left( \left( \frac{L}{2} \right) \geq m \right) \leq \Pr(C \geq m) \leq \frac{1}{2}$.

- This implies that $\Pr(L \geq 1 + \sqrt{2m}) \leq \frac{1}{2}$, and we are done.