Resolving Collisions

The methods for dealing with collisions can be classified into the two main types

- **open addressing**, aka. closed hashing
- **hashing with chaining**, aka. closed addressing, open hashing.

There are applications e.g. computer chess where you do not resolve collisions at all.

Resolving Collisions

The methods for dealing with collisions can be classified into the two main types

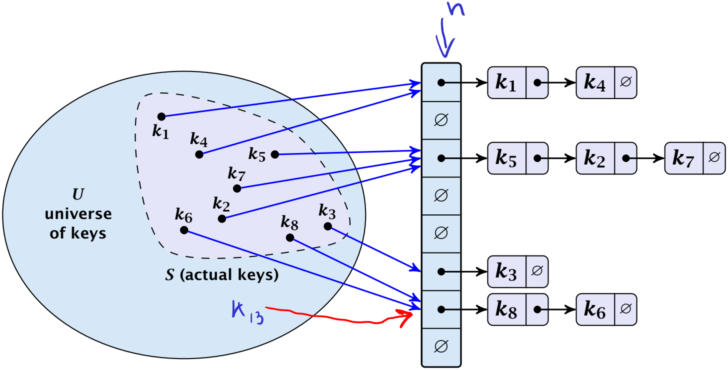
- **open addressing**, aka. closed hashing
- **hashing with chaining**, aka. closed addressing, open hashing.

There are applications e.g. computer chess where you do not resolve collisions at all.

Hashing with Chaining

Arrange elements that map to the same position in a linear list.

- Access: compute $h(x)$ and search list for $\mathrm{key}[x]$.
- Insert: insert at the front of the list.



Hashing with Chaining

Let A denote a strategy for resolving collisions. We use the following notation:

- A^+ denotes the average time for a **successful** search when using A;
- A^- denotes the average time for an **unsuccessful** search when using A;
- We parameterize the complexity results in terms of $\alpha := \frac{m}{n}$, the so-called fill factor of the hash-table.

We assume uniform hashing for the following analysis.

Hashing with Chaining

Let A denote a strategy for resolving collisions. We use the following notation:

- A^+ denotes the average time for a **successful** search when using A;
- A^- denotes the average time for an **unsuccessful** search when using A;
- We parameterize the complexity results in terms of $\alpha := \frac{m}{n}$, the so-called fill factor of the hash-table.

We assume uniform hashing for the following analysis.

Hashing with Chaining

Let A denote a strategy for resolving collisions. We use the following notation:

- A^+ denotes the average time for a **successful** search when using A;
- A^- denotes the average time for an **unsuccessful** search when using A;
- We parameterize the complexity results in terms of $\alpha := \frac{m}{n}$, the so-called fill factor of the hash-table.

We assume uniform hashing for the following analysis.

Hashing with Chaining

Let A denote a strategy for resolving collisions. We use the following notation:

- A^+ denotes the average time for a **successful** search when using A;
- A^- denotes the average time for an **unsuccessful** search when using A;
- We parameterize the complexity results in terms of $\alpha := \frac{m}{n}$, the so-called fill factor of the hash-table.

We assume uniform hashing for the following analysis.

Hashing with Chaining

Let A denote a strategy for resolving collisions. We use the following notation:

- A^+ denotes the average time for a **successful** search when using A;
- A^- denotes the average time for an **unsuccessful** search when using A;
- We parameterize the complexity results in terms of $\alpha := \frac{m}{n}$, the so-called fill factor of the hash-table.

We assume uniform hashing for the following analysis.

Hashing with Chaining

The time required for an unsuccessful search is 1 plus the length of the list that is examined.

Hashing with Chaining

The time required for an unsuccessful search is 1 plus the length of the list that is examined. The average length of a list is $\alpha = \frac{m}{n}$.

Hashing with Chaining

The time required for an unsuccessful search is 1 plus the length of the list that is examined. The average length of a list is $\alpha = \frac{m}{n}$. Hence, if A is the collision resolving strategy “Hashing with Chaining” we have

$$A^- = 1 + \alpha \ .$$

Hashing with Chaining

For a successful search observe that we do **not** choose a list at random, but we consider a random key k in the hash-table and ask for the search-time for k.

This is 1 plus the number of elements that lie before k in k's list.

Let k_ℓ denote the ℓ-th key inserted into the table.

Let for two keys k_i and k_j, X_{ij} denote the indicator variable for the event that k_i and k_j hash to the same position. Clearly, $\Pr[X_{ij} = 1] = 1/n$ for uniform hashing.

The expected successful search cost is

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

Hashing with Chaining

For a successful search observe that we do **not** choose a list at random, but we consider a random key k in the hash-table and ask for the search-time for k.

This is 1 plus the number of elements that lie before k in k's list.

Let k_ℓ denote the ℓ-th key inserted into the table.

Let for two keys k_i and k_j, X_{ij} denote the indicator variable for the event that k_i and k_j hash to the same position. Clearly, $\Pr[X_{ij} = 1] = 1/n$ for uniform hashing.

The expected successful search cost is

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

Hashing with Chaining

For a successful search observe that we do **not** choose a list at random, but we consider a random key k in the hash-table and ask for the search-time for k.

This is 1 plus the number of elements that lie before k in k's list.

Let k_ℓ denote the ℓ-th key inserted into the table.

Let for two keys k_i and k_j, X_{ij} denote the indicator variable for the event that k_i and k_j hash to the same position. Clearly, $\Pr[X_{ij} = 1] = 1/n$ for uniform hashing.

The expected successful search cost is

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

Hashing with Chaining

For a successful search observe that we do **not** choose a list at random, but we consider a random key k in the hash-table and ask for the search-time for k.

This is 1 plus the number of elements that lie before k in k's list.

Let k_ℓ denote the ℓ-th key inserted into the table.

Let for two keys k_i and k_j, X_{ij} denote the indicator variable for the event that k_i and k_j hash to the same position. Clearly, $\Pr[X_{ij} = 1] = 1/n$ for uniform hashing.

The expected successful search cost is

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

Hashing with Chaining

For a successful search observe that we do **not** choose a list at random, but we consider a random key k in the hash-table and ask for the search-time for k.

This is 1 plus the number of elements that lie before k in k's list.

Let k_ℓ denote the ℓ-th key inserted into the table.

Let for two keys k_i and k_j, X_{ij} denote the indicator variable for the event that k_i and k_j hash to the same position. Clearly, $\Pr[X_{ij} = 1] = 1/n$ for uniform hashing.

The expected successful search cost is

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

Hashing with Chaining

For a successful search observe that we do **not** choose a list at random, but we consider a random key k in the hash-table and ask for the search-time for k.

This is 1 plus the number of elements that lie before k in k's list.

Let k_ℓ denote the ℓ-th key inserted into the table.

Let for two keys k_i and k_j, X_{ij} denote the indicator variable for the event that k_i and k_j hash to the same position. Clearly, $\Pr[X_{ij} = 1] = 1/n$ for uniform hashing.

The expected successful search cost is

$$\mathrm{E}\left[\frac{1}{m}\sum_{i=1}^{m}\Big(1 + \overbrace{\sum_{j=i+1}^{m} X_{ij}}^{\text{keys before } k_i}\Big)\right]$$

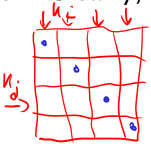
Hashing with Chaining

For a successful search observe that we do **not** choose a list at random, but we consider a random key k in the hash-table and ask for the search-time for k.

This is 1 plus the number of elements that lie before k in k's list.

Let k_ℓ denote the ℓ-th key inserted into the table.

Let for two keys k_i and k_j, X_{ij} denote the indicator variable for the event that k_i and k_j hash to the same position. Clearly, $\Pr[X_{ij} = 1] = 1/n$ for uniform hashing.



The expected successful search cost is

$$\mathrm{E}\left[\frac{1}{m}\sum_{i=1}^{m}\underbrace{\left(1+\sum_{j=i+1}^{m}X_{ij}\right)}_{\text{cost for key } k_i}\right]$$

Hashing with Chaining

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

Hashing with Chaining

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

Hashing with Chaining

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

$$=\frac{1}{m}\sum_{i=1}^{m}\left(1+\sum_{j=i+1}^{m}\frac{1}{n}\right)$$

Hashing with Chaining

$$
\begin{aligned}
\mathrm{E}\left[\frac{1}{m}\sum_{i=1}^{m}\left(1+\sum_{j=i+1}^{m}X_{ij}\right)\right] &= \frac{1}{m}\sum_{i=1}^{m}\left(1+\sum_{j=i+1}^{m}\mathrm{E}\left[X_{ij}\right]\right)\\
&= \frac{1}{m}\sum_{i=1}^{m}\left(1+\sum_{j=i+1}^{m}\frac{1}{n}\right)\\
&= 1+\frac{1}{mn}\sum_{i=1}^{m}(m-i)
\end{aligned}
$$

Hashing with Chaining

$$
\begin{aligned}
\mathrm{E}\left[\frac{1}{m}\sum_{i=1}^{m}\left(1+\sum_{j=i+1}^{m}X_{ij}\right)\right] &= \frac{1}{m}\sum_{i=1}^{m}\left(1+\sum_{j=i+1}^{m}\mathrm{E}\left[X_{ij}\right]\right)\\
&= \frac{1}{m}\sum_{i=1}^{m}\left(1+\sum_{j=i+1}^{m}\frac{1}{n}\right)\\
&= 1+\frac{1}{mn}\sum_{i=1}^{m}(m-i)\\
&= 1+\frac{1}{mn}\left(m^2-\frac{m(m+1)}{2}\right)
\end{aligned}
$$

Hashing with Chaining

$$
\begin{aligned}
\mathrm{E}\left[\frac{1}{m}\sum_{i=1}^{m}\left(1+\sum_{j=i+1}^{m}X_{ij}\right)\right] &= \frac{1}{m}\sum_{i=1}^{m}\left(1+\sum_{j=i+1}^{m}\mathrm{E}\left[X_{ij}\right]\right)\\
&= \frac{1}{m}\sum_{i=1}^{m}\left(1+\sum_{j=i+1}^{m}\frac{1}{n}\right)\\
&= 1+\frac{1}{mn}\sum_{i=1}^{m}(m-i)\\
&= 1+\frac{1}{mn}\underbrace{\left(m^2-\frac{m(m+1)}{2}\right)}_{\frac{1}{mn}\left(\frac{m^2}{2}\left(-\frac{m}{2}\right)\right)}\\
&= 1+\frac{m-1}{2n}
\end{aligned}
$$

$-\frac{1}{2n}$

Hashing with Chaining

$$
\begin{aligned}
\mathrm{E}\Big[\frac{1}{m}\sum_{i=1}^{m}\Big(1+\sum_{j=i+1}^{m}X_{ij}\Big)\Big] &= \frac{1}{m}\sum_{i=1}^{m}\Big(1+\sum_{j=i+1}^{m}\mathrm{E}\big[X_{ij}\big]\Big)\\
&= \frac{1}{m}\sum_{i=1}^{m}\Big(1+\sum_{j=i+1}^{m}\frac{1}{n}\Big)\\
&= 1+\frac{1}{mn}\sum_{i=1}^{m}(m-i)\\
&= 1+\frac{1}{mn}\Big(m^2-\frac{m(m+1)}{2}\Big)\\
&= 1+\frac{m-1}{2n} = 1+\frac{\alpha}{2}-\frac{\alpha}{2m}\ .
\end{aligned}
$$

Hashing with Chaining

$$
\begin{aligned}
\mathrm{E}\left[\frac{1}{m}\sum_{i=1}^{m}\left(1+\sum_{j=i+1}^{m}X_{ij}\right)\right] &= \frac{1}{m}\sum_{i=1}^{m}\left(1+\sum_{j=i+1}^{m}\mathrm{E}\left[X_{ij}\right]\right)\\
&= \frac{1}{m}\sum_{i=1}^{m}\left(1+\sum_{j=i+1}^{m}\frac{1}{n}\right)\\
&= 1+\frac{1}{mn}\sum_{i=1}^{m}(m-i)\\
&= 1+\frac{1}{mn}\left(m^2-\frac{m(m+1)}{2}\right)\\
&= 1+\frac{m-1}{2n} = 1+\frac{\alpha}{2}-\frac{\alpha}{2m}\ .
\end{aligned}
$$

Hence, the expected cost for a successful search is $A^{+} \leq 1+\frac{\alpha}{2}$.

Hashing with Chaining

Disadvantages:

- pointers increase memory requirements
- pointers may lead to bad cache efficiency

Advantages:

- no à priori limit on the number of elements
- deletion can be implemented efficiently
- by using balanced trees instead of linked list one can also obtain worst-case guarantees.

Open Addressing

All objects are stored in the table itself.

Define a function $h(k, j)$ that determines the table-position to be examined in the j-th step. The values $h(k, 0)$,. . . ,$h(k, n - 1)$ must form a permutation of $0, \ldots, n - 1$.

Search(k): Try position $h(k, 0)$; if it is empty your search fails; otw. continue with $h(k, 1)$, $h(k, 2)$, ...

Insert(x): Search until you find an empty slot; insert your element there. If your search reaches $h(k, n - 1)$, and this slot is non-empty then your table is full.

Open Addressing

All objects are stored in the table itself.

Define a function $h(k, j)$ that determines the table-position to be examined in the j-th step. The values $h(k, 0)$,...,$h(k, n-1)$ must form a permutation of $0, \ldots, n-1$.

Search(k): Try position $h(k, 0)$; if it is empty your search fails; otw. continue with $h(k, 1)$, $h(k, 2)$, ...

Insert(x): Search until you find an empty slot; insert your element there. If your search reaches $h(k, n-1)$, and this slot is non-empty then your table is full.

Open Addressing

All objects are stored in the table itself.

Define a function $h(k, j)$ that determines the table-position to be examined in the j-th step. The values $h(k, 0)$,...,$h(k, n-1)$ must form a permutation of $0, \dots, n-1$.

Search(k): Try position $h(k, 0)$; if it is empty your search fails; otw. continue with $h(k, 1)$, $h(k, 2)$, ...

Insert(x): Search until you find an empty slot; insert your element there. If your search reaches $h(k, n-1)$, and this slot is non-empty then your table is full.

Open Addressing

All objects are stored in the table itself.

Define a function $h(k,j)$ that determines the table-position to be examined in the j-th step. The values $h(k,0)$,...,$h(k,n-1)$ must form a permutation of $0, \ldots, n-1$.

Search(k): Try position $h(k,0)$; if it is empty your search fails; otw. continue with $h(k,1)$, $h(k,2)$, ...

Insert(x): Search until you find an empty slot; insert your element there. If your search reaches $h(k,n-1)$, and this slot is non-empty then your table is full.

Open Addressing

All objects are stored in the table itself.

Define a function $h(k, j)$ that determines the table-position to be examined in the j-th step. The values $h(k, 0)$,...,$h(k, n-1)$ must form a permutation of $0, \ldots, n-1$.

Search(k): Try position $h(k, 0)$; if it is empty your search fails; otw. continue with $h(k, 1)$, $h(k, 2)$, ...

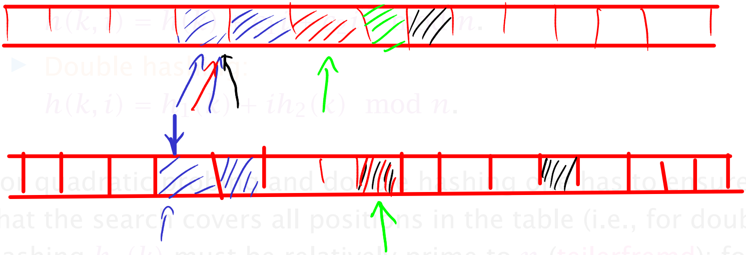
Insert(x): Search until you find an empty slot; insert your element there. If your search reaches $h(k, n-1)$, and this slot is non-empty then your table is full.

Open Addressing

1 + 2 4 8 16

Choices for $h(k,j)$:

- Linear probing:
  $h(k,i) = h(k) + i \mod n$
  (sometimes: $h(k,i) = h(k) + ci \mod n$).
- Quadratic probing:
  $h(k,i) = h(k) + c_1 i + c_2 i^2 \mod n$.
- Double hashing:
  $h(k,i) = h_1(k) + ih_2(k) \mod n$.



For quadratic probing and double hashing one has to ensure that the search covers all positions in the table (i.e., for double hashing $h_2(k)$ must be relatively prime to n (teilerfremd); for quadratic probing c_1 and c_2 have to be chosen carefully).

Open Addressing

Choices for $h(k,j)$:

- Linear probing:
  $h(k,i) = h(k) + i \mod n$
  (sometimes: $h(k,i) = h(k) + ci \mod n$).
- Quadratic probing:
  $h(k,i) = h(k) + c_1 i + c_2 i^2 \mod n$.
- Double hashing:
  $h(k,i) = h_1(k) + i h_2(k) \mod n$.

For quadratic probing and double hashing one has to ensure that the search covers all positions in the table (i.e., for double hashing $h_2(k)$ must be relatively prime to n (teilerfremd); for quadratic probing c_1 and c_2 have to be chosen carefully).

Open Addressing

Choices for $h(k,j)$:

- **Linear probing:**
  $h(k,i) = h(k) + i \mod n$
  (sometimes: $h(k,i) = h(k) + ci \mod n$).
- **Quadratic probing:**
  $h(k,i) = h(k) + c_1 i + c_2 i^2 \mod n$.
- **Double hashing:**
  $h(k,i) = h_1(k) + i\overset{1}{\overline{\underset{3}{\underline{h_2(k)}}}} \mod n$.

For quadratic probing and double hashing one has to ensure that the search covers all positions in the table (i.e., for double hashing $h_2(k)$ must be relatively prime to n (teilerfremd); for quadratic probing c_1 and c_2 have to be chosen carefully).

Open Addressing

Choices for $h(k,j)$:

- Linear probing:
  $h(k,i) = h(k) + i \mod n$
  (sometimes: $h(k,i) = h(k) + ci \mod n$).
- Quadratic probing:
  $h(k,i) = h(k) + c_1 i + c_2 i^2 \mod n$.
- Double hashing:
  $h(k,i) = h_1(k) + i h_2(k) \mod n$.

For quadratic probing and double hashing one has to ensure that the search covers all positions in the table (i.e., for double hashing $h_2(k)$ must be relatively prime to n (teilerfremd); for quadratic probing c_1 and c_2 have to be chosen carefully).

Linear Probing

- Advantage: Cache-efficiency. The new probe position is very likely to be in the cache.
- Disadvantage: Primary clustering. Long sequences of occupied table-positions get longer as they have a larger probability to be hit. Furthermore, they can merge forming larger sequences.

Lemma 21

Let L be the method of linear probing for resolving collisions:

$$L^+ \approx \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right)$$

$$L^- \approx \frac{1}{2}\left(1 + \frac{1}{(1-\alpha)^2}\right)$$

Linear Probing

- Advantage: Cache-efficiency. The new probe position is very likely to be in the cache.
- Disadvantage: Primary clustering. Long sequences of occupied table-positions get longer as they have a larger probability to be hit. Furthermore, they can merge forming larger sequences.

Lemma 21

Let L be the method of linear probing for resolving collisions:

$$L^+ \approx \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right)$$

$$L^- \approx \frac{1}{2}\left(1 + \frac{1}{(1-\alpha)^2}\right)$$

Linear Probing

- Advantage: Cache-efficiency. The new probe position is very likely to be in the cache.
- Disadvantage: Primary clustering. Long sequences of occupied table-positions get longer as they have a larger probability to be hit. Furthermore, they can merge forming larger sequences.

Lemma 21

Let L be the method of linear probing for resolving collisions:

$$L^+ \approx \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right)$$

$$L^- \approx \frac{1}{2}\left(1 + \frac{1}{(1-\alpha)^2}\right)$$

Quadratic Probing

- Not as cache-efficient as Linear Probing.
- Secondary clustering: caused by the fact that all keys mapped to the same position have the same probe sequence.

Lemma 22

Let Q be the method of quadratic probing for resolving collisions:

$$Q^+ \approx 1 + \ln\left(\frac{1}{1-\alpha}\right) - \frac{\alpha}{2}$$

$$Q^- \approx \frac{1}{1-\alpha} + \ln\left(\frac{1}{1-\alpha}\right) - \alpha$$

Quadratic Probing

- Not as cache-efficient as Linear Probing.
- Secondary clustering: caused by the fact that all keys mapped to the same position have the same probe sequence.

Lemma 22

Let Q be the method of quadratic probing for resolving collisions:

$$Q^+ \approx 1 + \ln\left(\frac{1}{1-\alpha}\right) - \frac{\alpha}{2}$$

$$Q^- \approx \frac{1}{1-\alpha} + \ln\left(\frac{1}{1-\alpha}\right) - \alpha$$

Double Hashing

- Any probe into the hash-table usually creates a cache-miss.

Lemma 23

Let A be the method of double hashing for resolving collisions:

$$D^+ \approx \frac{1}{\alpha} \ln\left(\frac{1}{1-\alpha}\right)$$

$$D^- \approx \frac{1}{1-\alpha}$$

Double Hashing

- Any probe into the hash-table usually creates a cache-miss.

Lemma 23

Let A be the method of double hashing for resolving collisions:

$$D^+ \approx \frac{1}{\alpha}\ln\left(\frac{1}{1-\alpha}\right)$$

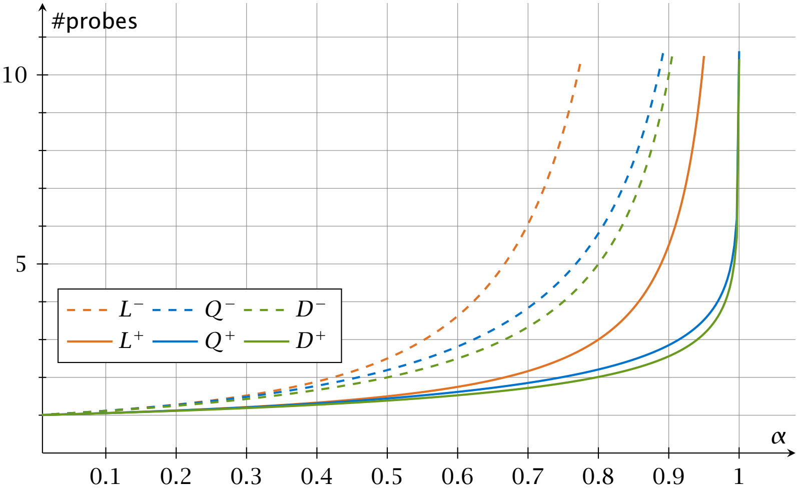
$$D^- \approx \frac{1}{1-\alpha}$$

Open Addressing

Some values:

α	Linear Probing		Quadratic Probing		Double Hashing	
	L^+	L^-	Q^+	Q^-	D^+	D^-
0.5	1.5	2.5	1.44	2.19	1.39	2
0.9	5.5	50.5	2.85	11.40	2.55	10
0.95	10.5	200.5	3.52	22.05	3.15	20

Open Addressing



Analysis of Idealized Open Address Hashing

We analyze the time for a search in a very idealized Open Addressing scheme.

- The probe sequence $h(k,0), h(k,1), h(k,2), \dots$ is equally likely to be any permutation of $\langle 0, 1, \dots, n-1 \rangle$.

Analysis of Idealized Open Address Hashing

Analysis of Idealized Open Address Hashing

Let X denote a random variable describing the number of probes in an unsuccessful search.

Analysis of Idealized Open Address Hashing

Let X denote a random variable describing the number of probes in an unsuccessful search.

Let A_i denote the event that the i-th probe occurs and is to a non-empty slot.

$$\Pr[A_1 \cap A_2 \cap \cdots \cap A_{i-1}]$$

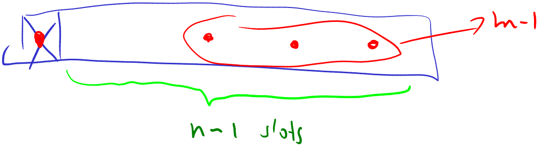
Analysis of Idealized Open Address Hashing

Let X denote a random variable describing the number of probes in an unsuccessful search.

Let A_i denote the event that the i-th probe occurs and is to a non-empty slot.

$$\Pr[A \mid B] = \frac{\Pr[A \cap B]}{\Pr[B]}$$

$$\Pr[A_1 \cap A_2 \cap \cdots \cap A_{i-1}]$$
$$= \Pr[A_1] \cdot \Pr[A_2 \mid A_1] \cdot \Pr[A_3 \mid A_1 \cap A_2] \cdot \ldots \cdot \Pr[A_{i-1} \mid A_1 \cap \cdots \cap A_{i-2}]$$



Analysis of Idealized Open Address Hashing

Let X denote a random variable describing the number of probes in an unsuccessful search.

Let A_i denote the event that the i-th probe occurs and is to a non-empty slot.

$$\begin{aligned}\Pr[A_1 \cap A_2 \cap \cdots \cap A_{i-1}]& \\ &= \Pr[A_1] \cdot \Pr[A_2 \mid A_1] \cdot \Pr[A_3 \mid A_1 \cap A_2] \cdot \\ &\qquad \ldots \cdot \Pr[A_{i-1} \mid A_1 \cap \cdots \cap A_{i-2}]\end{aligned}$$

$$\Pr[X \geq i]$$

Analysis of Idealized Open Address Hashing

Let X denote a random variable describing the number of probes in an unsuccessful search.

Let A_i denote the event that the i-th probe occurs and is to a non-empty slot.

$$\begin{aligned}\Pr[A_1 \cap A_2 \cap \cdots \cap A_{i-1}]& \\ &= \Pr[A_1] \cdot \Pr[A_2 \mid A_1] \cdot \Pr[A_3 \mid A_1 \cap A_2] \cdot \\ &\qquad \ldots \cdot \Pr[A_{i-1} \mid A_1 \cap \cdots \cap A_{i-2}]\end{aligned}$$

$$\Pr[X \geq i] = \frac{m}{n} \cdot \frac{m-1}{n-1} \cdot \frac{m-2}{n-2} \cdot \ldots \cdot \frac{m-i+2}{n-i+2}$$

Analysis of Idealized Open Address Hashing

Let X denote a random variable describing the number of probes in an unsuccessful search.

Let A_i denote the event that the i-th probe occurs and is to a non-empty slot.

$$\begin{aligned}
&\Pr[A_1 \cap A_2 \cap \cdots \cap A_{i-1}] \\
&\qquad = \Pr[A_1] \cdot \Pr[A_2 \mid A_1] \cdot \Pr[A_3 \mid A_1 \cap A_2] \cdot \\
&\qquad\qquad \ldots \cdot \Pr[A_{i-1} \mid A_1 \cap \cdots \cap A_{i-2}]
\end{aligned}$$

$$\begin{aligned}
\Pr[X \geq i] &= \frac{m}{n} \cdot \frac{m-1}{n-1} \cdot \frac{m-2}{n-2} \cdot \ldots \cdot \frac{m-i+2}{n-i+2} \\
&\leq \left(\frac{m}{n}\right)^{i-1}
\end{aligned}$$

Analysis of Idealized Open Address Hashing

Let X denote a random variable describing the number of probes in an unsuccessful search.

Let A_i denote the event that the i-th probe occurs and is to a non-empty slot.

$$\begin{aligned}\Pr[A_1 \cap A_2 \cap \cdots \cap A_{i-1}]& \\ &= \Pr[A_1] \cdot \Pr[A_2 \mid A_1] \cdot \Pr[A_3 \mid A_1 \cap A_2] \cdot \\ &\quad \ldots \cdot \Pr[A_{i-1} \mid A_1 \cap \cdots \cap A_{i-2}]\end{aligned}$$

$$\begin{aligned}\Pr[X \geq i] &= \frac{m}{n} \cdot \frac{m-1}{n-1} \cdot \frac{m-2}{n-2} \cdot \ldots \cdot \frac{m-i+2}{n-i+2} \\ &\leq \left(\frac{m}{n}\right)^{i-1} = \alpha^{i-1} \ .\end{aligned}$$

Analysis of Idealized Open Address Hashing

$\mathrm{E}[X]$

Analysis of Idealized Open Address Hashing

$$E[X] = \sum_{i=1}^{\infty} \Pr[X \geq i]$$

Analysis of Idealized Open Address Hashing

$$\mathrm{E}[X] = \sum_{i=1}^{\infty} \Pr[X \geq i] \leq \sum_{i=1}^{\infty} \alpha^{i-1}$$

Analysis of Idealized Open Address Hashing

$$\mathrm{E}[X] = \sum_{i=1}^{\infty} \Pr[X \geq i] \leq \sum_{i=1}^{\infty} \alpha^{i-1} = \sum_{i=0}^{\infty} \alpha^{i}$$

Analysis of Idealized Open Address Hashing

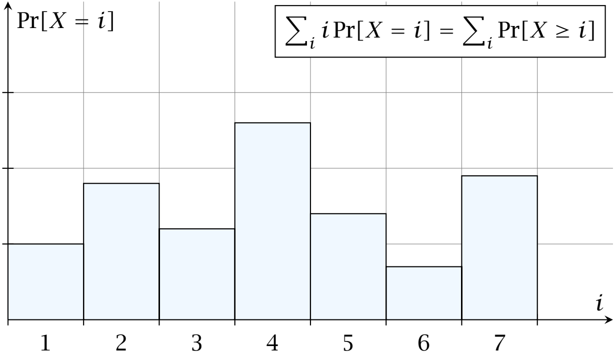
$$\mathrm{E}[X] = \sum_{i=1}^{\infty} \Pr[X \geq i] \leq \sum_{i=1}^{\infty} \alpha^{i-1} = \sum_{i=0}^{\infty} \alpha^{i} = \frac{1}{1-\alpha} \enspace .$$

Analysis of Idealized Open Address Hashing

$$\mathrm{E}[X] = \sum_{i=1}^{\infty} \Pr[X \geq i] \leq \sum_{i=1}^{\infty} \alpha^{i-1} = \sum_{i=0}^{\infty} \alpha^{i} = \frac{1}{1-\alpha} \ .$$

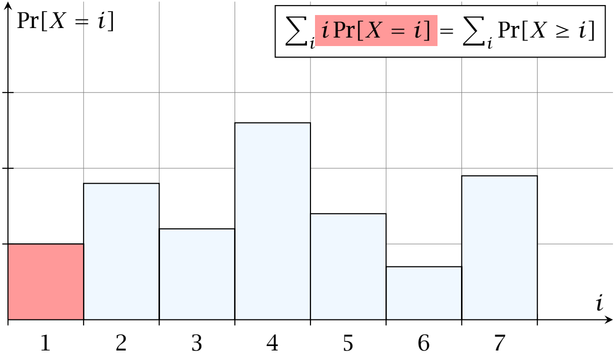
$$\frac{1}{1-\alpha} = 1 + \alpha + \alpha^2 + \alpha^3 + \dots$$

Analysis of Idealized Open Address Hashing



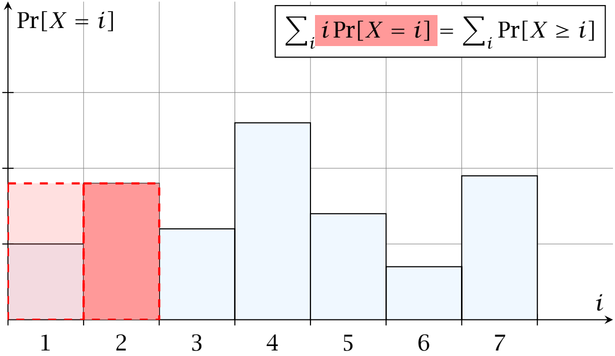
Analysis of Idealized Open Address Hashing

$i = 1$



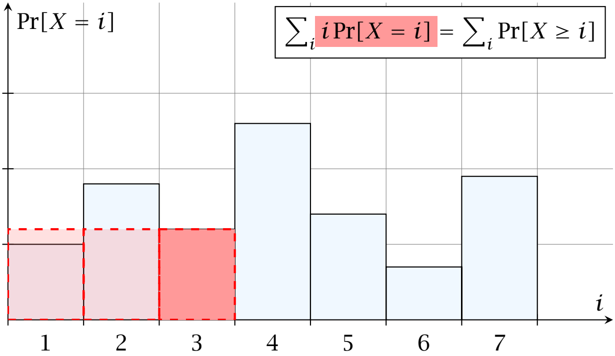
Analysis of Idealized Open Address Hashing

$i = 2$



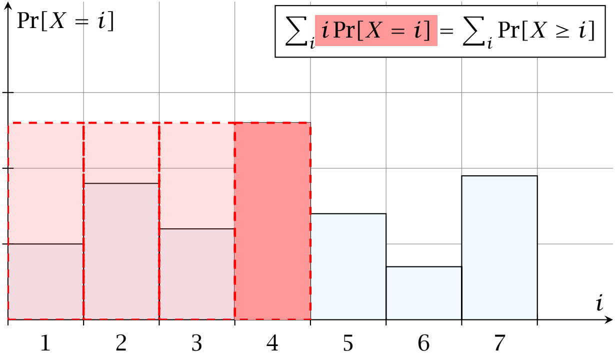
Analysis of Idealized Open Address Hashing

$i = 3$



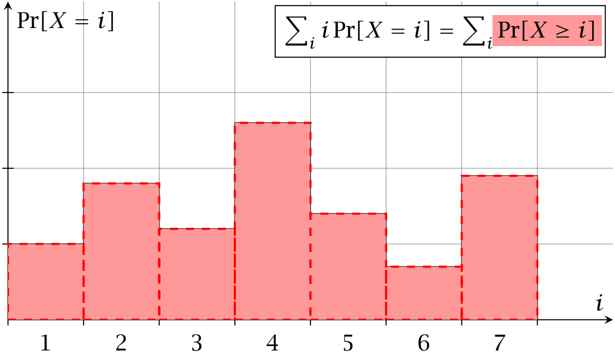
Analysis of Idealized Open Address Hashing

$i = 4$



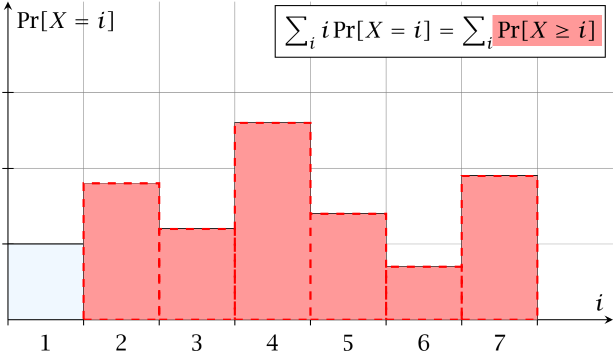
Analysis of Idealized Open Address Hashing

$i = 1$



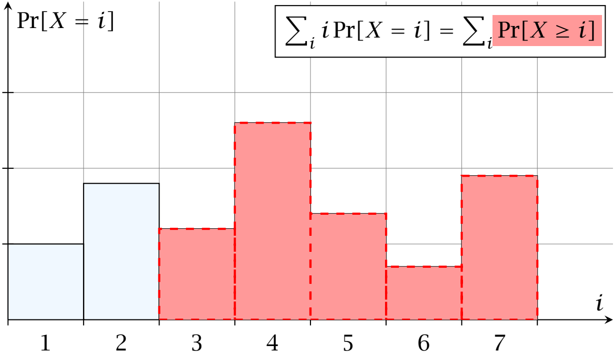
Analysis of Idealized Open Address Hashing

$i = 2$



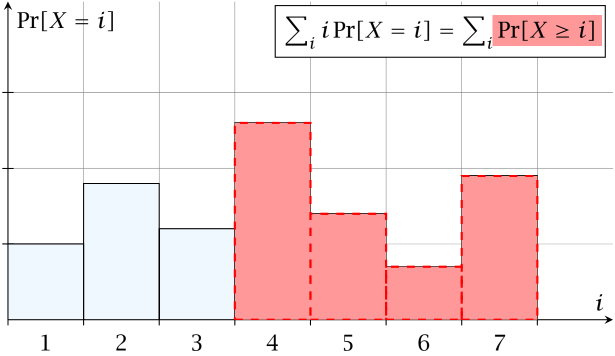
Analysis of Idealized Open Address Hashing

$i = 3$

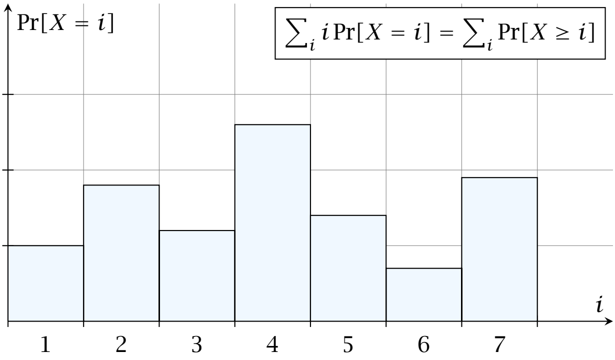


Analysis of Idealized Open Address Hashing

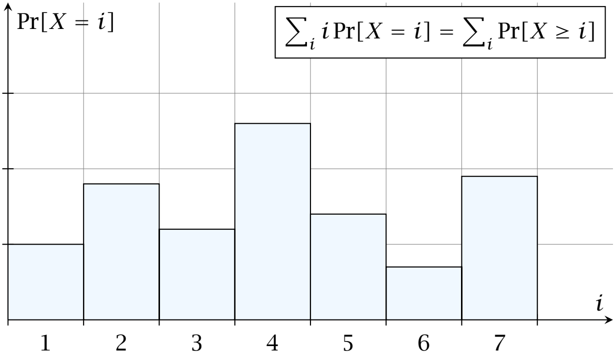
$i = 4$



Analysis of Idealized Open Address Hashing



Analysis of Idealized Open Address Hashing



The j-th rectangle appears in both sums j times. (j times in the first due to multiplication with j; and j times in the second for summands $i = 1, 2, \ldots, j$)