

Great Ideas in Computing

Allan Borodin

University of Toronto CSC196
Fall 2026

Class 9: October 7 (2026)

Announcements and Today's Agenda

Announcements

- Thanksgiving is this coming Monday so no classes.
- I have posted Assignment 2. It is due October 21.
I posted the final question today. Hopefully by the end of today you should be able to do all the questions on the assignment.
- Elise can answer questions in this weeks tutorial. The first term test will be similar to the first two assignments.
- Looking ahead to the next set of possible topics/great ideas.

Agenda

- Comments on Guest Presentation on Quantum Computation
- Hash Tables and the Birthday Paradox: another way to implement a Dictionary.

A hash table: One more way to implement a dictionary

We have a hash function $h : I \rightarrow M$ where $I = \{ID_1, \dots, ID_N\}$ is the set of all possible keys (i.e., integer identifiers) and $M = \{A[0], \dots, A[m-1]\}$ is a relatively small set of memory locations. M can be an array (i.e., a consecutive set of words in the memory, but in general M can be any set of memory locations that are reserved for the data structure.

That is, we are going to hash each of the $N = |I|$ possible items to a small set of $m = |M|$ memory locations.

Here we can have $N \gg n$ where n is the actual number of items we are storing.

What is a suitable hash function h ?

A hash table: One more way to implement a dictionary

We have a hash function $h : I \rightarrow M$ where $I = \{ID_1, \dots, ID_N\}$ is the set of all possible keys (i.e., integer identifiers) and $M = \{A[0], \dots, A[m-1]\}$ is a relatively small set of memory locations. M can be an array (i.e., a consecutive set of words in the memory, but in general M can be any set of memory locations that are reserved for the data structure.

That is, we are going to hash each of the $N = |I|$ possible items to a small set of $m = |M|$ memory locations.

Here we can have $N \gg n$ where n is the actual number of items we are storing.

What is a suitable hash function h ?

One possibility is $h(ID) = (a \cdot ID + b)(\text{mod } p)(\text{mod } m)$ for some $a \neq b$ and p where p is a large prime.

A hash table

Ignoring conflicts in the hash table, we can search in constant time for a particular item. How would we search for a record given its key?

A hash table

Ignoring conflicts in the hash table, we can search in constant time for a particular item. **How would we search for a record given its key?**

Need to deal with conflicts; i.e multiple items hashing to the same place in the hash table. When there is a conflict, one possibility is to use a pointer to a linked list containing the IDs that have been matched to the same place in the hash table.

A hash table

Ignoring conflicts in the hash table, we can search in constant time for a particular item. **How would we search for a record given its key?**

Need to deal with conflicts; i.e multiple items hashing to the same place in the hash table. When there is a conflict, one possibility is to use a pointer to a linked list containing the IDs that have been matched to the same place in the hash table.

Note: When we draw random numbers in the execution of an algorithm, we are not drawing truly random numbers. The generation of pseudo random numbers and pseudo random functions is an interesting and substantial topic, one related to complexity theory and cryptography, two of our possible future topics.

A hash table

Ignoring conflicts in the hash table, we can search in constant time for a particular item. **How would we search for a record given its key?**

Need to deal with conflicts; i.e multiple items hashing to the same place in the hash table. When there is a conflict, one possibility is to use a pointer to a linked list containing the IDs that have been matched to the same place in the hash table.

Note: When we draw random numbers in the execution of an algorithm, we are not drawing truly random numbers. The generation of pseudo random numbers and pseudo random functions is an interesting and substantial topic, one related to complexity theory and cryptography, two of our possible future topics.

Conceptually, we think of a hash function as *randomly* placing a record (i.e., its key) in a memory location. If the number n of actual keys in our data structure is less than m , the size of the hash table, we would like the $\text{Prob}[h(ID_1) = h(ID_2) | ID_1 \neq ID_2]$ to be small. **Note:** We expect $N = |I|$ to be much larger than n .

Hash tables in terms of balls and bins

One way to think about a *perfect* random hash function is that we are randomly and independently throwing n balls (keys) at m bins (memory location), with perfect randomness. With perfect randomness, each time we throw a ball it is equally likely to go to any bin; that is

$Prob[h(i) = j] = \frac{1}{m}$ for all i, j .

Supposed $m \geq n$.

What is the average (averaging over the m bins) of the number of balls in a bin?

Hash tables in terms of balls and bins

One way to think about a *perfect* random hash function is that we are randomly and independently throwing n balls (keys) at m bins (memory location), with perfect randomness. With perfect randomness, each time we throw a ball it is equally likely to go to any bin; that is

$$\text{Prob}[h(i) = j] = \frac{1}{m} \text{ for all } i, j.$$

Supposed $m \geq n$.

What is the average (averaging over the m bins) of the number of balls in a bin?

What does that suggest for the average time to search or update a record?

Hash tables in terms of balls and bins

One way to think about a *perfect* random hash function is that we are randomly and independently throwing n balls (keys) at m bins (memory location), with perfect randomness. With perfect randomness, each time we throw a ball it is equally likely to go to any bin; that is

$$\text{Prob}[h(i) = j] = \frac{1}{m} \text{ for all } i, j.$$

Supposed $m \geq n$.

What is the average (averaging over the m bins) of the number of balls in a bin?

What does that suggest for the average time to search or update a record?

What is the expected maximum number of balls in a bin?

Suppose $n \ll m$ (i.e., $n = o(M)$), how likely are we to have a collision. That is, how likely is it that two distinct keys will be hashed into the same memory location?

The Birthday Paradox

The birthday paradox: In probability theory, the birthday problem or birthday paradox concerns the probability that, in a set of n randomly chosen people, some pair of them will have the same birthday. By the pigeonhole principle, the probability reaches 100% when the number of people reaches 367 (since there are only 366 possible birthdays, including February 29).

However, 99.9% probability is reached with just 70 people, and 50% probability with 23 people. These conclusions are based on the assumption that each day of the year (excluding February 29) is equally probable for a birthday.

In more general terms, when $n \approx \sqrt{m}$, there is likely to be two balls being hashed into the same bin (even with perfect hashing).

End of discussion of hash tables

Suppose we are hashing n keys into $m = n$ locations. How long do we expect the longest chain of collisions might be?

End of discussion of hash tables

Suppose we are hashing n keys into $m = n$ locations. How long do we expect the longest chain of collisions might be?

With probability $1 - o(1)$, the expected load on the most populated bin is $\frac{\log n}{\log \log n}(1 + o(1))$. Here the $o(1)$ indicates a function $\ell(n) : \ell(n) \rightarrow 0$ as $n \rightarrow \infty$.

We can have shorter collision chains if we think of using a different second hash function to try to resolve a collision. As a balls and bins experiment (i.e., try two different hash functions and map the key into the least loaded bin), this would result in the expected load on the most populated bin to be $\approx \log \log n$ when $m = n$.

What are the advantages and disadvantages of hash tables compared to a binary search tree when implementing a Dictionary?

End of discussion of hash tables

Suppose we are hashing n keys into $m = n$ locations. How long do we expect the longest chain of collisions might be?

With probability $1 - o(1)$, the expected load on the most populated bin is $\frac{\log n}{\log \log n}(1 + o(1))$. Here the $o(1)$ indicates a function $\ell(n) : \ell(n) \rightarrow 0$ as $n \rightarrow \infty$.

We can have shorter collision chains if we think of using a different second hash function to try to resolve a collision. As a balls and bins experiment (i.e., try two different hash functions and map the key into the least loaded bin), this would result in the expected load on the most populated bin to be $\approx \log \log n$ when $m = n$.

What are the advantages and disadvantages of hash tables compared to a binary search tree when implementing a Dictionary?

You should think in terms of “average” time and worst case time for the operations we need to perform in a Dictionary?

End of discussion of hash tables

Suppose we are hashing n keys into $m = n$ locations. How long do we expect the longest chain of collisions might be?

With probability $1 - o(1)$, the expected load on the most populated bin is $\frac{\log n}{\log \log n}(1 + o(1))$. Here the $o(1)$ indicates a function $\ell(n) : \ell(n) \rightarrow 0$ as $n \rightarrow \infty$.

We can have shorter collision chains if we think of using a different second hash function to try to resolve a collision. As a balls and bins experiment (i.e., try two different hash functions and map the key into the least loaded bin), this would result in the expected load on the most populated bin to be $\approx \log \log n$ when $m = n$.

What are the advantages and disadvantages of hash tables compared to a binary search tree when implementing a Dictionary?

You should think in terms of “average” time and worst case time for the operations we need to perform in a Dictionary? What if a key is missing?