CSCI 136: 
Data Structures 
and 
Advanced Programming 
Lecture 27 
Hash tables 
Instructor: Dan Barowy 
Williams

Topics

Hash tables 
Hash functions

Your to-dos

1. Read **before Wed**: Bailey, Ch 16-16.2. 
2. Lab 9 (partner lab), due Tuesday 11/29 by 10pm. 
3. Quiz, due Saturday evening.

Announcements

Computer Science Colloquium 
Friday, November 18 @ 2:35pm in Wege (TCL 123) 
Daniel Malinsky (Columbia) 
Identifying Causal Determinants of Clinical Outcomes from 
Electronic Health Records Using Graphical Structure Learning: 
Challenges and Opportunities in Causal Discovery 

Many goals within causal inference, including estimating average 
treatment effects and understanding path-specific mechanisms, 
depend on knowing the qualitative causal structure underlying a 
domain. In this work we apply methods for graphical causal 
discovery (specifically the FCI algorithm) to observational data in the 
form of electronic health records (EHR) from Johns Hopkins Hospital. Our goal is to understand the causal determinants of 
postoperative length of stay for patients undergoing cardiac surgery procedures, in order to inform possible interventions that support 
faster patient recovery. We discuss the challenges in applying causal 
discovery methods to electronic health records and opportunities for 
future work.
Hash tables
My favorite data structure

Note about lab 9:
You should use the structure\texttt{Hashtable} implementation.
But if you want the extra challenge, implement your own!

Recall: arrays
An array is a data structure consisting of a \textit{sequential collection of elements}, each identified by an \textit{index}.

\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
13 & 2 & 451 & 42 & 9 & 6 & -4 & 8 \\
\end{array}

Performance guarantees:
1. \textbf{read} an element: $O(1)$
2. \textbf{write} an element: $O(1)$

Can we capture some of this for a more general structure?

Generalization: associative array
An associative array or is a data structure consisting of a \textit{sequential collection of elements}, each identified by a \textit{key}.
An associative array is a \textit{map}.

\begin{array}{cccccccc}
\text{A} & \text{B} & \text{C} & \text{D} & \text{E} & \text{F} & \text{G} & \text{H} \\
\hline
\text{Joe} & \text{Adam} & \text{Sue} & \text{Ed} & \text{Sam} & \text{Fay} & \text{Dan} & \text{Ted} \\
\end{array}

Performance guarantees:
1. \textbf{read} an element: $O(1)$?
2. \textbf{write} an element: $O(1)$?

\textbf{How} can we make this happen?
What about MapTree?

It is already a map, which is good, but…

Performance guarantees:
1. read an element: \(O(\log n)\) (assuming balance)
2. write an element: \(O(\log n)\) (assuming balance)
   
   Not fast enough!

Could we actually just use an array?

What do you think? What’s the obstacle?

Need: function to map key to index

Suppose we have a function:

\[ h(k) \rightarrow z \]

where \( k \) is a key of arbitrary type and \( z \in \mathbb{Z}_0^+ \),

then we could construct another function:

```c
int index(K key) {
    return h(key) \% A.length;
}
```

Hash function

A hash function is any function that can be used to map data of arbitrary size onto data of a fixed size.

Why not “Benedict Cumberbatch”? 
Hash table

A hash table is a data structure that implements the map abstract data type. A hash table uses a hash function to compute an index into an array of buckets, from which the desired value can be found.

```
   A
0 1 2 3 4 5 6 7
```

“Dan”, -4
index(“Dan”) → 6
A[index(“Dan”)] = -4

Hash function

Hash functions must also provide the following guarantees:

**Determinism**: a given input value must always generate the same hash value.

**Uniformity**: maps the expected inputs as evenly as possible over its output range.

**Equivalence**: any two values that are considered equivalent should produce the same hash value.

Question

Is a function that generates a random number a good hash function?

**No.** Random numbers do tend to be uniform, but are not deterministic.

Activity

See if you can come up with a simple hash function for strings.

**Determinism**: a given input value must always generate the same hash value.

**Uniformity**: maps the expected inputs as evenly as possible over its output range.

**Equivalence**: any two values that are considered equivalent should produce the same hash value.
Recap & Next Class

Today:

Hash tables
Hash functions

Next class:

Collisions
Graphs