Comp 363 - Design and Analysis of Computer Algorithms

Spring Semester 2020 - Week 10 - Part 1

Dr Nick Hayward

Hash Tables - intro

- many concepts to consider as we review *hash tables*
- e.g.
  - *initial implementation*
  - *collisions*
  - *hash functions*
  - *performance*
  - *...*
- useful to begin with a conceptual example
  - *helps review hash table*
  - *underlying functionality*
  - *...*

Algorithms and Data Structures

hash tables - manage a bookshop - part 1

- initial example set in a *bookshop*
  - *e.g. currently manage a bookshop*
  - *many valuable first editions*
  - *lower cost paperback publications*
  - *latest releases...*
- someone wants to purchase a book
  - *need to check price in a register*
- register contains variant prices
  - *different editions, publications*
  - *each book in the shop*

Algorithms and Data Structures

hash tables - manage a bookshop - part 2

- if register is **not** organised in alphabetical order
  - *may take a long time to check every entry for the required book*
- involves a *simple search*
  - *seller checks every line of the register*
  - *time of* `O(n)`*, linear time, not profitable for shop*
- if register is ordered alphabetically
  - *may then run a binary search to find required price*
  - *complexity will now fall to a time of* `O(log n)`*, logarithmic time*

Algorithms and Data Structures

hash tables - manage a bookshop - part 3

- a quick comparison for searching required items in register

items in register	O(n)	O(log n)
100	10 seconds	1 second (7 lines - check $\log_2 100$)
1000	1.66 minutes	1 second (10 lines - check $\log_2 1000$)
10000	16.6 minutes	2 seconds (14 lines = check ($\log_2 10000$)

- *binary search* is faster option for this type of register
  - *i.e. compared to simple search*
  - *still annoying to search through register for each requested book purchase*
- to help manage this register
  - *might initially consider an array*
  - *each item will need to store book's title and its price*
- if we then sort this array by title
  - *use binary search to find associated price*
  - *gives a time of* $O(\log n)$*, logarithmic time*
- need a way to query register and return price of book in $O(1)$ time
  - *instead of using a default array*
  - *we'll try implementing hash functions...*

Algorithms and Data Structures

hash tables - hash functions

- consider a *hash function*
  - *a simple concept*
  - *input a string, return an output number*
  - *hash returned*
- conceptual usage
  - *define a string as input data for query as a sequence of bytes*
- may define this usage of a hash function as *mapping strings to numbers*
- to help with this mapping
  - *some requirements for a hash function*
- e.g.
  - ***consistency** - needs to ensure input string always returns same number*
    - i.e. without predictable mapping, hash table will not work...
  - ***mapping** - hash function should map different words to different numbers*
    - i.e. function will be no use if it simply returns 7 for each input string
    - best case will allow every string to map to a different number

Algorithms and Data Structures

hash tables - basic implementation - input - part 1

- example usage allows us to implement desired query
  - *e.g. query register in bookshop*
  - *try to achieve querying for a book's price in* `O(1)` *time*
- begin with an empty array

```
------------------------
|   |   |   |   |   |   |
------------------------
| 0 | 1 | 2 | 3 | 4 | 5 |
------------------------
```

- use array to store bookshop's prices

Algorithms and Data Structures

hash tables - basic implementation - input - part 2

- start by adding a price for a book title
  - *e.g. The Glass Bead Game*
- pass this title to *hash function*
- hash function returns number 3
- use this number to store title's price at index 3 in array

```
----------------------------
|   |   |   | 7.95 |   |   |
----------------------------
| 0 | 1 | 2 |  3   | 4 | 5 |
----------------------------
```

Algorithms and Data Structures

hash tables - basic implementation - input - part 3

- input title **Hannibal's Footsteps** in hash function
  - *returns numerical value of 1*
  - *store price of title at index 1 in array*

```
------------------------------
|   | 18.95 |   | 7.95 |   |   |
------------------------------
| 0 |   1   | 2 |  3   | 4 | 5 |
------------------------------
```

- continue this pattern of input
  - *able to input each title in bookshop's register in hash function*
  - *then store associated price in array*
  - *an array full of prices...*

Video - Algorithms and Data Structures

hash tables - real-world usage - part 1

Hash tables - real-world usage examples - UP TO 4:21

Source - Hash tables - real-world usage - YouTube

Algorithms and Data Structures

hash tables - basic implementaton - query - part 1

- to retrieve a price for a title in bookshop's register
  - *do not need to search through array*
- pass title to hash function
  - *function returns a number*
- number will follow earlier rules
  - *provide consistent value for input title*
- e.g. 3 for input **The Glass Bead Game**
  - *use to get price from array*
- hash function returns where price is stored
  - *i.e. without need to search data structure*

Algorithms and Data Structures

hash tables - basic implementaton - query - part 2

- pattern works because *hash function* adheres to defined requirements
- consistently maps input string to same numbe
  - *i.e. index in array for stored value, e.g. price*
  - *input string once to get initial number for index position in array*
  - *input string whenever price is needed for title from array*
- maps different strings to different numbers
  - *every variant input string will map to different index position in array*
  - *each price may now be stored in array…*
- it knows size of array
  - *i.e. its maximum size*
  - *only returns valid numbers for index*
- now have a hash function and an array
  - *combine to produce required **hash table***
- data structure with added logic for default implementation and usage
  - *noticeable difference with default arrays and lists*
  - *customarily map straight to memory….*
- hash table uses hash function to calculate and record element storage

hash tables - programming usage

- hash tables are a useful and powerful option
  - *e.g. organising data with fast retrieval*
- also referenced as *hash maps*, *maps*, *dictionaries*, and *associative arrays*
- each input query
  - *e.g. book title from bookshop's register*
  - *returned instantly from underlying array for hash table*
  - *considering memory usage, system access &c.*

Algorithms and Data Structures

hash tables - programming usage - general usage

- for most applications
  - *no need to implement your own hash tables*
- e.g. Python
  - *implements hash tables, referenced as dictionaries*
- example usage creates new hash table with function `dict`

```
bookshop = dict()
```

- `bookshop` is new hash table
  - *store book titles and associated prices*

```
bookshop["The Glass Bead Game"] = 7.95
bookshop["Hannibal's Footsteps"] = 18.95
```

- populate hash table with titles and prices

```
{'The Glass Bead Game': 7.95, "Hannibal's Footsteps": 18.95}
```

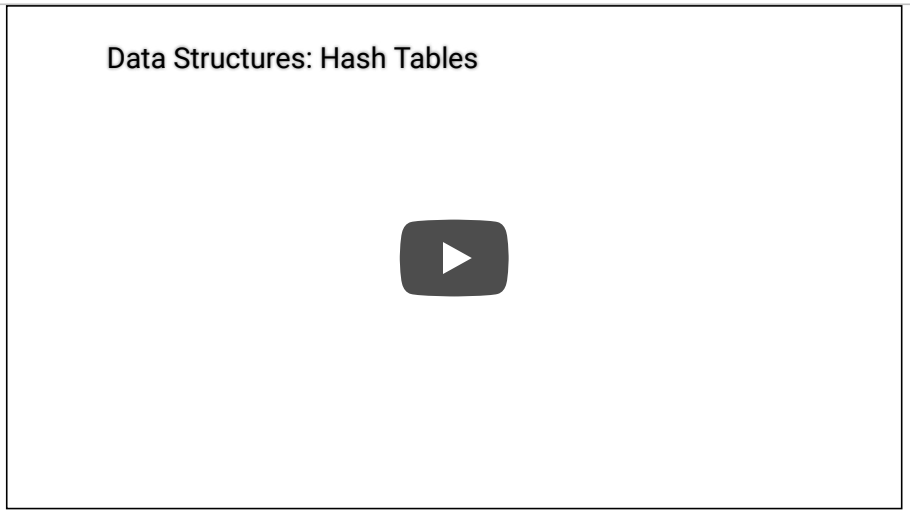
- retrieve price for stored title in hash table

```
print(bookshop["The Glass Bead Game"])
7.95
```

- clear mapping of keys to values in current *hash table*

Video - Algorithms and Data Structures

hash tables - part 3



Hash tables - Java usage - UP TO 4:42

Source - Hash tables - Java - YouTube

Algorithms and Data Structures

hash tables - programming usage - usage cases - part 1

- common use of hash tables is lookup of associative data sets
  - *e.g. username and ID or name and address &c.*
- consider briefly an *address book*
  - *need to map people's names to addresses, phone numbers, email addresses &c.*
- need a convenient and reliable way to execute functionality
  - *add a name - associate address &c. with specific name*
  - *enter a name - find and return address details associated with name*
- quickly see how a *hash table* is an ideal option for this type of usage
- e.g.
  - *create a map from name to address information*
  - *query and return associated data*

Algorithms and Data Structures

hash tables - programming usage - usage cases - part 2

- create a simple address book using a *hash table* with Python

```
# create hash table for address book
address_book = dict()

# add some entries and addresses
address_book["daisy"] = "dawlish"
address_book["emma"] = "cannes"

# check return of address for daisy...
print(address_book["daisy"])
```

- similar lookups used at a larger scale for various real-world uses
  - *e.g. perform a query to a domain name*
  - *query IP address for host server*
  - *URL is translated to an IP address from a lookup*
- lookup process is known as *DNS resolution*
  - *hash tables are one way to provide this type of functionality*

Video - Algorithms and Data Structures

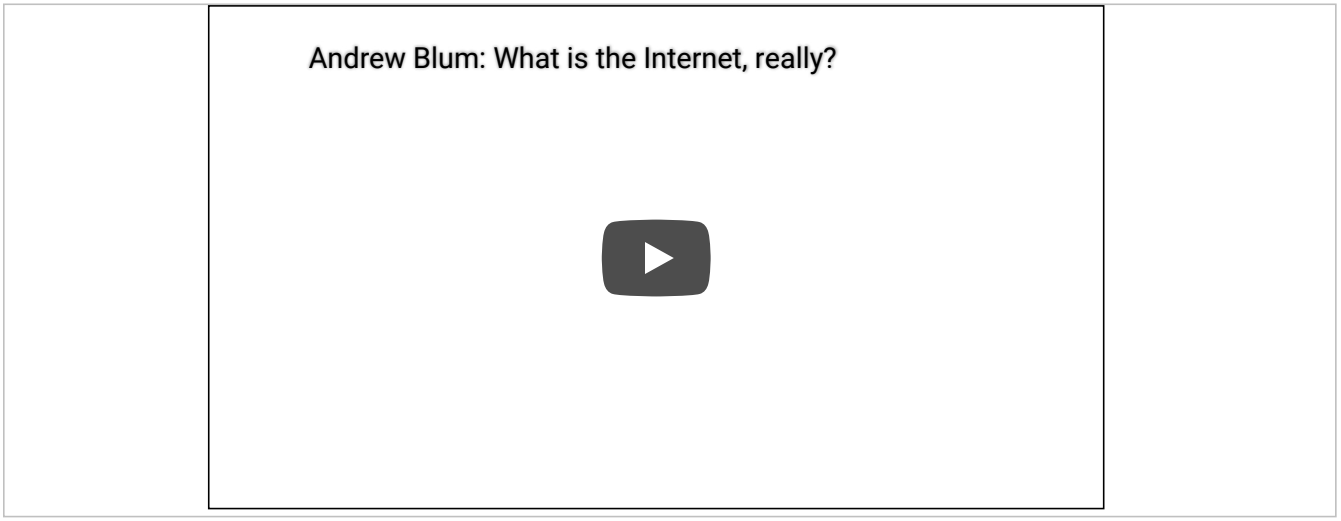
What is DNS?

What is DNS?

Source - What is DNS? - YouTube

Video - Algorithms and Data Structures

What is the Internet?



What is the Internet? Undersea cables... UP TO 11:15

Source - TED - What is the Internet? - YouTube

Resources

- Hash tables - Java - YouTube
- Hash tables - real-world usage - YouTube
- TED - What is the Internet? - YouTube
- What is DNS? - YouTube