Comp 460 - Algorithms & Complexity

Spring Semester 2020 - Week 10

Dr Nick Hayward

tables - intro

- *Hash tables* are a particularly useful, and fast, data structure
- conceptually define a hash table data structure as follows
  - *store each item in an easily determined location*
    - so no need to search for item
  - *no ordering to maintain*
    - for insertion and deletion of items
- this data structure has impressive performance
  - *i.e. time is concerned*
- there is a tradeoff with additional memory requirements
  - *conceptually harder implementation for custom patterns*

Algorithms and Data Structures

abstract data type - intro

- storage options and patterns described conceptually as an abstract data type
- need to define a specification for this particular abstract data type
- after defining specification
  - *then choose data structure*
- data structure as foundation for this implementation

Algorithms and Data Structures

abstract data type - table

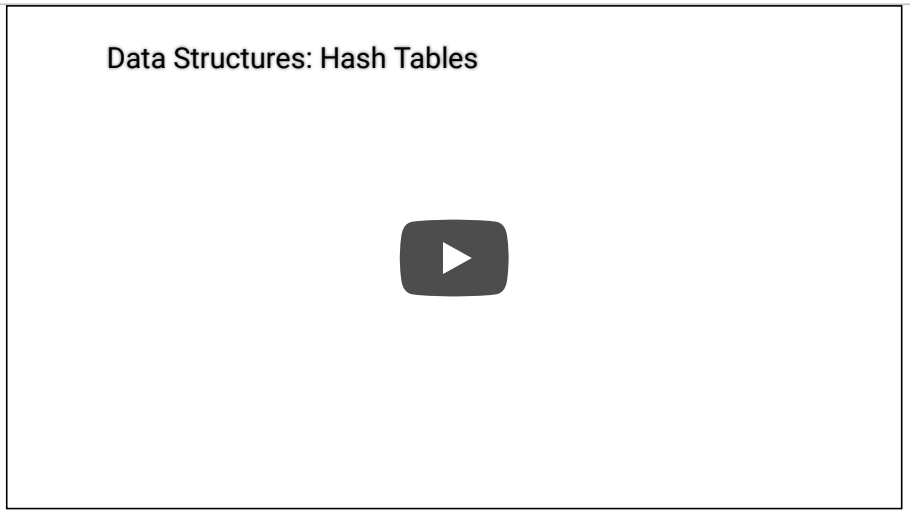
- initial specification for abstract data type outlined
- table may be used to store objects
  - *e.g.*

id	place	country
5	philae	egypt
21	athens	greece
37	rome	italy
24	sparta	greece

- objects may be arbitrarily complicated, e.g.
  - *each object has unique key*
  - *keys may be compared for equality*
  - *keys used to identify objects*
- assume there are methods &c. for the following
  - *check for empty or full table*
  - *insert object into table - assuming table is not already full*
  - *for a given key - retrieve object for that key*
  - *for a given key - update object for that key*
    - commonly replace current object at key with new object
  - *for a given key - delete object for that key*
    - assumes key is already stored in table
  - *traverse or list each item in current table*
    - if order defined - should follow increasing order…
- outline predicated on simple assumption
  - *each object is uniquely identified by its key*

Video - Algorithms and Data Structures

hash tables - part 1



Hash tables - intro - UP TO 1:15

Source - Hash tables - intro - YouTube

implementations of table data structure - intro

- consider three common approaches for implementation
  - *custom design and development of table data structure*
- e.g. might use one of the following options
  - *sorted arrays*
  - *binary search trees*
  - *hash tables*

Algorithms and Data Structures

implementations of table data structure - sorted array implementation - part 1

- if we choose a sorted *array* for *table* data structure
  - *determine full or empty in constant time* `O(1)`
  - *assuming we maintain a variable for its size*
- insert an element
  - *need to find its correct position*
  - *on average takes same time as finding an element*
- find an element
  - *crucial for all operations except traversal itself*
  - *use binary search*
  - *e.g. takes* `O(log n)`*, logarithmic time*
- consider complexity for *retrieval* and *update*
  - *also produce* `O(log n)`*, logarithmic, times*

Algorithms and Data Structures

implementations of table data structure - sorted array implementation - part 2

- if we need to delete or insert an item
  - *need to shift following element*
  - *left for delection*
  - *right for insertion*
- e.g.

```
[3, 6, 2, 33. 17. 97]
```

- delete node 33
  - *element 17 will need to shift to its left*
- insert node at position of node 2
  - *element 33 &c. will need to shift to the right*
- takes average n/2 steps
  - *such operations will have a complexity of O(n).*
- ordered traversal is simple for this type of data structure
  - *may also see complexity of O(n).*

Algorithms and Data Structures

implementations of table data structure - binary search tree implementation

- alternative to sorted arrays might use *binary search trees*
- whilst this is certainly possible
  - *worst case may also produce a tree that is very deep and narrow*
- unbalanced trees will have *linear* complexity for lookups
- *self-balancing binary search tree*
  - *able to produce a worst case same as average case*
- for such trees commonly see time complexity of `O(log n)`, logarithmic
  - *for insertion, deletion, search, retrieval, and update*
  - *may also see complexity of `O(n)`, linear, for traversal*
- downside of such self-balancing trees
  - *sheer complexity of implementation, management, and initial comprehension*

Video - Algorithms and Data Structures

stable marriage problem

A fun diversion - Stable Marriage Problem - UP TO 27:07

Source - Stable Marriage Problem - YouTube

Further details - The Stable Marriage Problem

implementations of table data structure - hash table implementation

- hash tables
  - *provide a benefit for such table data structure usage*
- may expend more space
  - *i.e. than actually required or necessary*
- extra space may also be beneficial
  - *i.e. speed up inherent operations of the table*

Video - Algorithms and Data Structures

hash tables - part 2

Hash tables - hash function and index - UP TO 2:33

Source - Hash tables - hash function - YouTube

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*

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}
```

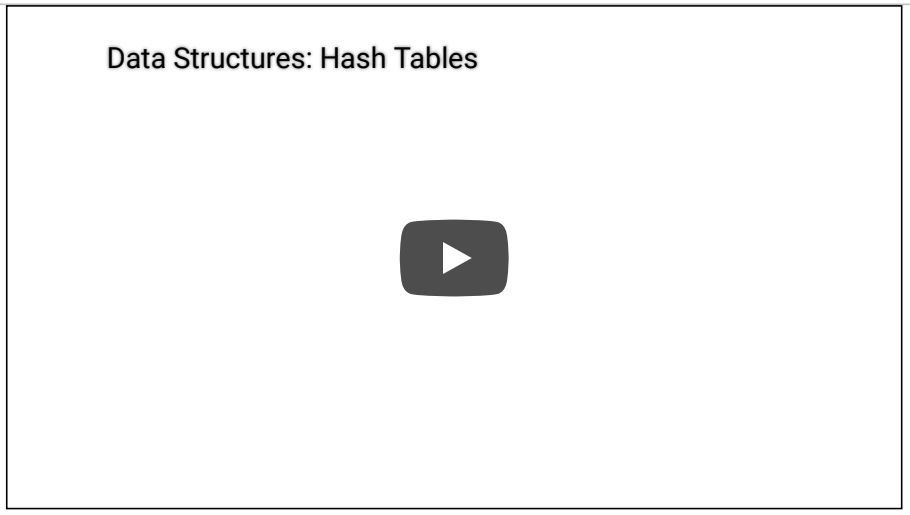
- retrieve price for stored title in hash table

```
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

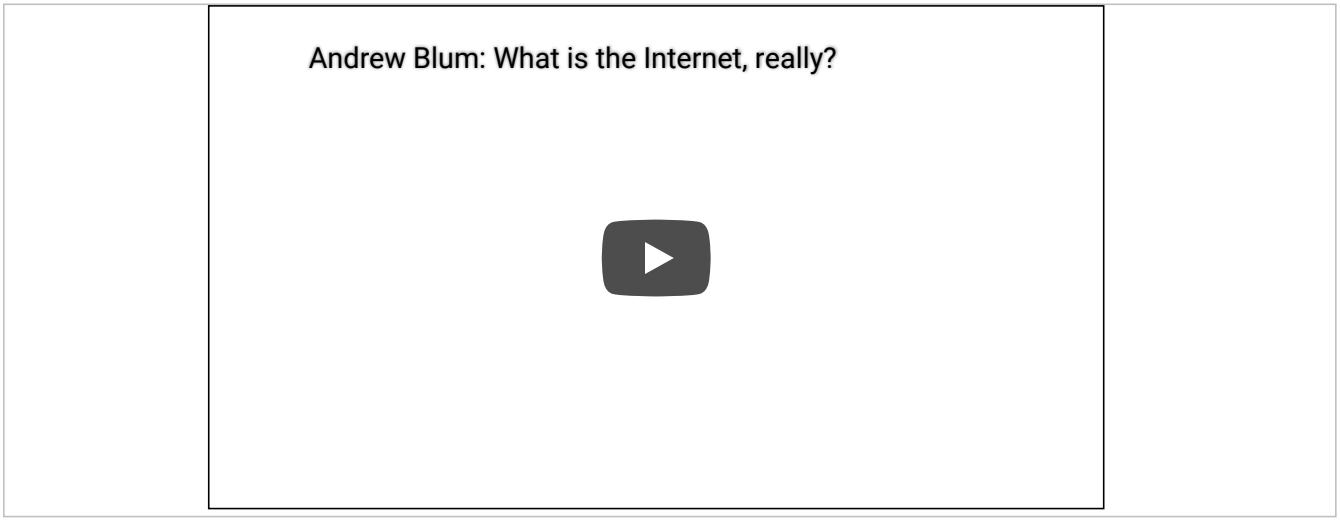
What is DNS?

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

various

- The Stable Marriage Problem

videos

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