Lab : Searching

Goals

* Continue to gain proficiency at top-down design and bottom-up implementation.
* Practice reading data from a file.
* Practice using a list of lists to store data.
* Understand how to adapt linear search and binary search to real-world data.

Video game dataset

For this this assignment, we will be working with a real-world dataset from [CORGIS](https://corgis-edu.github.io/corgis/csv/) (which is an acronym for: The Collection of Really Great, Interesting, Situated Datasets). We will be using a [dataset](https://corgis-edu.github.io/corgis/csv/video_games/) that contains records on over a thousand video games released between 2004 and 2008. This includes a variety of information on consoles, formats, sales, review scores, and average playtime. While this dataset contains many features, but we will be focusing on only a small subset of these features.

The video game data is stored in CSV format and is located at: /data/cs21/videogames/video\_games.csv

Here are the first five lines and last five lines from this file:

Title,MaxPlayers,Genre,ReviewScore,Sales,UsedPrice,Console,Rating,Year,PlayLength

AC/DC Live: Rock Band - Track Pack,4,Action / Simulation,63,0.19,14.95,X360,M,2008,3.933333333

AC/DC Live: Rock Band - Track Pack,1,Action / Simulation,60,0.19,14.95,PS3,E,2008,3.933333333

Ace Combat 6: Fires of Liberation,1,Action / Simulation,80,0.58,19.95,X360,E,2007,13.75

Ace Combat X: Skies of Deception,1,Action / Simulation,75,0.32,22.95,PSP,E,2006,4.116666667

...

Zendoku,1,Role-Playing (RPG) / Strategy,68,0.04,16.95,DS,E,2007,4

Zoids Assault,1,Role-Playing (RPG) / Strategy,46,0.07,17.95,X360,M,2007,12.05

Zoo Keeper,1,Action,74,0.1,14.95,DS,T,2004,5

Zoo Tycoon DS,1,Simulation / Strategy,44,0.92,12.95,DS,T,2005,-1

Zubo,1,Adventure / Role-Playing (RPG) / Strategy,75,0.05,14.95,DS,E,2008,15

**Notice that the file is sorted by game title.**

Each line contains 10 features. The following table provides more detailed information about each of these features.

|  |  |  |  |
| --- | --- | --- | --- |
| *Table 1. Features of Games Data* | | | |
| Feature | Type | Description | Notes |
| 0 | string | Game Title |  |
| 1 | integer | Max number of players |  |
| 2 | string | Genre | Listing multiple, seperated by slashes |
| 3 | integer | Review Score |  |
| 4 | float | Sales | Measured in millions of dollars |
| 5 | float | Price | Average "used" price (in dollars) |
| 6 | string | Console |  |
| 7 | string | Rating |  |
| 8 | integer | Release Year |  |
| 9 | float | Play Length | Average time in hours to play through the game (-1 if play through time is not available) |

In this lab we will focus on features 0, 6, and 8 (plus 3, 4, and 9 if you do the optional challenge). In the next lab, which adds sorting, we will incorporate more functionality for the other features.

### Using search to explore video game data

You will implement a program called search-games.py, which will allow the user to search through this dataset and learn particular information.

Users may search by console and release year:

## **Games Search Example 1**

### Searching games by console and release year

User input is in bold.

$ **python3 search-games.py**

This program helps the user explore a data set of

over a thousand video games released between

2004 and 2008.

In particular, you can:

1. Search by console and release year

2. Search by game title

3. Quit

Choice: **1**

Enter year: **2005**

Enter console: **X360**

TITLE REVIEW SALES PRICE CONSOLE YEAR PLAYTIME

------------------------------ ------ ----- ----- ------- ---- --------

Amped 3 72 0.13 14.95 X360 2005 9.0

Battlefield 2: Modern Combat 77 0.36 16.95 X360 2005 6.9

Call of Duty 2 89 1.69 17.95 X360 2005 9.0

Condemned: Criminal Origins 81 0.33 14.95 X360 2005 7.5

Dead or Alive 4 85 0.38 17.95 X360 2005 6.0

GUN 75 0.22 19.95 X360 2005 8.0

Kameo: Elements of Power 79 0.31 11.95 X360 2005 10.2

Madden NFL 06 74 0.62 4.95 X360 2005 3.8

NBA 2K6 81 0.22 4.95 X360 2005 0.5

NBA LIVE 06 64 0.29 4.95 X360 2005 4.0

Need for Speed: Most Wanted 83 0.95 17.95 X360 2005 12.0

NHL 2K6 75 0.10 5.95 X360 2005 2.5

Perfect Dark Zero 79 0.70 17.95 X360 2005 8.0

Peter Jackson's King Kong: The 80 0.32 14.95 X360 2005 7.0

Quake 4 75 0.27 14.95 X360 2005 8.4

Ridge Racer 6 74 0.10 14.95 X360 2005 8.0

Tiger Woods PGA Tour 06 71 0.33 9.95 X360 2005 -1.0

Tony Hawk's American Wasteland 75 0.29 14.95 X360 2005 13.0

18 matches found

1. Search by console and release year

2. Search by game title

3. Quit

Choice: **1**

Enter year: **2006**

Enter console: **Wii**

TITLE REVIEW SALES PRICE CONSOLE YEAR PLAYTIME

------------------------------ ------ ----- ----- ------- ---- --------

Call of Duty 3 69 1.15 19.95 Wii 2006 8.4

Dragon Ball Z: Budokai Tenkaic 72 0.24 14.95 Wii 2006 14.6

Elebits 75 0.22 14.95 Wii 2006 9.0

Excite Truck 72 0.40 19.95 Wii 2006 14.0

Far Cry: Vengeance 38 0.07 16.95 Wii 2006 -1.0

Happy Feet 46 0.13 14.95 Wii 2006 0.7

Ice Age 2: The Meltdown 66 0.07 17.95 Wii 2006 8.5

Kororinpa: Marble Mania 69 0.08 17.95 Wii 2006 -1.0

Madden NFL 07 81 0.48 9.95 Wii 2006 -1.0

Marvel Ultimate Alliance 73 0.38 17.95 Wii 2006 14.0

Metal Slug Anthology 73 0.11 16.95 Wii 2006 7.0

Monster 4x4: World Circuit 51 0.82 12.95 Wii 2006 -1.0

Need for Speed: Carbon 67 0.47 17.95 Wii 2006 13.6

Open Season 49 0.07 17.95 Wii 2006 3.8

Pokmon Battle Revolution 53 0.81 39.95 Wii 2006 15.4

Rampage: Total Destruction 46 0.19 16.95 Wii 2006 5.0

Rayman Raving Rabbids 76 1.26 17.95 Wii 2006 6.2

Red Steel 63 0.58 14.95 Wii 2006 8.2

SpongeBob Squarepants: Creatur 57 0.44 14.95 Wii 2006 6.8

Super Monkey Ball: Banana Blit 74 1.00 14.95 Wii 2006 5.4

Super Swing Golf 72 0.14 16.95 Wii 2006 -1.0

The Ant Bully 45 0.09 16.95 Wii 2006 3.6

The Legend of Zelda: Twilight 95 3.33 39.95 Wii 2006 43.0

Tom Clancy's Splinter Cell: Do 61 0.11 17.95 Wii 2006 11.0

Trauma Center: Second Opinion 80 0.28 17.95 Wii 2006 9.0

WarioWare: Smooth Moves 83 0.90 39.95 Wii 2006 3.0

Wii Play 58 14.66 14.95 Wii 2006 4.0

27 matches found

1. Search by console and release year

2. Search by game title

3. Quit

Choice: **1**

Enter year: **2004**

Enter console: **Wii**

Sorry, no matches were found.

1. Search by console and release year

2. Search by game title

3. Quit

Choice: **3**

Goodbye!

## **Games Search Example 2**

### Searching games by title

User input is in bold.

$ **python3 search-games.py**

This program helps the user explore a data set of

over a thousand video games released between

2004 and 2008.

In particular, you can:

1. Search by console and release year

2. Search by game title

3. Quit

Choice: **2**

Enter game title (or start of title): **Sonic**

TITLE REVIEW SALES PRICE CONSOLE YEAR PLAYTIME

------------------------------ ------ ----- ----- ------- ---- --------

Sonic and the Secret Rings 69 1.08 16.95 Wii 2007 9.3

Sonic Chronicles: The Dark Bro 74 0.52 17.95 DS 2008 20.0

Sonic Riders: Zero Gravity 56 0.45 16.95 Wii 2008 4.3

Sonic Rivals 64 0.76 17.95 PSP 2006 6.5

Sonic Rivals 2 60 0.46 22.95 PSP 2007 22.0

Sonic Rush 82 1.20 14.95 DS 2005 6.2

Sonic Rush Adventure 78 0.52 24.95 DS 2007 9.0

Sonic the Hedgehog 46 0.41 14.95 X360 2007 3.1

Sonic the Hedgehog 43 0.40 16.95 PS3 2007 3.1

Sonic Unleashed 66 1.09 17.95 Wii 2008 13.0

Sonic Unleashed 54 0.42 17.95 PS3 2008 13.0

Sonic Unleashed 60 0.41 17.95 X360 2008 13.0

12 matches found

1. Search by console and release year

2. Search by game title

3. Quit

Choice: **2**

Enter game title (or start of title): **frogg**

TITLE REVIEW SALES PRICE CONSOLE YEAR PLAYTIME

------------------------------ ------ ----- ----- ------- ---- --------

Frogger: Helmet Chaos 66 0.11 19.95 PSP 2005 -1.0

Frogger: Helmet Chaos 69 0.06 17.95 DS 2005 -1.0

2 matches found

1. Search by console and release year

2. Search by game title

3. Quit

Choice: **2**

Enter game title (or start of title): **CS21**

Sorry, no matches were found.

1. Search by console and release year

2. Search by game title

3. Quit

Choice: **3**

Goodbye!

### Program Requirements

* All numeric data in the file should be type cast to the correct type when stored in the program.
* Compares data in lowercase format (so that user input is case insensitive) but displays results with their original case.
* Repeatedly presents a menu to the user with three choices: to search by game title, to search by console/year, or to quit.
* Ends when the user chooses to quit.
* Ensures that the user provides a valid menu choice and a valid year.
* Uses formatted printing to show search results in a table format that fits in an 80-character wide terminal window.
* **Uses linear search to find matches by console/year.**
* **Uses binary search to find matches by game title.**

### Tips

#### **1. Continue to use TDD**

We **strongly recommend** that you create a top-down design with a fully fleshed out main and stubs for all other functions. However, you may not work with a partner on this lab. Also you don’t need to get approval from an instuctor for your design, though we’d be glad to review it with you in lab, office hours, or ninja sessions. You should only begin implementing the other functions after you have a clear plan in place for the structure of main.

#### **2. Create modular, reusable functions**

Avoid duplicating similar code in multiple places throughout the program. Think about how you could re-use the same function. For example, we are doing two types of searchs (by year/console and by title), but the results of those searches are displayed in exactly the same way. Consider having a function that takes in a list of search results and displays them. Then you could call this function for either type of search.

#### **3. Displaying matches**

We want the results of our searches to be nicely displayed in the terminal window. However, some game titles names are quite long. We can use slicing to truncate these strings to a fixed maximum length, allowing us to create an easy to read table format. This [section](https://www.cs.swarthmore.edu/courses/CS21Book/ch07.html#string-slices) of the online textbook explains how to slice a string. For instance, the following example shows how to slice out the first 10 characters from a string:

>>> alphaString = "abcdefghijklmnopqrstuvwxyz"

>>> alphaString[0:10]

'abcdefghij'

Notice how this nicely works, even if we provide an upper bound that is longer than the string we’re trying to slice (unlike when dealing with lists):

>>> shortString = "xyz"

>>> shortString[0:10]

'xyz'

#### **4. Adapting linear search**

The version of linear search that we discussed in class stops as soon as it finds the item it is looking for, and returns the index of that item or -1 if not found. For this problem, your linear search will need to accumulate a list of matching results and return that.

In addition, the basic version of linear search assumes that you are searching a flat list. We will be searching a list of lists, so will need to adapt linear search to use double indexing to get at the data within each sublist.

#### **5. Adapting binary search**

Because the data is sorted by game title, we can use binary search to very quickly locate a particular game. However, game titles can be somewhat long or may include a subtitle. Additionally, the data may contain multiple copies of the same game, if it was released for different consoles under the same name.

For example, there were apparently 16 "LEGO" games released on 5 different consoles.

...

Legendary <- this has a different prefix

LEGO Batman: The Videogame

LEGO Batman: The Videogame

LEGO Batman: The Videogame

LEGO Batman: The Videogame

LEGO Batman: The Videogame

LEGO Indiana Jones: The Orig.

LEGO Indiana Jones: The Orig.

LEGO Indiana Jones: The Orig.

LEGO Indiana Jones: The Orig.

LEGO Star Wars II: The Orig.

LEGO Star Wars II: The Orig.

LEGO Star Wars II: The Orig.

**LEGO Star Wars: The Complete Saga** <- suppose binary search finds this

LEGO Star Wars: The Complete Saga

LEGO Star Wars: The Complete Saga

LEGO Star Wars: The Complete Saga

Lemmings <- this has a different prefix

...

##### **Finding the first match**

First write a binary search function that searches for a partial match to a game title and returns the index of the first match that it finds (or -1 otherwise). Make sure your binary search function uses startswith (see [here](https://docs.python.org/3/library/stdtypes.html#str.startswith)) rather than == so it doesn’t miss any partial matches (e.g. using "LEGO" to match on "LEGO Star Wars: The Complete Saga").

However, there might be more matches that are just before or just after that initial index you locate. So next you’ll need to search in the vicinity of that location to find all of the other potential matches.

##### **Finding nearby matches based on an initial seed**

We have conveniently provided you with a function to help with finding nearby matches based on an initial seed. This is called lookForwardBackward() and you can import it from the cs21s21 library. This function takes four parameters and returns a list of matching items that it found looking either forward or backwards. Those parameters are:

* A list with the data to search over
* An integer where the search should start from (this is the index that your "binary search" function will return)
* The partial match (string) that the function should be searching for (e.g. "LEGO" in the above example)
* A search direction. This is an integer which should have the value 1 (to search forward) or -1 (to search backwards)

You will need to import the function at the top of the program (from cs21s21 import lookForwardBackward) and call it twice for each binary search (once to get the forward matches and again to get the backward matches).

A call to the lookForwardBackward() function might look something like this:

backwardMatches = lookForwardBackward(currData, currIndex, title, -1)