**SI507 Project Document**

Tianhao Liang

**Project code**

Link to github repo: <https://github.com/liangth35/SI507project>

**README:**

**To run the code:**

If you want to rerun the whole crawling process: delete gamelist.csv and gameData.json, then run all the cells in Project\_retrieve\_data.ipynb. It will take roughly an hour to crawl and scrap 6544 games from steam.

Then, run Project.py in command line, a link will be generated (http://127.0.0.1:5001), the flask app will be available in that address. On the homepage, you can type in the range of price, release date, rating, and number of reviews of games you would like to search for, you can also choose to search for discounted games only. After the ranges have been set, click on "Search" button and the results will show up. On the result page, you can choose to sort the games by price, release date, rating, and number of reviews in both descending and ascending order. You can also select an attribute and plot its distribution by clicking on "Plot" button. When in the plot page, clicking "Return" button will take you back to results page.

**Required packages:**

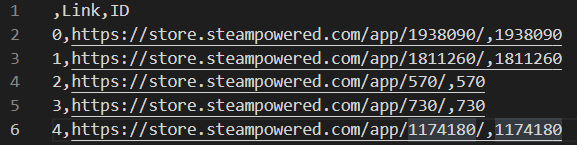
bs4, flask, requests, pandas, plotly

**Data sources**

First, I crawled the searching page on steam to obtain a list of games and the urls of their store pages. The url I used for crawling was [https://store.steampowered.com/search/?ignore\_preferences=1&category1=998&os=win&filter=globaltopsellers&page=%d](https://store.steampowered.com/search/?ignore_preferences=1&category1=998&os=win&filter=globaltopsellers&page=%25d), in which %d was replaced by the page numbers. There are about 25 games on each searching page. I scraped 404 pages in total and got 6654 valid game id–url pairs.

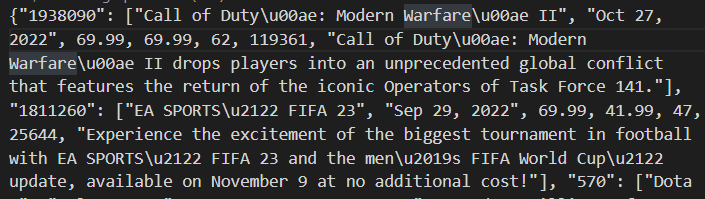
I obtained the html files of these pages using request.get, and I used beautifulsoup to extract the game ids and urls form the html files. I put the game ids and urls into a pandas dataframe. (The urls were used for crawling later and game ids were used as dictionary key in the next part.)

I implemented caching in this part by saving the dataframe as a csv file. When executed, if the program finds the csv file, it directly reads from the file instead of scraping the pages again. This is part of the cached csv file:



For the next step, I scraped the urls I obtained in the previous part, there were 6654 urls and I applied request.get to all of them and extracted the **name, release date, original price, discounted price, rating, review number, and description** of each game using beautifulsoup.select. Among those fields, release date, discounted price, reviews, and rating would be used as the for dimensions of the kd-tree, so users can sort the games by one of these fields. I saved the game data into a dictionary, with the game id being the key. The resulting dictionary has 6654 items.

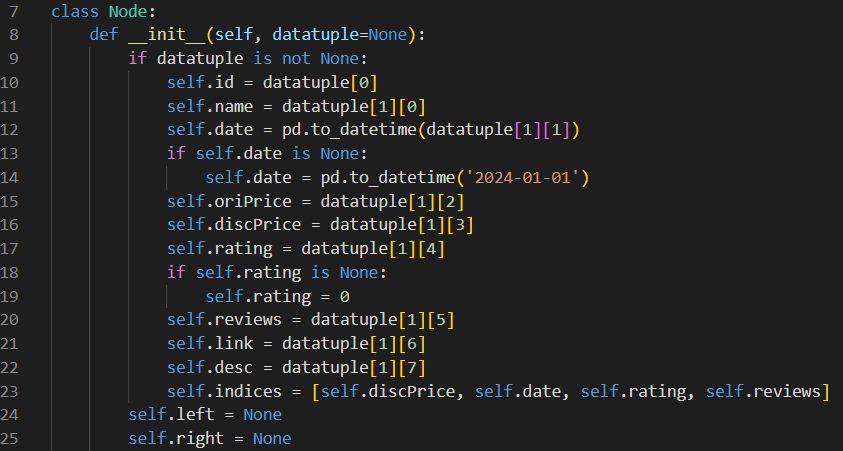
In this part, I also implemented caching. This time, I used a json file to cache the dictionary. This is part of my cached file:



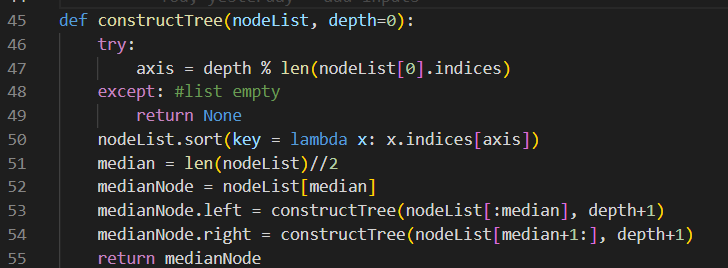
**Data Structure**

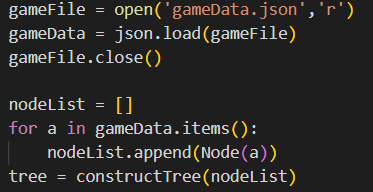
README:

The data structure I use in this project is a kd-tree. It is a variant of binary search tree, in which nodes are partitioned in 4 dimensions in turn as depth grows. I organize the game data into classes named Node. Other than the game data, each node has left and right child, which are set to None by default. I use 4 numeric attributes in game data: release date, original price, discounted price, and rating, as the 4 dimensions of this kd-tree.



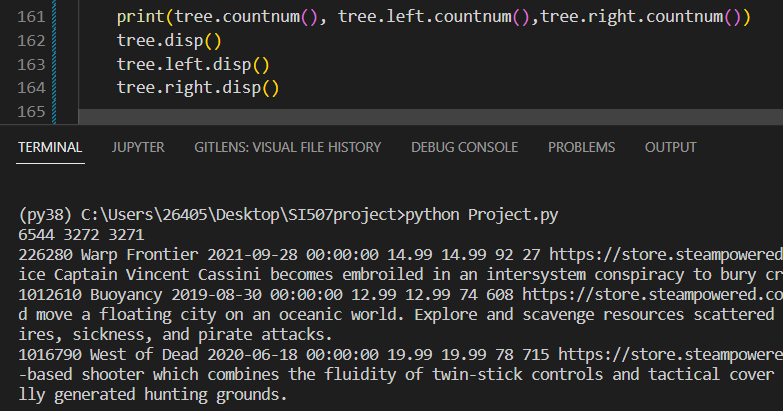
When constructing the tree, the current dimension is calculated from the depth first. Then, the nodelist is sorted in that dimension and partitioned into two halves. These two halves of nodes are then fed to two recursive calls of the kd-Tree function and the returned results become the left and right child node. At last, the root node of the tree is returned.





Because I use classes to construct my tree, the resulting tree is not json serializable. To solve this problem, I put both the tree-constructing code and the flask code in Project.py so that the tree can be directly passed to the flask app.

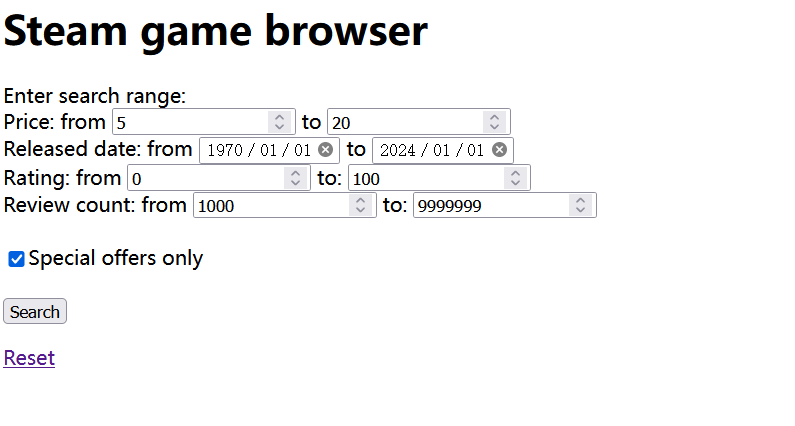
To demonstrate my tree structure, I print out part of it: in the screenshot, the number of nodes in the tree is 6544, the same as the number of games I crawled, the number of nodes in its left and right subtree are 3272 and 3271 respectively, summing up to 6543. In the kd-tree, discPrice is the first dimension. From the printed content of the three nodes, we can observe that the left child has smaller discPrice(12.99) than the root node(14.99), while the right child has greater discPrice(19.99) than the root node, meaning the kd-tree is valid.



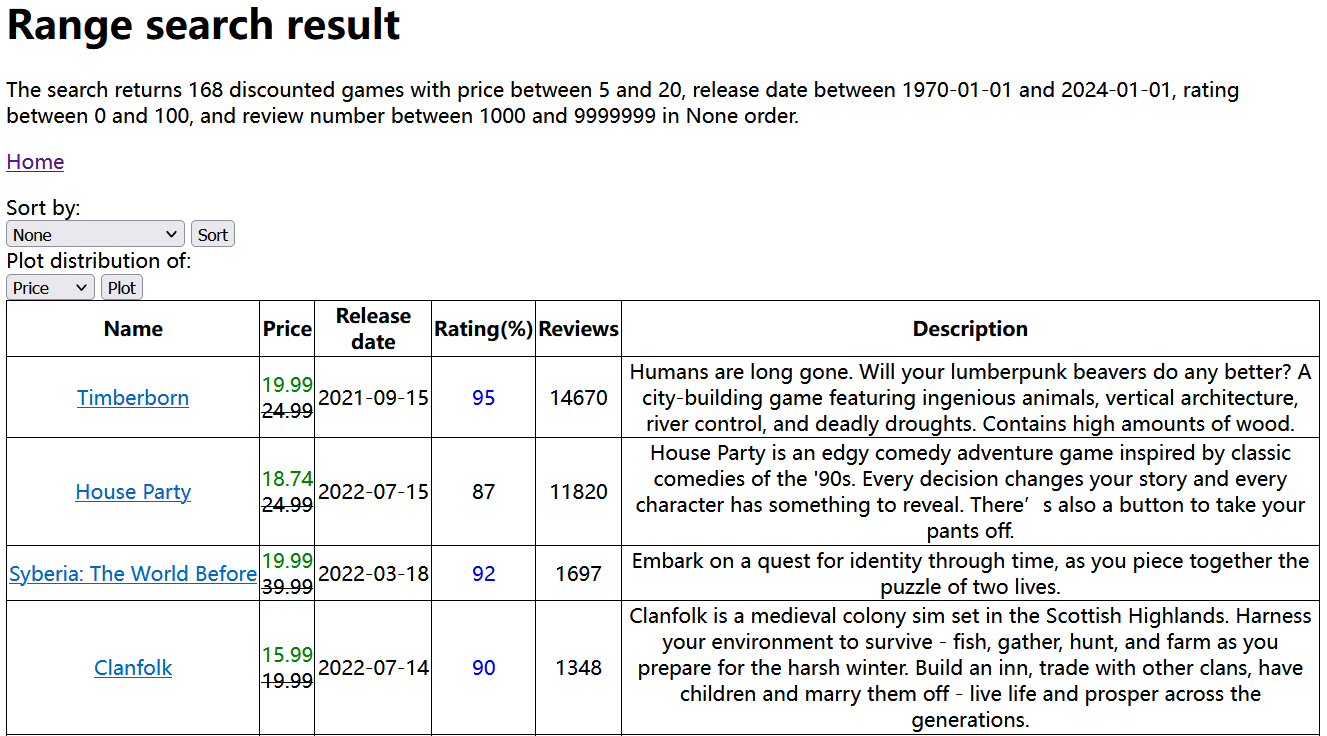
**Interaction and Presentation Options**

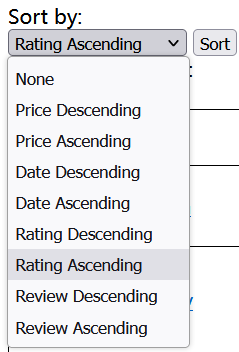
**Summary:** In my project, the user can search for games with specific range of price, release date, rating, and number of reviews. They can also filter out the discounted games, sort the results by price, release date, rating, and number of reviews in either ascending and descending order, and plot the distributions of these four attributes.

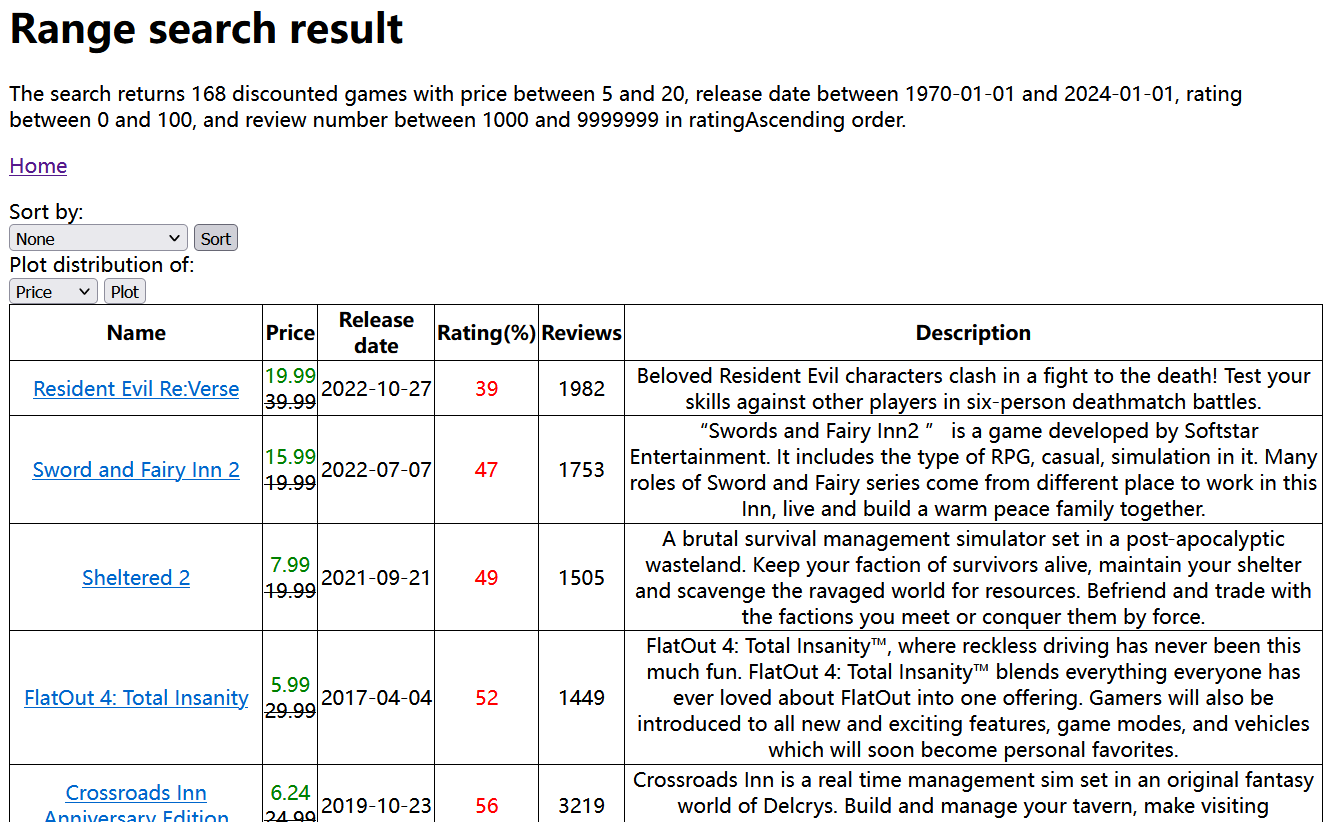
Flask is used to build the interaction interface of this project. On the homepage, the user can type in the range of price, release date, rating, and number of reviews of the games he would like to search for. He can also choose to search for discounted games by clicking on "Special offers only".

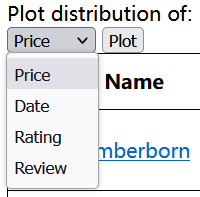
****

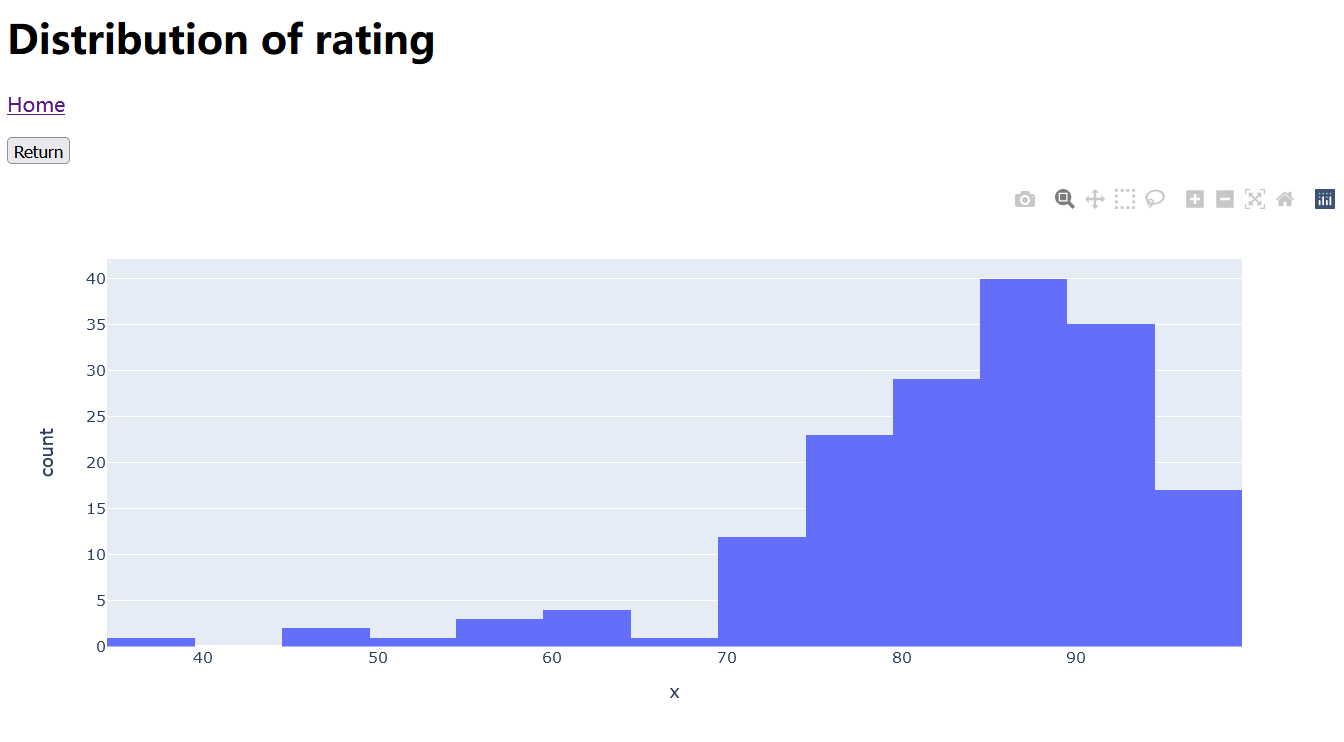
After entering the range and clicking on "Search", the program will conduct a range search on the kd-tree and the following result page will appear. The name, price, release date, rating, review number, and description of the games in the range specified by the user will be shown in the table. The names are actually links to the steam store pages of the games. If the games are discounted, the prices will become green and the original prices will be crossed out. The rating will be blue if >=90 and will be red if <60. There is a media query that hides the description if the screen width is less than 1080px.



By selecting the sort type and clicking on "Sort", the results will be sorted by the specified key in either ascending or descending order. For example, in the screenshot, the results are sorted by rating in ascending order.



The user can also plot the distribution of price, release date, rating, and reviews of the resulting games by selecting the variable and clicking on "Plot". This is the screenshot of distribution of rating, it is plotted using plotly. Clicking on "Return" will allow the user to return to the result page.



**Link to demo video:**

<https://drive.google.com/file/d/1WqeOt_qBJmEy1C2ld7dkJ55vplLwGyf-/view?usp=sharing>