**Beer Guidebook**

For a Belgian citizen it is almost a part of the DNA to drink Belgian beer. Because the range of national beers is very large, this piece of work takes the opportunity to set up an application that covers the idea of a personal beer guidebook. Every time a user enjoys a beer, the guidebook should be expanded by simply inserting new information into the application (which is connected to a csv file). A new entry consists of a key (name and alcohol level of the beer) and a value section. The value section contains the following information: Feature/Feeling of the taste, region of origin, city of origin and rating score. Beside inserting new entries, in starting the app the user can also choose to delete, sort or search for existing entries.

The main data structure (the guidebook) uses a (python) dictionary (Python Software Foundation, 09/21). The key is characterized by an immutable tuple (name and alcohol level) whereas the value section uses a mutable list of three strings (feature, region, city) and one floating point number (score). The choice of this set-up can be explained with regard to three dimensions:

1. Running Time:

Because of the built-in hash function that Python dictionaries uses, this data structure should lead to constant running time (i.e. does not directly depend on the number of data points) for insertion, deletion and searching. Furthermore, sorting can also be done in less than quadradic time, using for example Timsort (MIT OpenCourseWare, 2013).

1. Memory Space:

Memory space can be an issue if there are relatively few entries compared to the size of the dictionary (i.e. number of buckets). But as the number of keys get close to the number of buckets, the space issue in using a dictionary becomes less significant while comparing it to other data structures such as an array, a linked lists or a binary search tree. The appendix contains (A1: Overview: Data Structures and Memory Space) an overview of the main advantages and disadvantages with respect to memory allocation for the different data structures (Brookshear & Brylow, 2018) (CS50, 2020) (PyCon 2017, 2017).

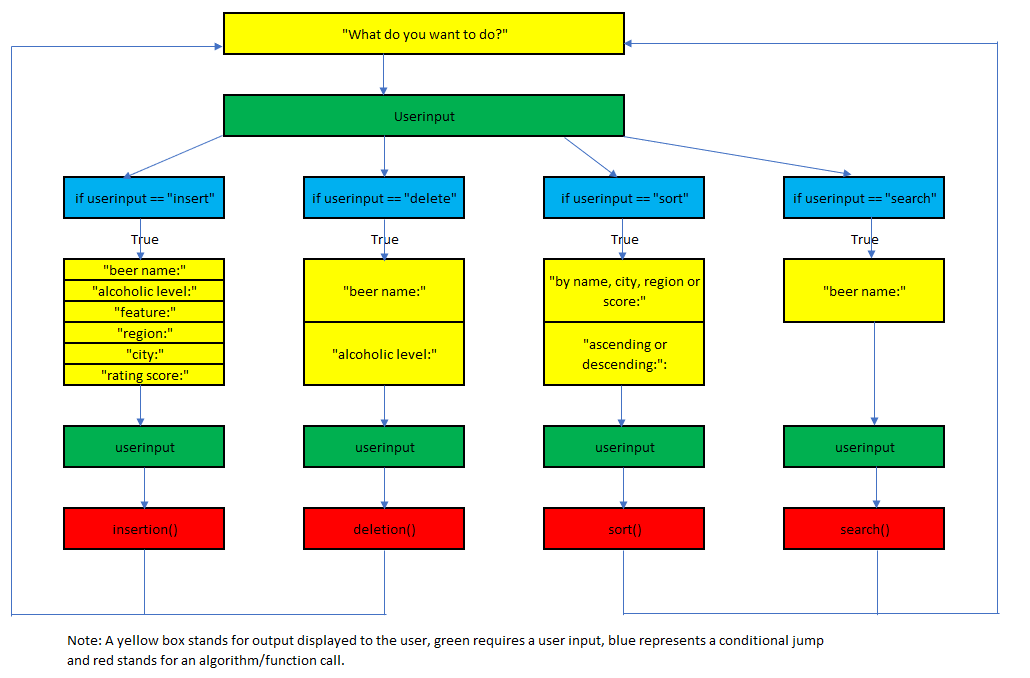
1. Developer Time:

The Python dictionary is a built-in data structure and is very user friendly. It has built-in functions for deletion (“pop” or “del”), searching (“==” or “in”) and sorting (“sorted”) (Python Software Foundation, 09/21).

Taking these three criteria into account, the decision in using a python dictionary data structure is mainly related to the great advantages in running and developer time whereas space issues only seem to be temporarily (when resizing is required).

Looking to the application as a whole, Figure 1 depicts the structure quite sufficiently.

**Figure 1: Main structure of the beer guidebook algorithm**



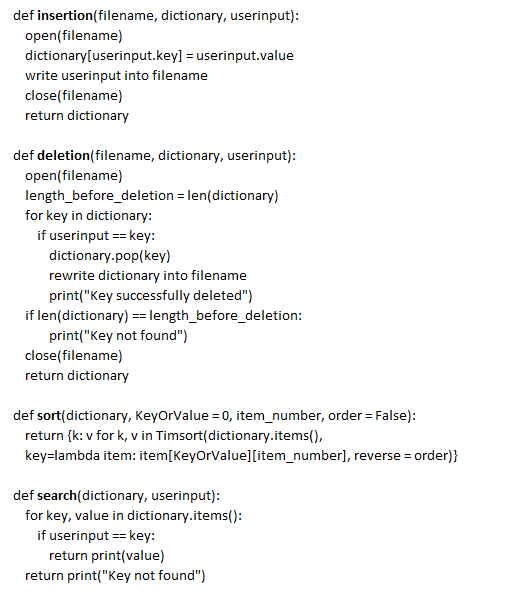
The user will first be asked to enter what he wants to do: insert, delete, search or sort? After providing the desired action by the user, the application goes into a conditional jump:

* “insert” requires that the user provides information about the name and alcohol level, taste features, region, city and a personal rating score.
* “delete” only asks the name and the alcohol level of the beer brand.
* “search” behaves almost similarly in asking the user to provide a beer name.
* “sort” sends additional queries for 1) by name, region, city or rating 2) ascending or descending.

Following these additional information, the application can then call one of the functions: insertion, deletion, sort, search. In general, the whole process from the beginning to the end can be repeatedly done (see the loop arrows at the end which go back to the beginning).

Looking to the pseudocodes of each function (Figure 2), one can distinguish between two groups. Insertion and deletion work in a similar way in the sense that a csv file will be opened/loaded. After opening the file, in calling insertion or deletion the dictionary will be resized and therefore the csv file will be expanded/overwritten. On the other side, sort and search only work with an existing dictionary without changing the data structure itself.

**Figure 2: Pseudocode for insertion, deletion, sort and search algorithms**



The final step then still consists of testing the reliability of the program. This part is divided into a built-in analysis and a final test data check. The built-in analysis uses small checkpoints while building the application (in mostly running the “print” function). Checkpoints will be included to test for the following questions:

* Are there differences in using capital or small letters when asking the user for input (Boolean test: Are the inputs equal?).
* After every successful deletion, became the length of the dictionary smaller by one unit?
* Will multiple insertion be correctly applied?

The final check then still consists of testing multiple dictionaries which already have existing data (using a csv file where some datasets are artificially generated and others are created by the developer itself). The application can then be tested with respect to 6 checkpoints in running a program “testing.py” ((1) Test for immutability of key, (2) - (5) test for correct use of inserting, deletion, searching, sorting, (6) test for spelling errors). The result of testing.py should lead to 100 % of correct answers.

Appendix

A1) Overview: Data Structures and Memory Space

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Linked List |  | Pro: | Resizing flexible |  |  |  | |
|  |  | Con: | Every entry requires space | |  |  | |
|  |  |  | for value and space for next pointer | |  |  | |
| Array |  | Pro: | Every entry requires only space | |  |  | |
|  |  |  | for value (no pointer necessary because contiguous structure) | | | |
|  |  | Con: | Resizing inflexible because of contiguous structure | | |  | |
| Binary Search tree | | Pro: | Resizing flexible |  |  |  | |
|  |  | Con: | Every entry requires space | |  |  | |
|  |  |  | for value and space for two next pointers | | |  | |
|  |  |  | (left and right child) |  |  |  | |
| Dictionary | | Pro: | Every entry requires only space | |  |  | |
|  |  |  | for value (no pointer included) | | | |
|  |  | Con: | Resizing expensive because a set of buckets will | | |  | |
|  |  |  | be allocated. The larger the number of unused buckets, the higher the cost. | | | |

Reference List:

Brookshear, J. & Brylow, D. (2018) Computer Science: An Overview. 13th ed. London: Pearson Education Limited.

CS50 2020 (2020) Lecture 5 – Data Structures. Available from: https://www.youtube.com/watch?v=2T-A\_GFuoTo&t=6012s [Accessed 10 December 2021]

MIT OpenCourseWare (2013) Hashing with Chaining. Available from: https://www.youtube.com/watch?v=0M\_kIqhwbFo [Accessed 07 December 2021]

Python Software Foundation (Latest update 09/21) 3.6.15 Documentation – 4.10. Mapping Types-dict. Available from: https://docs.python.org/3.6/library/stdtypes.html#typesmapping [Accessed 12 December 2021]

PyCon 2017 (2017) Raymond Hettinger Modern Python Dictionaries A confluence of a dozen great ideas PyCon 2017. Available from: https://www.youtube.com/watch?v=npw4s1QTmPg&t=1s [Accessed 07 December 2021]