U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

# Book Scanning

Artificial Intelligence
2020/2021
3MIEIC02

Inês Silva, up201806385@fe.up.pt
Mariana Truta, up201806543@fe.up.pt
Rita Peixoto, up201806257@fe.up.pt

# Project Specification

| Input file | Description |
|---|---|
| 6 2 7 | There are 6 books, 2 libraries, and 7 days for scanning. |
| 1 2 3 6 5 4 | The scores of the books are 1, 2, 3, 6, 5, 4 (in order). |
| 5 2 2 | Library 0 has 5 books, the signup process takes 2 days, and the library can ship 2 books per day. |
| 0 1 2 3 4 | The books in library 0 are: book 0, book 1, book 2, book 3, and book 4. |
| 4 3 1 | Library 1 has 4 books, the signup process takes 3 days, and the library can ship 1 book per day. |
| 3 2 5 0 | The books in library 1 are: book 3, book 2, book 5 and book 0. |

Figure 1. Example of a input data set

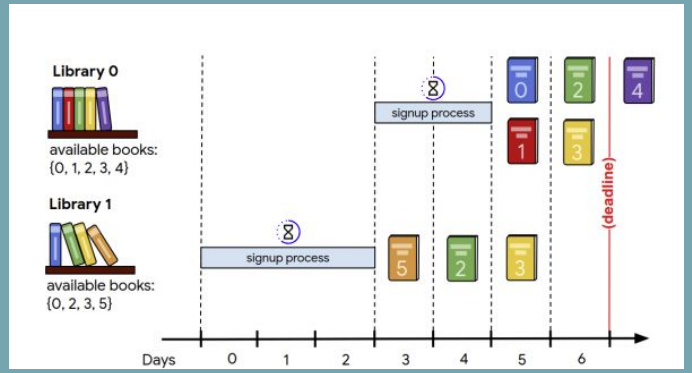


Figure 2. Based on the input data set of Figure 1, books 0, 1, 2, 3, and 5 are scanned by Day 6, and the total score is 16.

This project consists of solving an optimization problem: book scanning.
Given a description of a set of libraries (L) and books available (B), there is the need to plan which books to scan from which library to maximize the total score of all scanned books, taking into account that each library needs to be signed up before it can ship books and all books must be scanned within D days.

Read all the information about this problem in
https://storage.googleapis.com/coding-competitions.appspot.com/HC/2020/hashcode_2020_online_qualification_round.pdf

# References

As this project consists of a hashcode qualification problem, the research made lead us to articles about the implemented solutions by some of the teams:

- https://medium.com/@kevinjohn1999/how-we-achieved-a-98-27-of-the-best-score-at-google-hashcode-2020-94314a904190
- https://towardsdatascience.com/google-hash-code-2020-a-greedy-approach-2dd4587b6033

Related topics:

- https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/
- https://www.sciencedirect.com/topics/computer-science/local-search-algorithm

# Formulation of the optimization problem

### Solution Representation

The solution describes which books to ship from which library and the order in which libraries are signed up. In the output files, the first line contains the number of libraries to sign up and the following lines describe the libraries in the order to be signed up in two lines each - the first line containing the library id and the number of books sent, the second line containing the IDs of the books to scan from the library.

### Rigid Constraints

- Many libraries can have a copy of the same book, but we only need to scan each book once;
- If the same book is shipped from multiple libraries, the solution will be accepted but the score for the book will be awarded only once;
- Only one library at a time can be going through the signing process;
- Each library must be described only once;
- You don't need to scan all books from a library you describe;
- If a library signup process finishes after D days, its description will be ignored;
- Books shipped after D days will be ignored;
- the signup process for a library can start only when no other signup processes are running;
- The libraries can be signed up in any order.

### Evaluation Function

```python
def calculate_total_score(books_dict, scores):
    all_books = set()
    for id in books_dict:
        all_books.update(books_dict[id])
    return score(list(all_books), scores)
```

The evaluation function iterates through the library books saving them on a set (to eliminate the duplicate books) and then add the score of all those books producing the library score.

### Neighborhood Function

```python
def find_best_neighbour(choosen_libraries, choosen_books, libraries, scores, n_days):
    best_score = calculate_total_score(choosen_books, scores)
    best_libraries = copy.deepcopy(choosen_libraries)
    best_books = copy.deepcopy(choosen_books)
    found_better = False

    for current_lib in choosen_libraries:
        day = 0
        new_list = []
        scanned_books_dict = dict()
        scanned_books_set = set()
        all_libraries = copy.deepcopy(libraries)
        for lib in choosen_libraries:
            if lib == current_lib:
                all_libraries.remove(lib)
                break
            else:
                new_list.append(lib)
                scanned_books_dict[lib.id] = lib.get_books(n_days-day)
                scanned_books_set.update(scanned_books_dict[lib.id])
                all_libraries.remove(lib)
                day += lib.signup_days


        while day < n_days and len(all_libraries) > 0:
            id = choose_best_score(n_days - day, all_libraries, scores, scanned_books_set)
            if id == -1:
                break
            for lib in all_libraries:
                if lib.id == id:
                    scanned_books_dict[lib.id] = lib.get_books(n_days-day)
                    scanned_books_set.update(scanned_books_dict[lib.id])
                    new_list.append(lib)
                    day += lib.signup_days
                    all_libraries.remove(lib)

        new_score = calculate_total_score(scanned_books_dict, scores)
        if new_score > best_score:
            found_better = True
            best_libraries = copy.deepcopy(new_list)
            best_books = copy.deepcopy(scanned_books_dict)
            best_score = new_score
            print("better!")
        else:
            print("not better :(")

    return found_better, best_libraries, best_books, best_score
```

The find_first_neighbour function is the same as the find_best_neighbour, simply returning the first better neighbour it encounters.

```python
if new_score > best_score:
    found_better = True
    best_libraries = copy.deepcopy(new_list)
    best_books = copy.deepcopy(scanned_books_dict)
    best_score = new_score
    print("better!")
    return found_better, best_libraries, best_books, best_score
```

# Progress so far ...

The project is being developed in Python 3, with a text interface.
The development environment is Visual Studio Code.
For the most part of the project, the data structures used are arrays, sets, lists and classes.
The structure of the files is:

- src: where the source code is located;
- input: where all the libraries data is stored;
- output: where the output files of the program are stored;
- docs: where the documents are kept, such as this document.

At this point, it was implemented a greedy algorithm to find the solution.
The program has to be run with a single argument, specifying the input file where the libraries data is stored (for example, running the command *python bookscanning.py a_example.txt*).

So far, the best results for each file are:

| **File name** | **Total score** |
|---|---|
| a_example.txt | 21 |
| b_read_on.txt | 5 822 900 |
| c_incunabula.txt | 5 689 822 |
| d_tough_choices.txt | 5 028 530 |
| e_so_many_books.txt | 4 212 457 |
| f_libraries_of_the_world.txt | 5 283 965 |
| Total | 26 037 695 |