Algorithmique et structure de données

Binary search trees

In this lab, we are going to use binary search trees (BST) to create an index over an unsorted data collection. The data collection will act as a table in a database. It represents fictional characters that appear in English novels, including their name, surname, title of book they appear in, and their age. We consider that each name is unique. This "database table" will be stored as an array of fictional characters.

In such a table, looking up a person by name will be costly: we would have to look over all the array values (possibly all). The worse-case complexity is therefore O(n) and this is called a table scan.

When performing a query in a database, we usually do not query the data themselves directly. Instead, we query an *index*. The index is a (mostly) in-memory data structure that sorts records keys. Contrary to the database table, the index doesn't contain all information. Instead, each key in the index is associated with the address of the corresponding data in the actual unsorted collection. Once we find the key in the index (O(log(h) for a binary search tree), we know where to find the value in the database table (array in this case: the lookup would be O(1)). The query is therefore down to a O(1)+O(log(h)) pipeline, which is expected to be less then O(n). This speed-up comes to the cost of occupying more memory space (the size of the index, i.e. a O(n*(sizeof(key)+sizeof(int))) memory space).

The following figure illustrate what we want to achieve:

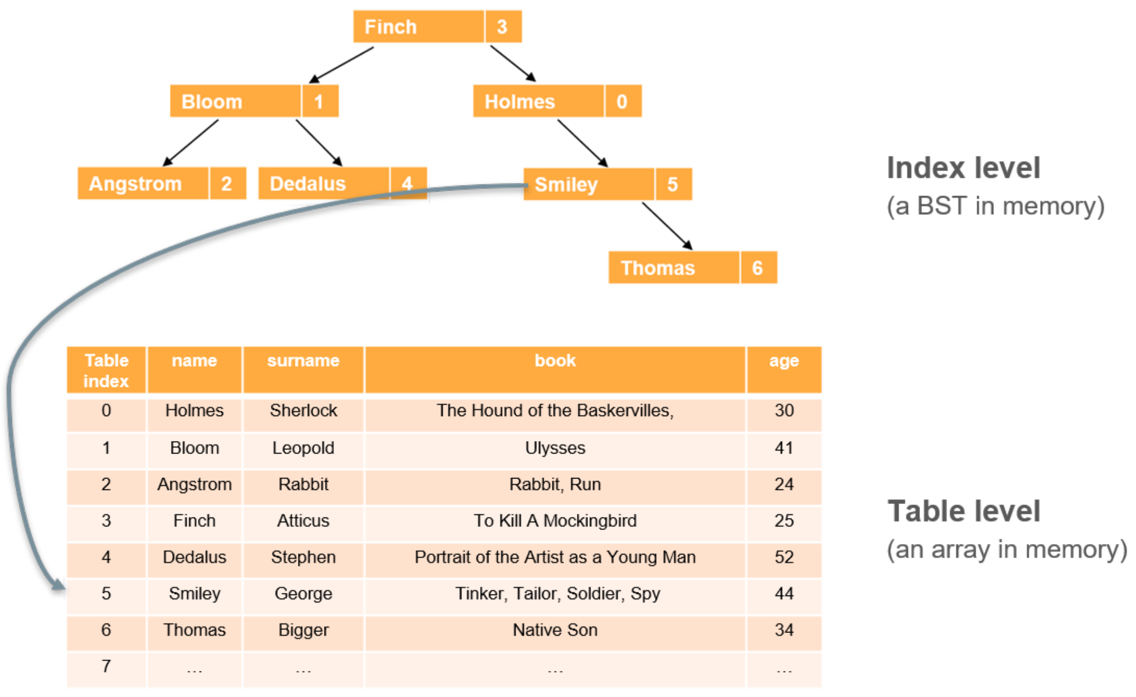

Table index	name	surname	book	age
0	Holmes	Sherlock	The Hound of the Baskervilles,	30
1	Bloom	Leopold	Ulysses	41
2	Angstrom	Rabbit	Rabbit, Run	24
3	Finch	Atticus	To Kill A Mockingbird	25
4	Dedalus	Stephen	Portrait of the Artist as a Young Man	52
5	Smiley	George	Tinker, Tailor, Soldier, Spy	44
6	Thomas	Bigger	Native Son	34
7	…	…	…	…

1. Create the table

1.1 Create the FictionalCharacter struct

Create a C/C++ **FictionalCharacter** structure (*struct)*, that will hold the information about a character.

(File(s): FictionalCharacter.h)

1.2 Create the table in memory

Create a function called **load(…)** that takes the absolute path to a semi-column separated file as an input. It contains data to use for initializing the database table (a simple array of FictionalCharacter). The function will parse that file and return a new array of FictionalCharacter. It is desirable that the array is a dynamic array.

The complete CSV file you can use is located at :

https://gist.github.com/cgravier/cbfa37637781671cc7b183d627259d61

It is reproduced in appendix for your convenience.

(File(s): FictionalCharacter.cpp)

2. Create the index

2.1 Create the BST struct

Create a binary search tree (BST) structure called **Node**. It contains a field called **key** of type **char*** (or **string**) and a field called **index** of type **int**.

(File(s): BST.cpp)

2.2 Create the BST insertion algorithm

Create a function called **insert(…)**, that allows to insert the given key and index into the BST as a new node. It returns **true** if the node was added, **false** otherwise.

(File(s): BST.cpp)

2.3 Create control function

Create a function to check either a given tree given by its root node is a BST or not (return **true** | **false**).

(File(s): BST.cpp)

2.4 Create a getHeight() function

Create a function that computes the height of a BST given its root node (a pointer to a Node).

(File(s): BST.cpp)

2.5 Build the index

Create a function called **index(…)**, that takes a database table as created in 1.2, and build the associated index.

3. Let’s play

Create a new pair of C/C++ files (.h and .cpp) called Database(.h, .cpp).
You will implement the following.

3.1 Create database

Create a function called **createDatabase()**, that takes as an input :

- The absolute path to a semi column separated file containing fictional characters infos
- An empty array of FictionalCharacters (that we will fill)
- An pointer to a Node that is initialized to NULL (that will serve as the pointer to the root of the index that we will be creating).

This function calls **load(…)** from FictionalCharacters.cpp to create the database table. Then, iterating on each value of the array, it inserts a (key, value) pair into the index by calling the **insert()** function of from BST.cpp. The two last arguments of our function will therefore be initialized after this call.

3.2 Wrap it up

Create a main.cpp file. In the main function you will :

- Call the createDababase function
- Once the database table and the index are built, call the **getHeight()** function on the index you created.

Can you explain your result and its consequences ?

3.3 Adapt, improve, overcome

Devise a way to prevent the issue we identified in 3.2 and code it.

4. Query

In this section, we will be interested in building queries over the database of FictionalCharacters we just built.

4.1 Lookup

Create a function in the BST.(h, cpp) files in order to be able to lookup a character given its name. The function will first query the BST to get the character index, and then fetch the fictional character information from the database table using the index associated to its key.

You can afterwards call this function in your main method:

- with a name that exists in the database
- with a name that doesn't belong to the database

4.2 Range queries

We will do the same thing for range queries. A range query is a query that lookup records that are in a given range. In this case, we will add a function in the BST that is able to lookup for all indexes of records whose keys are between two strings.

For instance, calling rangeQuery("Bo", "Bsu") will send back the index of the records of "`Bolling`", "`Brodie`"; "`Brother`".

After having created that range query functionality and checked it is correct by calling it in your main function, you code the answer to the following query :

- ⇨ What is the average age of the characters whose names are in the "ja" <-> "pi" range?

Appendix

A. 50 fictional character infos as a CSV file.

```
Bond;James;Casino Royale;37
di Rondo;Cosimo Piovasco;The Baron in the Trees;46
Lolita;Lolita;Vladimir Nabokov;36
the Pooh;Winnie;Winnie the Pooh;39
Adams;Nick;In Our Time;36
Angstrom;Rabbit;Rabbit Run;32
Barnes;Jake;The Sun Also Rises;29
Bart;Lily;The House of Mirth;41
Bloom;Leopold;Ulysses;45
Bloom;Molly;Ulysses;55
Bolling;Binx;The Moviegoer;54
Brodie;Jean;The Prime of Miss Jean Brodie;49
Brother;Big;1984;43
Buendia;Aureliano;One Hundred Years of Solitude;30
Caulfield;Holden;The Catcher in the Rye;33
Caulfield;Phoebe;The Catcher in the Rye;34
Crawford;Janie;Their Eyes Were Watching God;30
Dalloway;Clarissa;Mrs. Dalloway;38
Dedalus;Stephen;Portrait of the Artist as a Young Man;51
Finch;Atticus;To Kill A Mockingbird;41
Flyte;Sebastian;Brideshead Revisited;27
Gatsby;Jay;The Great Gatsby;61
Glass;Seymour;Nine Stories;38
Golightly;Holly;Breakfast at Tiffany's;36
Henderson;Eugene;Henderson the Rain King;62
Holden;Judge;Blood Meridian Cormac McCarthy;56
Holmes;Sherlock;The Hound of the Baskervilles;47
Humbert;Humbert;Lolita;58
Jimson;Gulley;The Horse's Mouth;46
Marcel;Proust;Remembrance of the past;34
Marlowe;Philip;The Big Sleep;57
Maturin;Stephen;Master and Commander;30
Matzerath;Oskar;The Tin Drum;52
McCrae;Augustus;Lonesome Dove;56
Merrill;Neddy;The Swimmer;39
Moriarty;Dean;On the Road;41
Motes;Hazel;Wise Blood;32
O'Hara;Scarlett;Gone With the Wind Margaret Mitchell;29
Pan;Peter;The Little White Bird;51
Peace;Sula;Sula;60
Portnoy;Alex;Portnoy's Complaint;48
Reilly;Ignatius;A Confederacy of Dunces;38
Ripley;Tom;The Talented Mr. Ripley;55
Samsa;Gregor;The Metamorphosis;33
Shimerda;Antonia;My Antonia;37
Smiley;George;Tinker;28
Spade;Sam;The Maltese Falcon;27
Stark;Willie;All the King's Men;30
Thomas;Bigger;Native Son;27
```