Introduction to Computer Science for Engineers

Fibonacci Numbers ✔✔✔ ☺ 🎓

This assignment **closed** November 10, 2024 at 23:15.

Fibonacci Series

A special series of numbers are the Fibonacci numbers. They can be defined recursively by the following algorithm:

```
1  # Algorithm:  fib(n):
2  # Input:      a natural number n
3  # Output:     the n-th Fibonacci number
4
5      if n = 0 or n = 1
6          then 1
7          else fib(n-2) + fib(n-1)
8      fi
```

Assignments

1. Write a recursive Python function `fib1(n: int) -> int` to calculate the nth Fibonacci number. Use the script to print out the first 15 Fibonacci numbers.

2. Show how the number of function calls increases using the example `fib1(5)`. Explain what *tree recursion* is. Provide your solution using the comment block at the top of the template file.

3. Now, rewrite your function `fib1` into an iterative solution (using a loop instead of recursion) and name it `fib2(n: int) -> int`

   Show using the same example as above that `fib2` is more efficient than `fib1`.

   Hint: Store the values for `fib(n-2)` and `fib(n-1)` in two separate variables.

4. Use the script to calculate the 23rd Fibonacci number using both functions and count the number of function calls (using the global variable `number_recursive_calls` for `fib1`) and the number of loop iterations for `fib2`.

Template files

Get all files in an archive templates.zip or templates.tgz.

`fibonacci.py`

Miit Dholakia | | |

Introduction to Computer Science for Engineers

Searching Books

This assignment **closed** November 10, 2024 at 23:15.

Searching Books

Your friend is coming over to get back the book “Clean Code” by Robert C. Martin that they lent you. Unfortunately, you misplaced the book somewhere in your bookshelf. Fortunately, you have a digital record of all books and placed them ordered by name in your bookshelf. Since we can use `Computer Science` to solve the problem in two hours to not do something in real life for two minutes, you decide to write an algorithm to search for the book and return its position in the list of books.

Assignments

1. Write the function `search_linear(a: list[str], item: str) -> int | None` to find the position of `item` in the array `a` with linear search. If the item does not exist, return `None`.

   Hint: Use Pythons default string compare method to compare two strings. E.g.: `"a" == "aa"` and `"a" < "aa"`.

2. Since you read a lot of books, finding items in the list via linear search takes too much time. Implement the function `search_binary(a: list[str], item: str) -> int | None` to find the position of `item` in the array `a` with binary search. If the item does not exist, return `None`. The list will be sorted.

3. You realize that the friend is coming over tomorrow and that the problem is now a problem of future you. Still exited from implementing the two algorithms, you decide to compare both of them.

   Copy the source code of both functions you just wrote over to `search_linear_cmp_count(a: list[str], item: str) -> int` and `search_binary_cmp_count(a: list[str], item: str) -> int` respectively. Instead of returning the position of the element, count the number of comparisons needed to find the item. If the item is not in the list, return the number of comparisons needed by the algorithm.

   Hint: Both `<` and `==` count as one comparison.

   Hint 2: This task will not be tested automatically, only `1.` and `2.`.

4. Print some examples and results in the script.

Template files

Get all files in an archive templates.zip or templates.tgz.

`searching_books.py`

Miit Dholakia |

Ternary Search

This assignment **closed** November 10, 2024 at 23:15.

As we know from the lecture, binary search divides the search space by two for every recursive function call. Let's take it up a notch. Instead of having one split point and thus reducing the number of elements to search by half, let us introduce two split points and thus reduce the number of elements to search to a third. It has to be faster, right?

Assignments

1. Write the function `binary_search(a: list[int], item: int) -> int | None` to find the position of `item` in the array `a` with binary search. If the item does not exist, return `None`.

   Hint: Use the helper function for the implementation.

2. Write the function `ternary_search(a: list[int], item: int) -> int | None` to find the position of `item` in the array `a` with ternary search. If the item does not exist, return `None`.

   Ternary search uses two "middle" points to split the given list into three parts. It then checks in which part the item must be and recursively calls itself, until it finds the position of the item. If the item is in one of the split points, the algorithm can return early, since this is the position of the item.

   Set the two "middle" points to one third and two-thirds of the list respectively.

In the `Searching Books` assignment, we used the number of comparisons to compare different search algorithms. In this task, we will use the absolute runtime. The template has two functions, one to measure the time for binary search (`time_binary_search`) and one for ternary search (`time_ternary_search`). The script itself generates a sorted array `ARRAY` to search some random values `SEARCH_VALUES`.

3. Compare the two algorithms. Play a bit with the numbers, e.g. increase or decrease the number of elements. Is ternary search really faster than binary search?

Hint: All provided arrays for this task are sorted.

Note: Comparing the time it takes different algorithms to run has it flaws. Your system could do some other stuff in the background and thus interfere with the run-time, or some data could be cached due to previous runs. This simple approach will be enough for this task for now.

Template files

Get all files in an archive templates.zip or templates.tgz.

`ternary_search.py`

Miit Dholakia |