Laboratory Paper

Barbu David Constantin
Grupa 1, Anul I, Secțiunea Calculatoare
(engleza)

Contents

1 Problem statement

The purpose of this lab work is to solve the following problem: a fisherman must choose from a set of lobsters those with the maximum sum of their values, but with a sum of dimensions smaller than the capacity of his net. The problem differs from the classic knapsack problem in that the chosen lobsters must be enumerated.

Specifically, it is considered that the fisherman has a net with a given maximum capacity. He has at his disposal an undetermined number of lobsters, each associated with three essential characteristics: name, size, and value. The program should receive as input the maximum capacity of the net and detailed information about each lobster. The objective is to determine the optimal combination of lobsters that maximizes the total value, while respecting the capacity constraint of the net.

The formulation of the problem can be presented mathematically as follows:

- **Input**:
  - C - the maximum capacity of the net (a positive constant).
  - n - the number of available lobsters.
  - For each lobster i ($i = 1, 2, \dots, n$):
    - name_i - the name of lobster i (a string of characters).
    - size_i - the size of lobster i (a positive real number).
    - value_i - the value of lobster i (a positive real number).
- **Output**:
  - A subset of selected lobsters such that:
    - The sum of the sizes of the selected lobsters is less than or equal to C.
    - The sum of the values of the selected lobsters is maximum.
    - The selected lobsters are enumerated.

Thus, the problem falls into the class of combinatorial optimization problems, being a variation of the knapsack problem, but with an additional requirement to enumerate the selected lobsters. Implementing an efficient algorithm for this problem will need to consider combinatorial complexity, ensuring that the obtained solution is optimal given the constraints provided.

2 Algorithms

2.1 Algoritm rucsac

In order to get the maximum possible value that can fit in the net I used the knapsack dynamic programming algorithm. The time complexity is O(n * c) where n is the number of lobsters and c the maximum capacity.

Algorithm 1 (Knapsack)

```
Let a table  $K[0..n][0..C]$
for i = 0 to n do
    for cap = 0 to C do
        if i == 0 or cap == 0 then
            $K[i][cap] = 0$
        else if w[i] <= cap then
            $K[i][cap] = \max(v[i] + K[i-1][cap - w[i]], K[i-1][cap])$
        else
            $K[i][cap] = K[i-1][cap]$
        end if
    end for
end for
return $K[n][C]$
```

2.2 Explanation of the knapsack algorithm

Let's detail the steps of the algorithm:

- We define a table K where $K[i][cap]$ represents the maximum value that can be obtained using the first i objects and having the capacity *cap* available in the knapsack.
- We initialize the table K with zero for base cases where either the number of objects is zero, or the knapsack capacity is zero.
- For each object i and each capacity *cap*:
    - If the size of object i is less than or equal to the capacity *cap*, then we have two options:
        * Include object i and add its value to the optimal value of the knapsack with remaining capacity $cap - w[i]$.

        * Do not include object i and take the optimal value of the knapsack without this object.

    - We choose the option that offers the maximum value.
    - If the size of object i is greater than the capacity cap, we cannot include the object, so we keep the optimal value without this object.

- The final result, i.e., the maximum value that can be obtained with capacity C using all the objects, is found in $K[n][C]$.

2.3 Algorithm for enumerating selected items

To determine which objects have been selected to achieve the maximum value, we can traverse the table K in reverse order, starting from $K[n][C]$:

Algorithm 2 Algorithm for Enumerating Selected Items

```
1: Initialize the empty list  selectedItems
2: cap = $C$
3: for i = n to 1  with step  -1 do
4:    if K[i][cap] != K[i-1][cap] then
5:       Add object  i  to  selectedItems
6:       cap = cap − $w[i]$
7:    end if
8: end for
9: return selectedItems
```

2.4 Explanation of the Algorithm for Enumerating Selected Items

The algorithm for enumerating selected items utilizes the table K to identify the objects included in the optimal solution:

- We start from position $K[n][C]$ and check if the current value differs from the one above it $K[i-1][cap]$.

- If the values are different, it means object i was included in the knapsack. We add the object to the list of selected items and decrease the available capacity cap by the size of object $i-1$.

- We continue this process until we reach the first object.

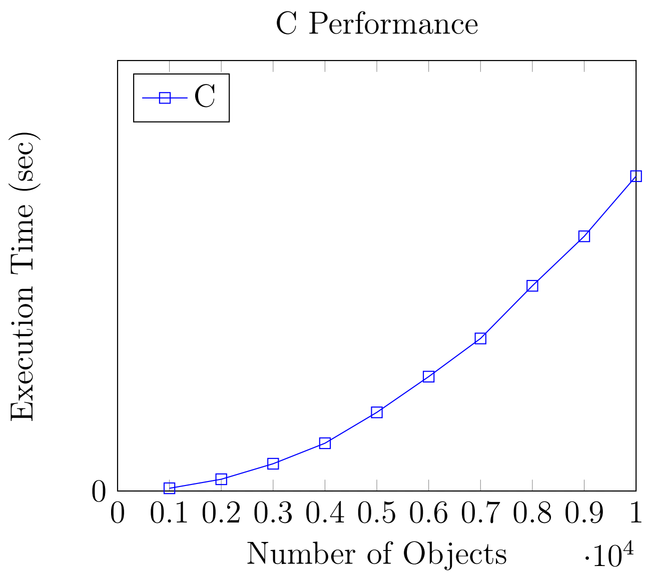
- The resulting list, selectedItems, contains the objects that were included to achieve the maximum value.

The complexity of the enumeration algorithm is O(number of objects), so it does not affect the overall program complexity.

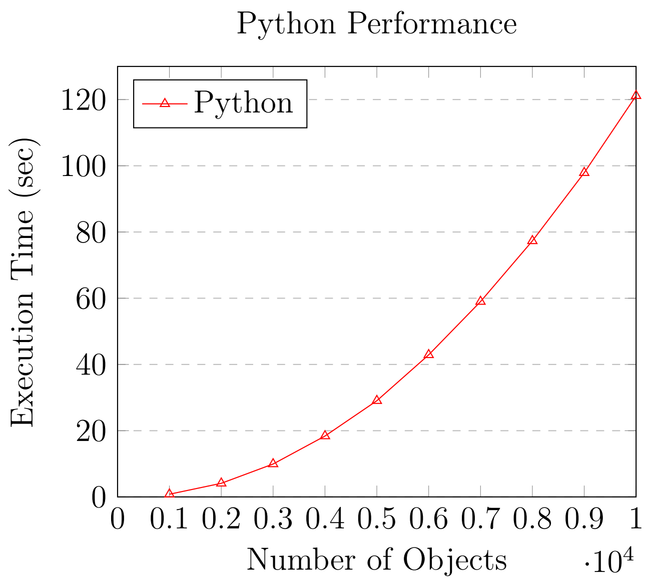
3 Experimental Data

The program was ran on ten test files. Each test file was randomly generated using the "test-generator" program. The data had lobsters with names less than 100 charachter, with sizes and values in between 1 and 1000. The first test file has 1000 lobsters and 1000 maximum bag capacity. It then grows by 1000 for every test file. The parametres used for generating the files can be found in the params folder. The same test files were used for testing both the C and Python implementation. Both implementations produced the same results.

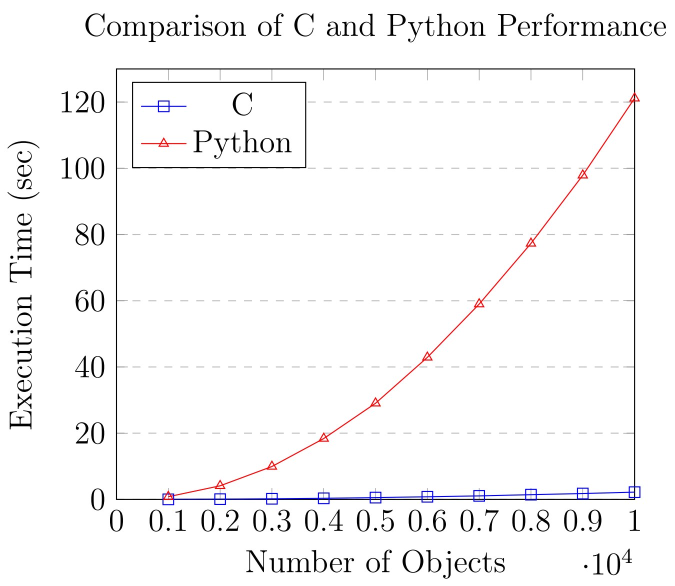
Here is a graph mapping the C implementation execution time to the number of objects in each test case.



The execution time grows exponentially, because the time complexity is O(n*c), and here c is equal to n. The same is true of the Python implementation, but it is much slower (50x).



The next graph compares the C and Python implementations.



4 Tools used

All of the tools I used during the development of this program are free and open source. For builiding the project I used the make utility with the gcc compiler, and for writing, vim, a text editor. Th OS used was Arch Linux, also FOSS. During development I ran into runtime errors, and I used Valgrind for debugging. For writing the report Latex was used because it is plainly better than proprietary alternatives like Microsof Word.

5 Results and conclusions

My findings confirm the correctness of the algorithm across all tested scenarios, demonstrating its reliability and robustness in producing accurate results. A detailed comparison between the C and Python implementations of the algorithm revealed significant insights. The C implementation, while more challenging and time-consuming to code due to its lower-level nature and stricter syntax, proved to be considerably faster in execution. This speed advantage makes C an ideal choice for performance-critical applications. Conversely, the Python implementation, characterized by its simplicity and ease of coding, albeit slower in execution, is more suitable for rapid development and prototyping where development time is a higher priority than execution speed.

The evaluation of software tools highlighted the superiority of Free and Open Source Software (FOSS) tools in terms of flexibility, cost-efficiency, and community support. These tools not only provided comprehensive functionalities but also facilitated easier integration and customization, making them preferable for both development and deployment.

A possible continuation of this project could involve implementing the algorithm in a functional programming language such as Haskell. This would provide an opportunity to compare the functional paradigm with the imperative nature of C and the object-oriented approach of Python, potentially uncovering new insights into the efficiency and expressiveness of different programming paradigms.

In conclusion, this report has demonstrated the algorithm's correctness and provided a comprehensive evaluation of different development tools and programming languages. The insights gained emphasize the strengths of FOSS tools and the trade-offs between coding ease and execution speed in C and Python