FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

Report
on the practical task No. 8
“Practical analysis of advanced algorithms”

Performed by
Anastasia Miroshnikova
j4133c
Accepted by
Dr Petr Chunaev

St. Petersburg
2020

Goal

Practical analysis of advanced algorithms.

Formulation of the problem

I. Choose two algorithms (interesting to you and not considered in the course) from the above-mentioned book sections.
II. Analyse the chosen algorithms in terms of time and space complexity, design technique used, etc. Implement the algorithms and produce several experiments. Analyse the results.

Brief theoretical part

Two classical algorithms were chosen for the analysis. The first one is matrix-chain multiplication, and the second one is Floyd-Warshall algorithm for finding all shortest paths between vertices in a directed weighted graph. Both of the algorithms use dynamic programming as a main technique.

Matrix-chain multiplication

1) Finding optimal substructure

Let us identify subproblem as computing the optimal parenthesization of product $A_i \cdot A_{i+1} \cdot \ldots \cdot A_j$, where $i \leqslant j$. Suppose that optimal parenthesization splits the product at the A_k, A_{k+1}. Then the parenthesization of the "prefix" subchain $A_i \cdot A_{i+1} \cdot \ldots \cdot A_k$ is the optimal parenthesization. If it were not optimal, then the optimal parenthesization of the larger subproblem $A_i \cdot A_{i+1} \cdot \ldots \cdot A_j$ would had less that optimal cost, which is impossible.

Thus, the way to construct the optimal solution from optimal solutions to subproblems.

2) The recursive solution

Next, we define the cost of an optimal solution recursively in terms of the optimal solutions to subproblems. The subproblem is formulated as computing the cost of multiplication $A_i \cdot A_{i+1} \cdot \ldots \cdot A_j$. Let *m[i, j]* be the minimum number of required operations for the subproblem. *m[i, j]* equals to 0 if $i = j$ since no multiplication is required. Let us assume that the optimal parenthesization splits the product $A_i \cdot A_{i+1} \cdot \ldots \cdot A_j$ between A_k and A_{k+1}, where $i \leq k < j$. Then, *m[i, j]* is equal to the minimum cost for computing the subproducts $A_{i \ldots k}$ and $A_{k+1 \ldots j}$, plus the cost of multiplying these two matrices together. Then the optimal cost is

$$m[i, j] = \begin{cases} 0 & \text{if } i = j, \\ \min\limits_{i \leq k < j} \{m[i, k] + m[k+1, j] + p_{i-1} p_k p_j\} & \text{if } i < j. \end{cases}$$

To help us keep track of how to construct an optimal solution, let us define *s[i, j]* to be a value of *k* at which we can split the product A i A i+1 A j to obtain an optimal

parenthesization. That is, *s[i, j]* equals a value *k* such that $m[i,j]=m[i,k]+m[k+1,j]+p_{i-1}\cdot p_k\cdot p_j$.

3) Computing the optimal costs

If we just implemented the recursive algorithm at this point, we would get an algorithm that takes exponential time to run. So we instead find the restrictions that can be used to improve it.

We have relatively few subproblems: one subproblem for each choice of *i* and *j* satisfying $1 \le i \le j \le n$, $C_n^2+n=O(n^2)$, that is, the maximum number of subproblems is $O(n^2)$.

To construct optimal solution we use bottom-up approach. The algorithm should fill in the table m in a manner that corresponds to solving the parenthesization problem on matrix chains of increasing length.

The algorithm constructed can be found by the link in the appendix.

The time complexity of the algorithm is $O(n^3)$ – we have $O(n^2)$ subproblems each of which is solved in O(n) time. The space complexity is $O(n^2)$ since we have to store additional matrix [n, n].

Floyd-Warshall algorithm

1) Optimal substructure

The algorithm considers the "intermediate" vertices of a shortest path, where an intermediate vertex of a simple path $p=v_1,v_2,\ldots,v_l$ is any vertex of *p* other than v_1 or v_l, that is, any vertex in the set $v_2,v_3,\ldots,v_{l-1}$.

The Floyd-Warshall algorithm is based on the following observation. Under our assumption that the vertices of *G* are $V=1,2,\ldots,n$, let us consider a subset $1,2,\ldots,k$ of vertices for some *k*. For any pair of vertices $i, j\in V$, consider all paths from *i* to *j* whose intermediate vertices are all drawn from $1,2,\ldots,k$, and let *p* be a minimum-weight path from among them. Consider that path *p* is simple. The Floyd-Warshall algorithm exploits a relationship between path *p* and shortest paths from *i* to *j* with all intermediate vertices in the set $1,2,\ldots,k-1$. The relationship depends on whether or not *k* is an intermediate vertex of path *p*.

- If *k* is not an intermediate vertex of path *p*, then all intermediate vertices of path *p* are in the set $1,2,\ldots,k-1$. Thus, a shortest path from vertex *i* to vertex *j* with all intermediate vertices in the set $1,2,\ldots,k-1$ is also a shortest path from *i* to *j* with all intermediate vertices in the set $1,2,\ldots,k$.
- If *k* is an intermediate vertex of path *p*, then we break *p* down into two parts by vertex k. p_1 is a shortest path from *i* to *k* with all intermediate vertices in the set $1,2,\ldots,k$. Because vertex *k* is not an intermediate vertex of path p_1, we see that p_1 is a shortest path from *i* to *k* with all intermediate vertices in the set

$1,2,\ldots,k-1$. Similarly, p_2 is a shortest path from vertex k to vertex j with all intermediate vertices in the set $1,2,\ldots,k-1$.

2) Recursive solution

Let $d_{ij}^{(k)}$ be the weight of a shortest path from vertex i to vertex j for which all intermediate vertices are in the set $1,2,\ldots,k$. When $k = 0$, a path from vertex i to vertex j with no intermediate vertex numbered higher than 0 has no intermediate vertices at all. Such a path has at most one edge, and hence $d_{ij}^{(k)}=w_{ij}$. A recursive formula is as folows:

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1. \end{cases}$$

The final solution is constructed with bottom-up technique again and is available by the link in the appendix.

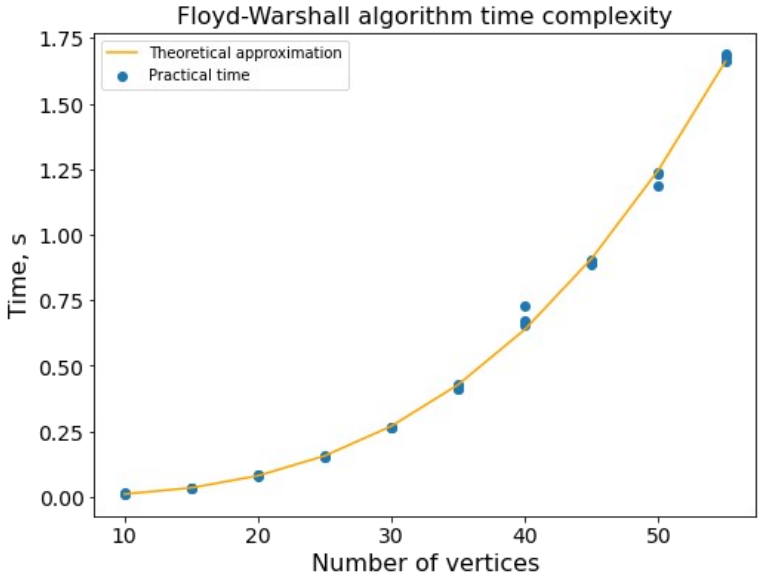
The time complexity is $O(n^3)$ since the algorithm contains 3 loops with length n each.

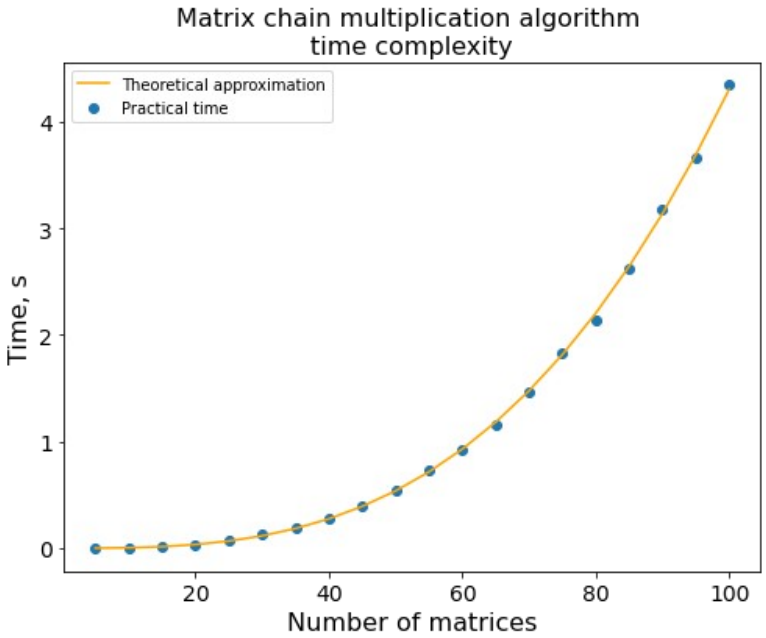
Results

Algorithms were implemented using Python 3.6.9 programming language.

The experiments conducted were to measure time complexity based on the size of the input. The random input was generated for each algorithm. For Floyd-Warshall algorithm it was the random weighted undirected graph with positive and negative weights with |V| varying from 10 to 50 and |E| having value from {2|V|, 3|V|, 4|V|}. For matrix-chain multiplication algorithm it was an array with random shapes of matrices. The size of an array (number of matrices) was varying from 5 to 100.

The time complexity of both Floyd-Warshall algorithm and matrix-chain multiplication algorithm happened to coincide with the theoretical approximation $O(n^3)$.





Conclusions

The dynamic programming approach for solving all shortest paths problem and matrix chain multiplication problem was discussed. Both algorithms solving the problems were implemented and the experiment of measuring time complexity was conducted. The measured times coincided with the theoretical values of $O(n^3)$.

Appendix

Source code can be found at https://github.com/Miffka/algo_itmo/tree/master/task8.