+FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION

OF HIGHER EDUCATION

ITMO UNIVERSITY

**Report**

**on the practical task No. 8**

**“(Extended). Practical analysis of advanced algorithms”**

Performed by

Chernobrovkin Timofey (412642)

Academic group J4133c

Accepted by

Dr Petr Chunaev

St. Petersburg

2023

# Goal

Practical analysis of advanced algorithms.

# 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.

# Theory

## Maximum subarray problem

In computer science, the maximum sum subarray problem, also known as the maximum segment sum problem, is the task of finding a contiguous subarray with the largest sum, within a given one-dimensional array A[1...n] of numbers. It can be solved in O(n) time and O(1) space.

Formally, the task is to find indices i and j with 1≤i≤j≤n, such that the sum is as large as possible. (Some formulations of the problem also allow the empty subarray to be considered; by convention, the sum of all values of the empty subarray is zero.) Each number in the input array A could be positive, negative, or zero.

Some properties of this problem are:

* If the array contains all non-negative numbers, then the problem is trivial; a maximum subarray is the entire array.
* If the array contains all non-positive numbers, then a solution is any subarray of size 1 containing the maximal value of the array (or the empty subarray, if it is permitted).
* Several different sub-arrays may have the same maximum sum.

Although this problem can be solved using several different algorithmic techniques, including brute force, divide and conquer, dynamic programming, and reduction to shortest paths, a simple single-pass algorithm known as Kadane's algorithm solves it efficiently.

Maximum subarray problems arise in many fields, such as genomic sequence analysis and computer vision.

Kadane's algorithm scans the given array A[1…n] from left to right. In the j-th step, it computes the subarray with the largest sum ending at j; this sum is maintained in variable *current\_sum*. Moreover, it computes the subarray with the largest sum anywhere in A[1…j], maintained in variable *best\_sum*, and easily obtained as the maximum of all values of *current\_sum* seen so far.

As a loop invariant, in the j-th step, the old value of *current\_sum* holds the maximum over all i ∈{1, …, j-1} of the sum A[i] + … +A[j-1]. Therefore, *current\_sum* + A[j] is the maximum over all i ∈{1, …, j-1} of the sum A[i] + … + A[j]. To extend the latter maximum to cover also the case i=j, it is sufficient to consider also the singleton subarray A[j … j]. This is done in line 6 by assigning max(A[j], *current\_sum* + A[j]) as the new value of *current\_sum*, which after that holds the maximum over all i ∈{1, …, j}of the sum A[i] + … + A[j].

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

If the input contains no positive element, the returned value is that of the largest element (i.e., the value closest to 0), or negative infinity if the input was empty. For correctness, an exception should be raised when the input array is empty, since an empty array has no maximum nonempty subarray. If the array is nonempty, its first element could be used in place of negative infinity, if needed to avoid mixing numeric and non-numeric values.

The algorithm can be adapted to the case which allows empty subarrays or to keep track of the starting and ending indices of the maximum subarray.

This algorithm calculates the maximum subarray ending at each position from the maximum subarray ending at the previous position, so it can be viewed as a trivial case of dynamic programming.

The runtime complexity of Kadane's algorithm is O(n) and its space complexity is O(1).

## Activity selection problem

The activity selection problem is a combinatorial optimization problem concerning the selection of non-conflicting activities to perform within a given time frame, given a set of activities each marked by a start time (si) and finish time (fi). The problem is to select the maximum number of activities that can be performed by a single person or machine, assuming that a person can only work on a single activity at a time. The activity selection problem is also known as the Interval scheduling maximization problem (ISMP), which is a special type of the more general Interval Scheduling problem.

A classic application of this problem is in scheduling a room for multiple competing events, each having its own time requirements (start and end time), and many more arise within the framework of operations research.

Assume there exist n activities with each of them being represented by a start time si and finish time fi. Two activities i and j are said to be non-conflicting if si ≥ fj or sj ≥ fi. The activity selection problem consists in finding the maximal solution set (S) of non-conflicting activities, or more precisely there must exist no solution set S' such that |S'| > |S| in the case that multiple maximal solutions have equal sizes.

The activity selection problem is notable in that using a greedy algorithm to find a solution will always result in an optimal solution.

# Materials and methods

In this task, all calculations were performed on the student's personal laptop. The work was performed in the Python programming language. The algorithms for the study were selected from the book Thomas H. Cormen Charles E. Leiserson Ronald L. Rivest Clifford Stein Introduction to Algorithms Third Edition, 2009 (or other editions).

# Results

Maximum subarray problem (page 68) and Activity selection problem (page 415) were taken for the task.

Time complexity dependencies were plotted for the selected algorithms. For a more accurate result, 5 iterations were performed for each step and after that the result was averaged. A total of 2000 iterations were performed. Empirical and theoretical curve was plotted on the time complexity graphs.

The maximum-subarray problem algorithm was implemented. The graph of time dependence on the number of elements of the processed array is shown in Figure 1.

Изображение выглядит как снимок экрана, текст, График, линия

Автоматически созданное описание

Figure 1 – The maximum-subarray problem

For example, let's set the array [-2, -3, 4, -1, -2, 5, -3]. Then at the output of the algorithm we will get the result shown in Figure 2.

Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание

Figure 2 - The maximum-subarray problem result

And indeed the subarray starting with element 2 and ending with element 5 has the largest sum.

The activity-selection problem algorithm was implemented. The graph of time dependence on the number of elements of the processed array is shown in Figure 3.

Изображение выглядит как текст, снимок экрана, График, диаграмма

Автоматически созданное описание

Figure 3 – An activity-selection problem

For example, let's set two arrays with start and end times s = [1, 3, 0, 5, 8, 5] and f = [2, 4, 6, 7, 9, 9]. Then at the output of the algorithm we have the result presented in Figure 4.

In our case, the input was an already sorted array.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Figure 4 – An activity-selection problem result

# Conclusion

The maximum-subarray problem and An activity-selection problem were implemented and analyzed in this assignment. The analysis was carried out on the basis of knowledge and skills obtained during the course of analysis and development of algorithms.

The time complexity of both algorithms turned out to be equal to O(n), which can be clearly seen on the obtained graphs. Space complexity is O(1). However, if the activity-selection problem algorithm takes an unsorted array as input, its time complexity increases to O(n\*logn) and its spatial complexity to O(n).

Both algorithms use an array for storing input data.

# Appendix

GitHub link: