УДК 331.53
Yulia Zalata[1], Svitlana Voitenko[2]
[1]student of group CST–810m, National University «Zaporizhzhia Polytechnic»
[2]senior teacher, National University «Zaporizhzhia Polytechnic»

INVESTIGATION OF THE POSSIBILITIES OF USING TREES IN THE COURSE OF SOLVING KNAPSACK PROBLEM USING DISTRIBUTED CALCULATIONS

The knapsack problem is of interest to researchers, since its solution is associated with a large number of applied problems in economics, logistics and other areas. In recent years, there has been an increasing interest in the use of backpacks as cryptographic primitives in the development of new models of encryption systems. The resistance of such encryption systems to analysis is determined by the complexity of finding an exact solution to the knapsack problem, as a result of which the improvement of accurate algorithms becomes relevant. To improve the efficiency of these algorithms, it is advisable to use distributed and parallel computing.

The general formulation of the knapsack problem varies depending on the specific application area. The knapsack problem is formulated as follows. You are given an ordered set of objects, each of which is associated with a non-negative, integer weight value. The task is to find all possible sets of these items, the weight of which is exactly equal to the given value. Each of these sets is hereinafter referred to as "packing the backpack", the total weight of the items in this set is the weight of the packing. As already noted, there are a large number of algorithms for solving the knapsack problem, but only a few of them are designed to find the exact solution. This is due to the fact that, in a significant number of practical problems, the quantities appearing in the knapsack problem have physical meaning. For example, when planning the loading of a transport ship, items in a backpack can be characterized by a real value of weight or volume. A solution approximated in terms of total weight / volume may be quite acceptable. This makes it relevant to study algorithms that are able to find quickly a solution that is close to the optimum. Such algorithms, in general, have exponential complexity, which means that even for a small number of objects, finding an exact solution to the knapsack problem will require a significant investment of time. This makes it advisable to use parallel and distributed computing.

There are several known approaches to finding the exact solution to the knapsack problem. The basic approach involves enumerating all possible layouts and checking the resulting weight values for each of them. The transition from this algorithm to a more efficient one is possible by reducing (cutting off) a certain number of packings, which obviously cannot contain solutions.

Two more algorithms for the exact solution of the knapsack problem are based on this principle: the algorithm for traversing the tree of packing options and the algorithm for merging Horowitz-Sani lists. The list merge algorithm is based on the idea of dividing the original set of items into two subsets. For each of the subsets, the weights of all possible packings are calculated, after which the lists of weights are sorted, and the elements from these lists are summed up in pairs. Since both lists of calculated weights are sorted, some of the pairs (the sum of which exceeds the specified weight value) can be excluded from consideration. The price for this is the high complexity of memory: it is necessary to store all intermediate values of the weights in memory. This makes the algorithm unusable for cryptographic analysis purposes. The algorithm for traversing a tree of packing options is based on representing the set of packing in the form of vertices of a connected graph. By traversing the graph using well-known algorithms, one can check all possible packing by analogy with the exhaustive search algorithm. The gain from using this algorithm lies in the possibility of cutting off tree branches that obviously do not contain solutions.

The introduction of the linearization algorithm into the distributed implementation of the search tree traversal algorithm will solve the problem of load sharing between processors. This solution will not be ideal, since the efficiency of the algorithm will decrease as the number of computational nodes used increases. Moreover, pruning subtrees during analysis can lead to an unpredictable reduction in computational load; as a consequence, it is required to provide dynamic load balancing, which will further complicate the algorithm and reduce its efficiency.