Knapsack – Dynamic programming

Stephen Derenski

4/27/2022

CSC250 - Algorithms and Data Structures 1

When it comes to the Knapsack Problem there are a few things that come up due to it, Dynamic Programming being one of them. Though before going into those areas, the Knapsack Problem is associated with combination optimization. Given a set of items, all having weight and value to them. Determine if the number of items is less than the given weight limit and has the highest total value. The given “knapsack” will have a fixed size to work with. Though the most common use of this problem in the modern-day is by building different methods for cryptography by pushing the limits of different methods to develop security codes. For instance, the National Institute of Standards and Technology (NIST) has found an algorithm that uses the knapsack problem in multiple dimensions to develop keys for cryptography.

Dynamic Programming is a style to solve problems that are optimal substructures and overlapping sub-problems. Optimal substructures are solutions to a given optimization problem that can be obtained by the combination of optimal solutions to its sub-problems. Whereas, overlapping sub-problems mean that the space of sub-problems must be small. The recursive algorithm is just doing the same problem repeatedly without generating new subproblems.

This is where the two different approaches come into play for solving them to make the problems more efficiently. The first is a Top-Down Approach to designing the algorithm to handle the sub-problems. It does so by recursively using the solution to its sub-problems, and if its sub-problems are overlapping, then one can easily store the solution to the sub-problems in a table. The other is a Bottom-Up Approach. It tries to solve the sub-problems first and use its solutions to build on and arrive at solutions to bigger sub-problems. This also uses a recursive algorithm structure to do so. Both cut down on the time it takes to solve a problem. Now, they don’t exactly translate to efficient solutions, just more optimized versions instead of brute-forcing them. They are used on anything that has overlapping subproblems, like the longest common subsequence or the knapsack problem.

By using one of these approaches on the knapsack problem, it reduces the growth rate of O(n2n) down to roughly O(N\*Capacity), which is a significantly slower growth rate than before. It does so by adding the solutions to a map and iterating through it as it solves the problem. When it comes across information already on the table, it discards the new information and moves on. While this might not give the exact right solution to a problem with overlapping subproblems, it does help in reducing the time it takes. Being able to get a reasonable answer promptly is better than taking ages to find the absolute best answer. Efficiently being correct is important with this style of programming and problems.