Summarizing the content:

1. Summaries the main points in this module. You may include references to the learning objectives.
2. Longest Common Subsequence
3. Knapsack Problem
4. Unbounded Knapsack
5. 0/1 Knapsack
6. How is this useful?
7. Dynamic programming can exchange space for time, and the running time of the algorithm is greatly reduced, but the space used by the algorithm is increased.
8. Compared with the greedy algorithm, the global optimal solution can be obtained.
9. How do you plan to use this information?

There are two types of problems that dynamic programming can solve:

1. Identify optimal sub-structure.
2. Take advantage of overlapping sub-problems.
3. Provide summary of your reading list — external resources, websites, book chapters, code libraries, etc.
4. <https://www.geeksforgeeks.org/dynamic-programming/>
5. <https://en.wikipedia.org/wiki/Knapsack_problem>

**Reflecting on the content:**

1. What is the most important thing you learnt in this module?

The steps of dynamic programming:

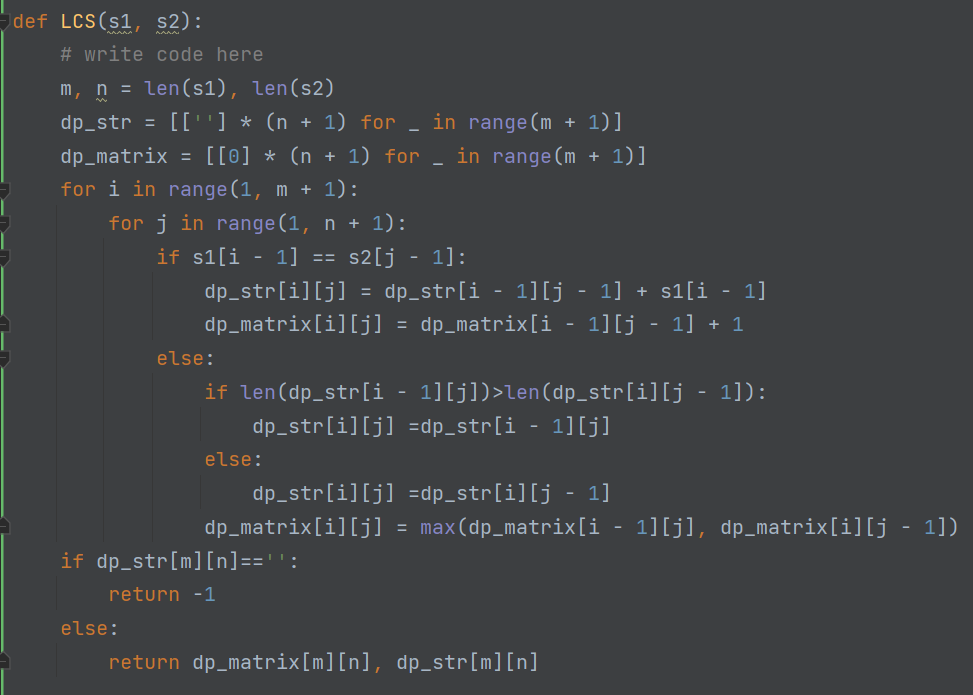
1. Identify optimal substructure.
2. Find a recursive formulation for the length of the longest common subsequence.
3. Use dynamic programming to find the length of the longest common subsequence.
4. If needed, keep track of some additional info so that the algorithm from Step 3 can find the actual LCS.
5. If needed, code this up like a reasonable person.
6. How does this relate to what you already know?

Dynamic programming is also a divide and conquer idea (for example, its state transition equation is a kind of divide and conquer), but unlike the divide and conquer algorithm, the divide and conquer algorithm decomposes the original problem into several sub-problems, and solves the sub-problems top-down. , combine the solutions of the subproblems to get the solution of the original problem. Dynamic programming also decomposes the original problem into several sub-problems, and then solves the smallest sub-problem from the bottom up, and stores the results in a table. When solving a large sub-problem, directly query the solution of the small sub-problem from the table. Avoid double calculations, thereby improving algorithm efficiency.

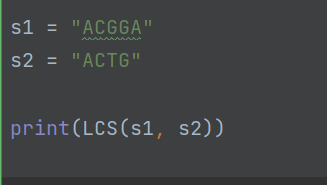
Activity1:

Write the code for finding the Longest Common Sub-sequence. Make sure you output the Matrix C and the longest sub-sequence. Test your code with various use-cases.

Code:

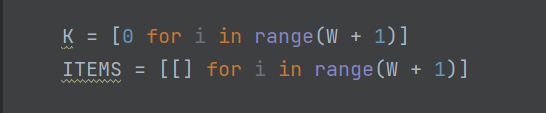


Result:

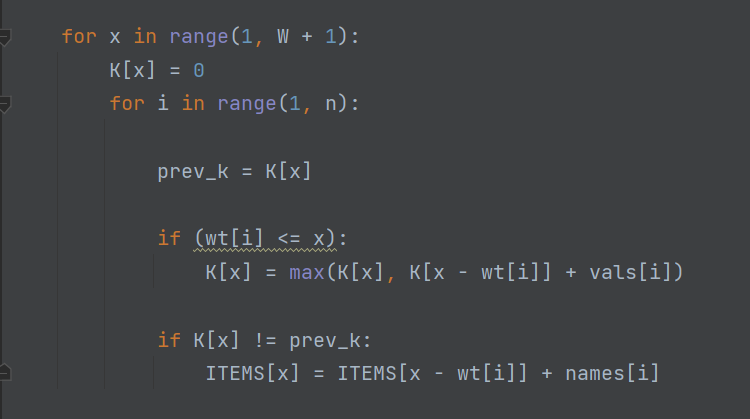


Activity2:

Have a look at the associated ipython notebook and run the code for Unbounded Knapsack Problem. Demonstrate your understanding of the code by (doing some of the following activities):

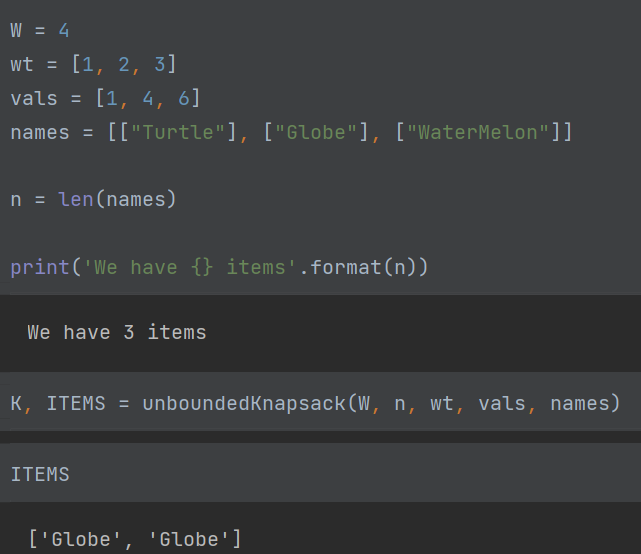


K represents the value of the existing items in the backpack, and ITEMS represents the existing items in the backpack.

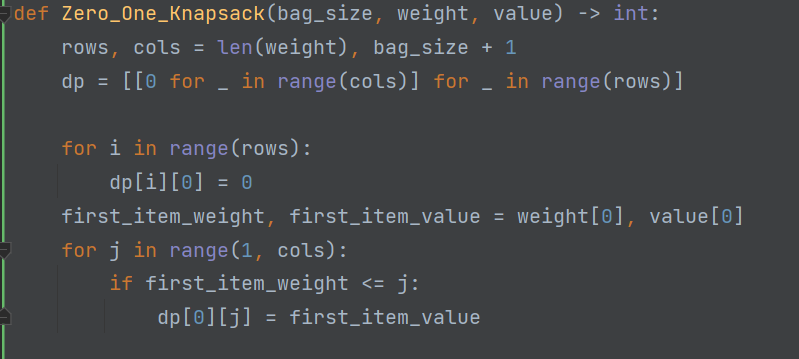


The outer loop loops over the weight of the backpack, and the inner loop loops over the items, using the iterative formula of dynamic programming.

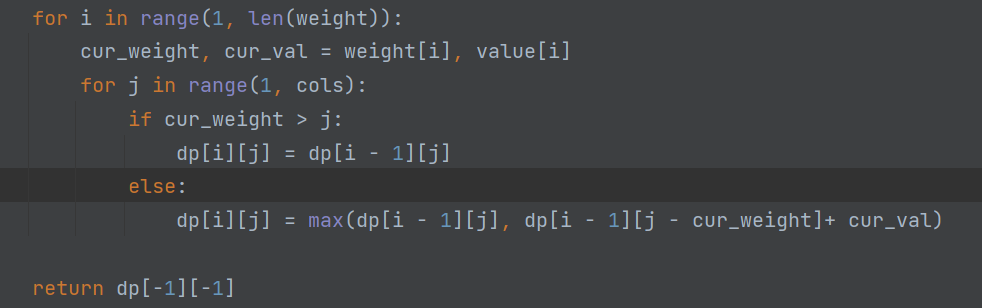
Program running result:



Initialize the array:



Recursive iterative formula:



Program running result:

