1. **Solve the following 0-1 Knapsack Problem Problem using the dynamic programming algorithm covered in the screencast.**

W=10 n = 4 items with Value, Weight : (10,5)1, (40,4)2 , (30,6)3 , (50,3)4

Recurrence relation for VKnap (n, W)= max{V(n-1, W), V(n-1, W-wn) + vn}

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| i=0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 10 | 10 | 10 | 10 | 10 | 10 |
| 2 | 0 | 0 | 0 | 0 | 40 |  |  |  |  |  |  |
| 3 | 0 |  |  |  |  |  |  |  |  |  |  |
| 4 | 0 |  |  |  |  |  |  |  |  |  |  |

Which items are taken:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Solve the following Longest Increasing Subseqence Problem using the dynamic programming algorithm covered in the screencast.**

Recurrence relation for LLIS (k)= maxi<k and xi < xk {L(i)} + 1

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ind | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Xi | 0 | 8 | 4 | 12 | 2 | 10 | 6 | 14 | 1 | 9 | 5 | 13 | 3 | 11 | 7 |
| F(i) | 1 | 2 | 2 | 3 | 2 | 3 | 3 | 4 | 2 | 4 | 3 | 5 | 3 | 5 | 4 |

What are the values in the longest increasing subsequence? 0, 2, 6, 9, 11

1. **Solve the following Longest Common Subseqence Problem using the dynamic programming algorithm covered in the screencast.**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | **A** | **G** | **A** | **C** | **A** | **C** | **T** |
|  | **0** | **0** | **0** | **0** | **0** | **0** | **0** | **0** |
| **G** | **0** | **0** | **1** | **1** | **1** | **1** | **1** | **1** |
| **C** | **0** | **0** | **1** | **1** | **2** | **2** | **2** | **2** |
| **A** | **0** | **1** | **1** | **2** | **2** | **3** | **3** | **3** |
| **G** | **0** | **1** | **2** | **2** | **2** | **3** | **3** | **3** |
| **A** | **0** | **1** | **2** | **3** | **3** | **3** | **3** | **3** |

What is the longest common subsequence? AGA

1. **Knapsack with Repeats**

There are many variations on the Knapsack problem. In this question, consider the same set up as the 0-1 Knapsack problem **except** suppose there are as many copies of each item as you would like. Thus, if one of the items is small but very valuable, it is possible to put many exact duplicates of that item into the knapsack. To maximize the value of items in the knapsack, the thief may decide to put in as many of the high value, low weight items as possible into the Knapsack.

To help the thief decide what items to take you will write an algorithm that guarantees to find the choice of items that maximizes the value of the items that the thief places in the Knapsack.

* 1. Find a counterexample that shows that taking **most valuable item** first will not maximize the value placed in the knapsack.

If my most valuable item is (50, 3) and the sack has a total capacity of 3, we can only take one and we will get a total value of 50. But say we have another item (30, 1). Here we can take 3 of this other item and get a total value of 90.

* 1. Find a counterexample that shows that taking **the smallest item** first will not maximize the value placed in the knapsack

If my smallest item is (1, 1) and the sack can hold 3, then we have a total value of 3. But if we have another item (50, 3) then if we take with item, we have a total value of 50.

* 1. Find a counterexample that shows that taking the **item with the highest value/weight ratio** first will not maximize the value placed in the knapsack

If the item with the highest value to weight ratio is (10, 2) and we have a total capacity of 3. Then the total value would be 10. Then if we have an item (4, 1) we would get a total value of 12.

Giving up on the greedy approach and not seeing how to approach this problem using divide and conquer, you decide to try dynamic programming.

1. **Specify the function that represents the quantity to be optimized**.

Let V(n, W) represent the maximum value for a set of n items and a capacity of W where items can be repeated.

1. Give the **recurrence relation** that describes the optimal substructure of the problem using

VKnap (n, W)= max{V(n-1, W), V(n, W-wn) + vn}

1. Give the **specification of the table** that you would use in a bottom up programmatic solution. Specify the dimensions of the table and what each entry in the table represents.

We just need a matrix of size n + 1 x W + 1 and initialize the zero column and zero row with zeros to make algorithm easier. Each entry will represent the maximum value for that n items and w capacity while allowing for repeated items.

1. Write the **pseudo code of the algorithm** for filling in the table that you would use in a bottom up programmatic solution. That is convert the recurrence relation (part **B.**) to an **iterative** algorithm.

FillTable()

For i from 0 to n -> V[i][0] = 0

For i from 0 to w -> V[0][i] = 0

For i from 1 to n

For j from 1 to w

V[i][j] = max{V(i-1, j), V(i, j-wi) + vi}

Return V;

1. Write the **pseudo code of the algorithm** for tracing back through the table to find the set of items that gives the maximum total value.

TraceBack()

Set[]

J = w;

For i from n to 1

If V[i][j] == V[i - 1][j]

Continue;

Else

While(V[i][j] % vi)

Set.append(i)

V[i][j] = V[i][j] – vi

J = j – wi

If j <= 0

Break;

Return set[]

1. Write the asymptotic complexity of filling in the table.

Filling in the table takes O(m\*n) time complexity.

1. **Min Stress Consulting**

Suppose that you run a simple consulting business. Your clients are mostly clustered on the East and West coasts. From month to month, you could either run your business from an office in NYC or an office in San Francisco. You have some budget numbers to help. If you run your business out of NYC in month i, you expect to incur costs NYi, and likewise costs SFi,for San Francisco. Furthermore, every time you move from one office to another, you incur a moving cost, M.

Given monthly cost estimates NYi, SFi,, for i=1 ... n, and the moving cost M,

Find the best schedule of where to work each month. For example, suppose that M = 10, and

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | i=1 | i=2 | i=3 | i=4 |
| NYi | 1 | 3 | 20 | 30 |
| SFi | 50 | 20 | 2 | 4 |

then the optimal schedule is [NY, NY, SF, SF].

Follow steps A. to F. to provide a dynamic programming solution to this problem.