1. **Find the optimal Matrix chain multiplication**

Find an optimal association (the ordering of carrying out the multiplications) of a matrix chain product with dimensions {d0=5, d1=10, d2=3, d3=12, d4=5, d5=20, d6=6}that is 6 matrices:   
 [5x10] \* [10x3] \* [3x12] \* [12x5]\*[5x20]\*[20x6] matrix etc.

Computing M(i,j)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i\j | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 0 | 150 | 870 |  |  |  |
| 2 |  | 0 | 360 |  |  |  |
| 3 |  |  | 0 | 180 |  |  |
| 4 |  |  |  | 0 | 1200 |  |
| 5 |  |  |  |  | 0 | 600 |
| 6 |  |  |  |  |  | 0 |

index of dim that splits the last multiplication – the k that is used

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i\j | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 |  | 1 |  |  |  |  |
| 2 |  |  | 2 |  |  |  |
| 3 |  |  |  | 3 |  |  |
| 4 |  |  |  |  | 4 |  |
| 5 |  |  |  |  |  | 5 |
| 6 |  |  |  |  |  |  |

1. **Longest Common Substring Problem:** Given two string sequences write an algorithm to find, find the length of longest substring present in both of the string sequences. A **longest common substring** is a sequence of characters that appears in the same order and are contiguous in both the strings.

Consider the following to Strings AABCDTYU FAGAHBCLD

**Longest Common Subsequence** would find **ABCD**

A**ABCD**TYU FAG**A**H**BC**L**D**

But **Longest Common Substring** would find **BC**

AA**BC**DTYU FAGAH**BC**LD

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

Let LCS(i, j) be the function that represents the longest common substring of two strings of size i and j.

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

LCS(i, j) = if char at i and j match then LCS(i-1, j-1) + 1; else 0

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 need a table of size i+1 x j+1. Each entry will represent the longest common substring of that location i, j.

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.

I = length of 1st string + 1

J = length of 2nd string + 1

For x from 0 to i

For y from 0 to j

If x == 0 and y == 0 then

LCS(x, y) = 0;

Else if chars at x and y match

LCS(x, y) = LCS(x – 1, y - 1) + 1;

Else

LCS(x, y) = 0

Return LCS

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.
2. Find max value in matrix.
3. Prepend char of max value in matrix to array
4. Keep prepending char at loc of max [i – 1, j - 1] until you reach a value of zero at the location.
5. Return array
6. Write the asymptotic complexity of filling in the table.

Filling in the table takes O(m\*n) complexity because we fill out an m x n matrix.

1. **Firestones Profit**: Firestones is considering opening a series of restaurants along Highway 1. The n possible locations are along the highway, and the distances of these locations from the downtown San Luis Obispo are, in miles and in increasing order, m1, m2, , mn. The constraints are:

* At each location, Firestones may open at most one restaurant. The expected profit from opening a restaurant at location i is pi, where pi > 0 and i = 1; 2; : : : ; n.
* Any two restaurants must be at least k miles apart, where k is a positive integer.

Give a dynamic programming algorithm to compute the maximum expected total profit subject to the given constraints.

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

Let MP(n) be the max total profit at for a set of n restaurants.

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

MP(n) = maxi<n and (m(n) **–** m(i)) < k{MP(i)} + pn

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 need a table of size n. Each entry will represent the max profit up to and including the store at i.

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.

MP(1) = p1;

For i from 2 to n

MP(i) = maxj<i and( m(i) – m(j) )< k{MP(j)} + pi;

Return MP;

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.
   * 1. Find max value of MP and place idx into array
     2. Profit = max value of MP
     3. i= idx of max value of MP
     4. Get the restaurant that has the biggest MP and is of at least k distance away from the max and its MP plus pi is equal to Profit.
     5. Place that restaurant into the array
     6. Profit = MP restaurant from step 4
     7. i = idx of restaurant from step 4
     8. Repeat steps 4 – 7 until profit is equal to zero.
     9. Return array.
2. Write the asymptotic complexity of filling in the table.

Filling in the table takes O(n2) complexity because we fill out an array of size n once but have to find the max compatible restaurant for each restaurant which takes n steps as well.

1. Change Making Revisited: This is a variation of the change making problem discussed in the screencast. Given an unlimited supply of coins of denominations d1 < d2 < : : : < dn, (**where d1 > 1**) we wish to make change for a value v; that is, we wish to find a set of coins whose total value is v. **In this variation, we want to know if we can make change for a value with k or fewer coins**.

Give a dynamic programming algorithm for the following problem: You are given d1 < d2 < : : : < dn and v. You must answer the following question:   
Is it possible to make change for v using coins of denominations d1 < d2 < : : : < dn using k or fewer coins?

E.g. d1 = 4 d2=7

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| F | F | F | T | F | F | T | T | F | F | T | T | F | T | T | T | F | T | T |

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

F(k) represents whether we can make change for a value give denominations.

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

F(k) = if F(k – dj) == T over j where dj < k then T; else F

Base Case: F(0) = T

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.

The table needs to be size k + 1. Each entry represents whether we can make exact change for the value k.

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.

F(0) = 0;

For i from 1 to k

If (F(i - dj) == T over j where dj < i)

F(i) = T;

Else

F(i) = F;

Return F;

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

Filling the table takes O(n) complexity because we go through the list of n items once.

**Test your DP skills with the following problem!**

1. **Pebbling a checkerboard (Challenge Problem).** Given a checkerboard which has 4 rows and n columns, and has an integer written in each square. We are also given a set of 2n pebbles, and we want to place **some or all** of these on the checkerboard (each pebble can be placed on exactly one square) so as to maximize the sum of the integers in the squares that are covered by pebbles. There is one constraint: for a placement of pebbles to be legal, no two of them can be on horizontally or vertically adjacent squares (diagonal adjacency is fine).
2. Determine the number of legal patterns that can occur in any column (in isolation, ignoring the pebbles in adjacent columns) and describe these patterns. Call two patterns compatible if they can be placed on adjacent columns to form a legal placement. Let us consider subproblems consisting of the first k columns 1≤ k ≤ n where each subproblem is assigned a type, which is the pattern occurring in the last column.

(b) Using the notions of compatibility and type, give an O(n)-time dynamic programming algorithm for computing an optimal placement

1. **Optimal Binary Search Tree**

Find the optimal Binary Search tree for the following set of keys and probabilities

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i | 1 | 2 | 3 | 4 | 5 | 6 |
| pi | 15 | 25 | 5 | 20 | 5 | 30 |

Minimal cost for Tree from i to j

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| i\j | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 0 | 15 | 55 |  |  |  |  |
| 2 |  | 0 | 25 | 35 |  |  |  |
| 3 |  |  | 0 | 5 |  |  |  |
| 4 |  |  |  | 0 | 20 |  |  |
| 5 |  |  |  |  | 0 | 5 |  |
| 6 |  |  |  |  |  | 0 | 30 |
| 7 |  |  |  |  |  |  | 0 |

Tree root

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| i\j | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 |  | 1 | 2 |  |  |  |  |
| 2 |  |  | 2 | 3 |  |  |  |
| 3 |  |  |  | 3 |  |  |  |
| 4 |  |  |  |  | 4 |  |  |
| 5 |  |  |  |  |  | 5 |  |
| 6 |  |  |  |  |  |  | 6 |
| 7 |  |  |  |  |  |  |  |