Top Down - Memoization

Bottoms Up - Dynamic

1. **Matrix Chain Multiplication**

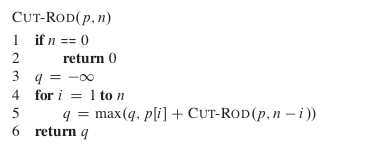
<https://en.wikipedia.org/wiki/Matrix_chain_multiplication>

* Take the sequence of matrices and separate it into two subsequences.
* Find the minimum cost of multiplying out each subsequence.
* Add these costs together, and add in the cost of multiplying the two result matrices.
* Do this for each possible position at which the sequence of matrices can be split, and take the minimum over all of them.

2. **Rod Cutting Problem**

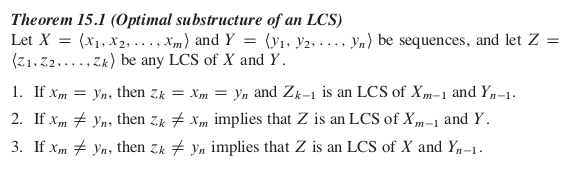
Given a rod of length n inches and a table of prices p i for i = 1, 2,...., n, determine the maximum revenue rn obtainable by cutting up the rod and selling the pieces.

Solution -

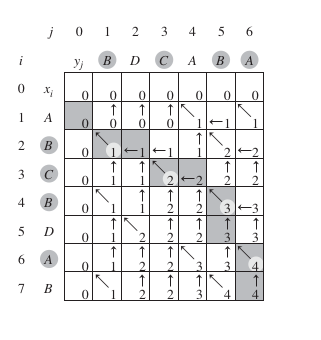


Runtime - O(n2) [Memoized]

**3. Longest Common Subsequence**

****

The following DP table can be used to reconstruct the LCS in O(m+n) time -



<https://www.codechef.com/problems/ORACLCS>

**4. Weighted Job Scheduling**

The basic reasoning is that any job j will either belong to the optimal solution in that case opt[j] will be vj + opt[p(j)] or it will not belong to the optimal solution in that case it will be opt[j-1].

Note - p(j) largest index i<j such that job i ends before job j starts. (basically all remaining jobs excluding overlapping)



Runtime - O(n) with memoization. If jobs are presorted.

**5. Maximum Subarray Problem**

[**https://en.wikipedia.org/wiki/Maximum\_subarray\_problem#Kadane's\_algorithm**](https://en.wikipedia.org/wiki/Maximum_subarray_problem#Kadane's_algorithm)

**6. Maximum Rectangle Problem**

Given a n x n array, find the rectangle with maximum sum.

Solution -

Fix to rows a and b

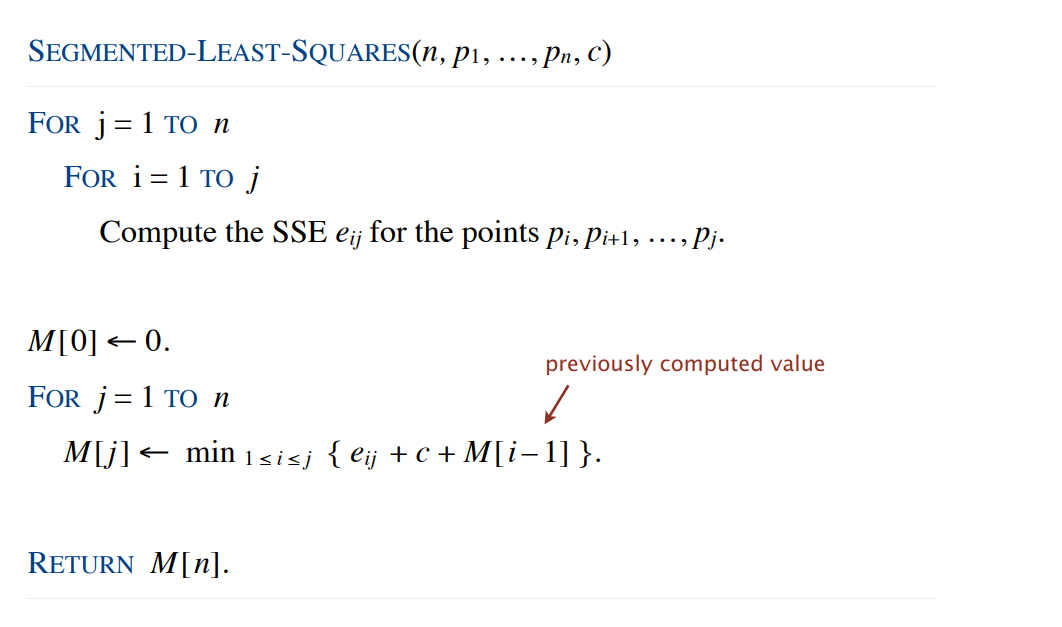
Construct a array containing prefix sum of each rows.

Use Kadane’s algorithm to find the maximum sum subarray - that is the maximum sum subarrays between rows a and b inclusive.

Iterate over all pair of rows in this way.

For more info - Check page 489 of Cracking the Coding Interview

**7. Segmented Least Squares**



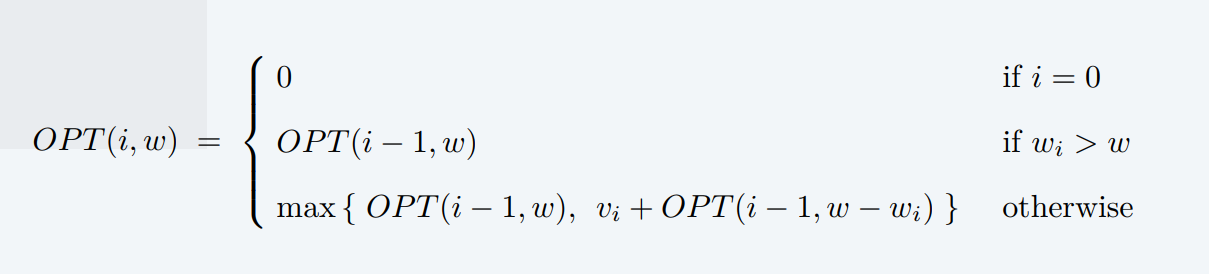
Time Complexity = O(n3)

Space Complexity = O(n2)

**8. Knapsack Problem**

Pack knapsack so as to maximize total value. There are n items: item i provides value vi > 0 and weighs wi > 0. Knapsack has weight capacity of W.

Solution -



Complexity - O(nW) time and space

Retrieval of items - An item i in taken if opt[i,W] > opt[i-1,W]. So traceback.

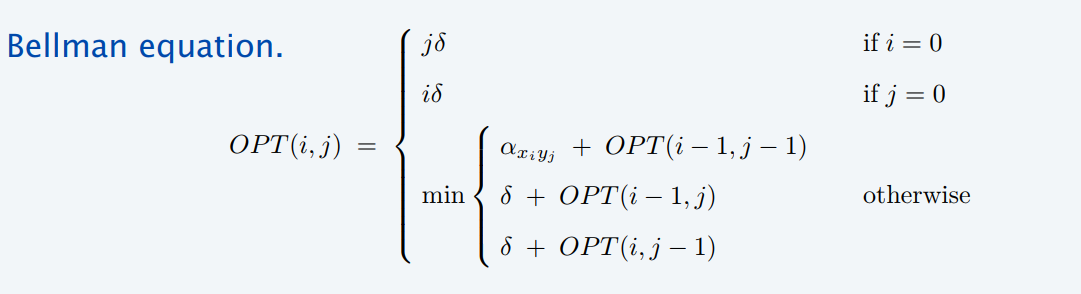
**9. Coin Changing Problem**

Given n coin denominations { c1, c2, …, cn } and a target value V, find the fewest coins needed to make change for V (or report impossible).

Greedy is not optimal for arbitrary coin denominations.

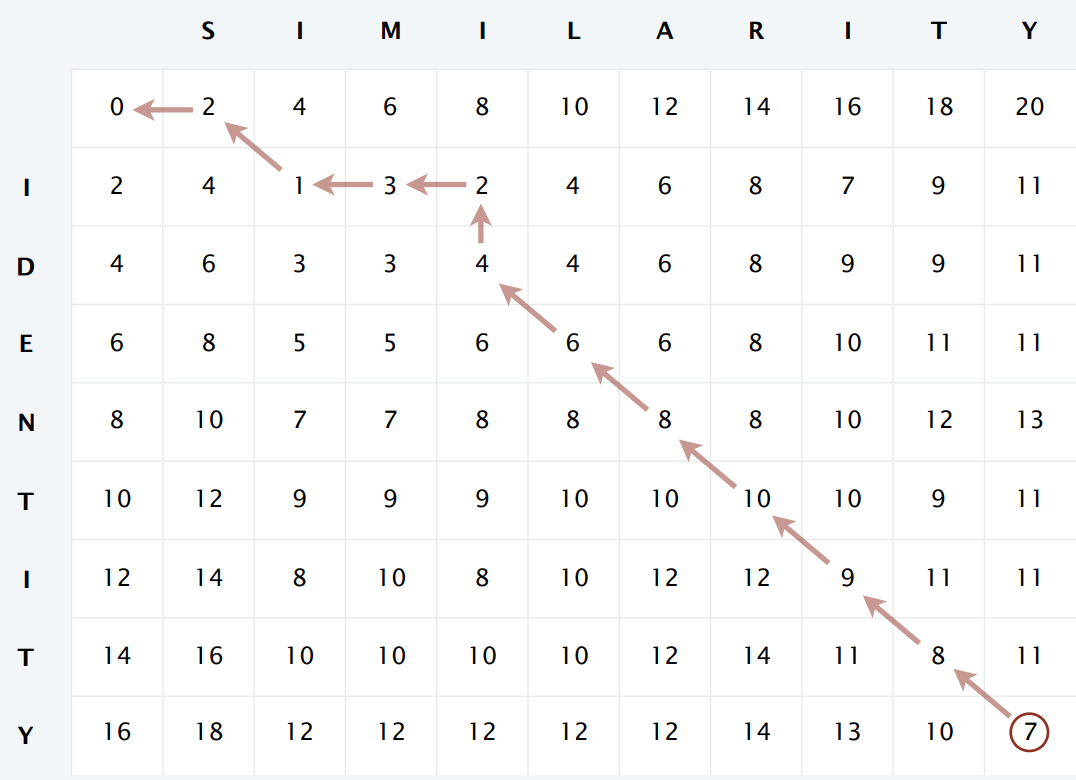
Solution - ?

**10. Edit Distance**

****

Delt = Cost of gap

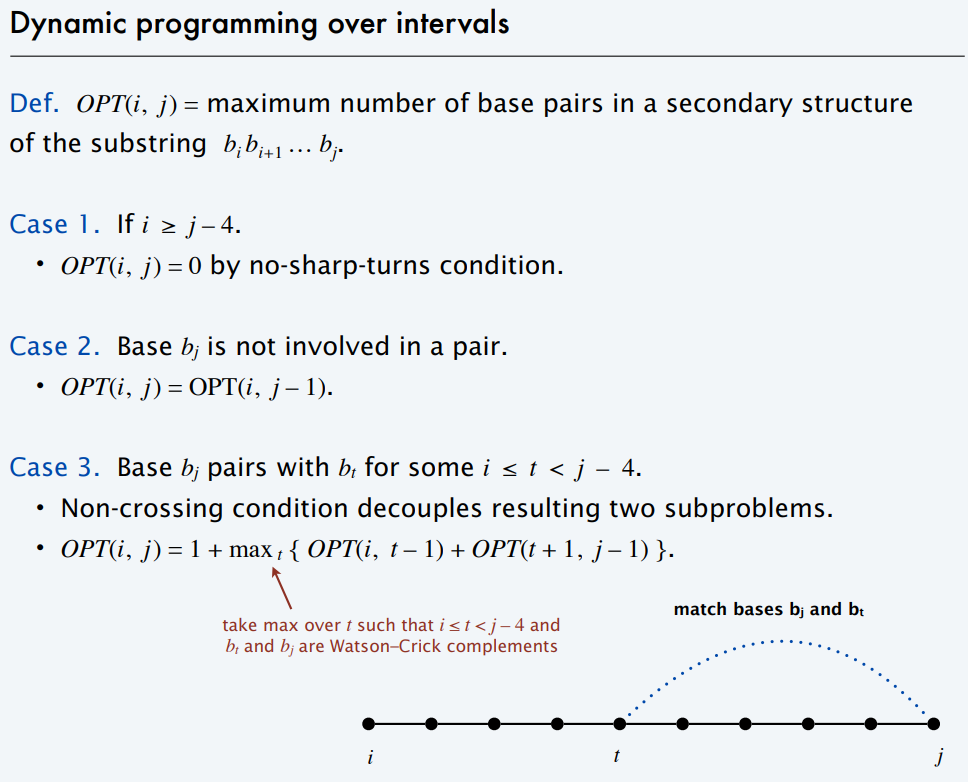
Alpha = Cost of mismatch



Once we have completely filled the DP table, we can check the prev three entries to figure out whether we have chosen a fap of mismatch in constant time and can figure out the final sequence alignment in linear time.

Time/Space Complexity - O(mn)

**11. DP over intervals**

****

**12. Shortest Path with negative weights. (To be Done!!)**

Some Conclusions -

1. If a v - t path have negative cycle, than there does not exist a shortest path between them. Thus we can find shortest path only if there does not exist any negative cycle.
2. We cannot use Dijkstra’s algorithm to find shortest path if the edge weights are negative.

