**Coin Row Problem :**Coin-rowproblem There is a row of n coins whose values are some positive integers c1, c2, . . . , cn, not necessarily distinct. The goal is to pick up the maximum amount of money subject to the constraint that no two coins adjacent in the initial row can be picked up.

Or in other words

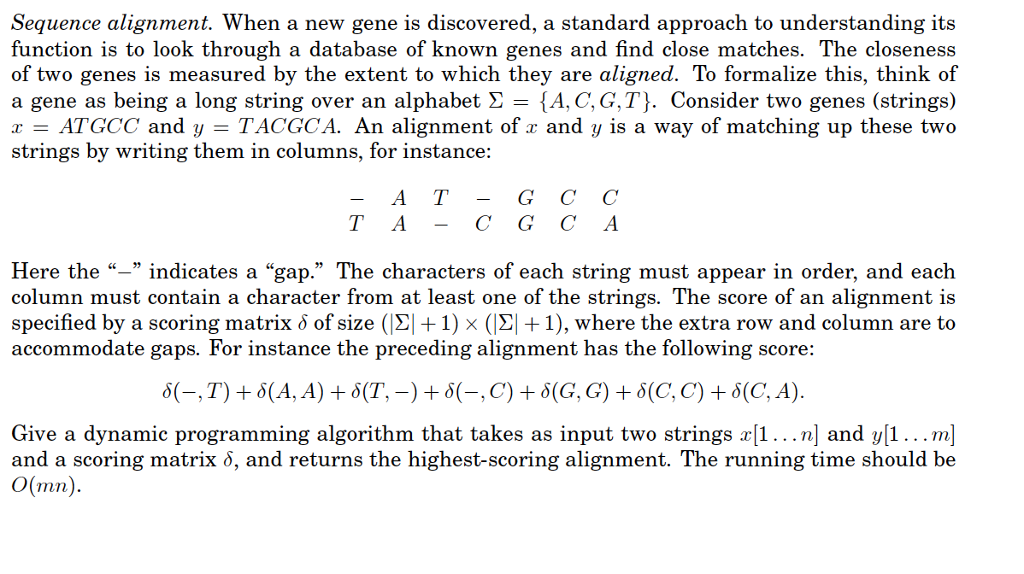
There is an integer array consisting positive numbers only. Find maximum possible sum of elements such that there are no 2 consecutive elements present in the sum.

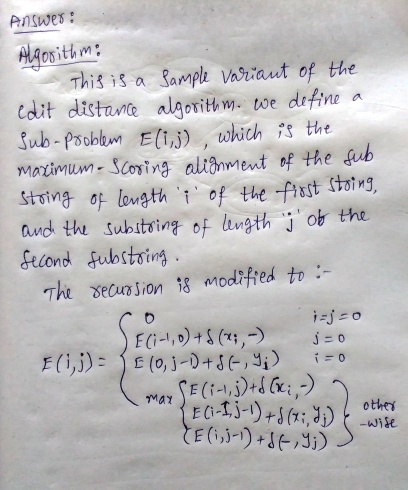
For ex: Coins[] = {5, 22, 26, 15, 4, 3,11}  
Max Sum = 22 + 15 + 11 = 48

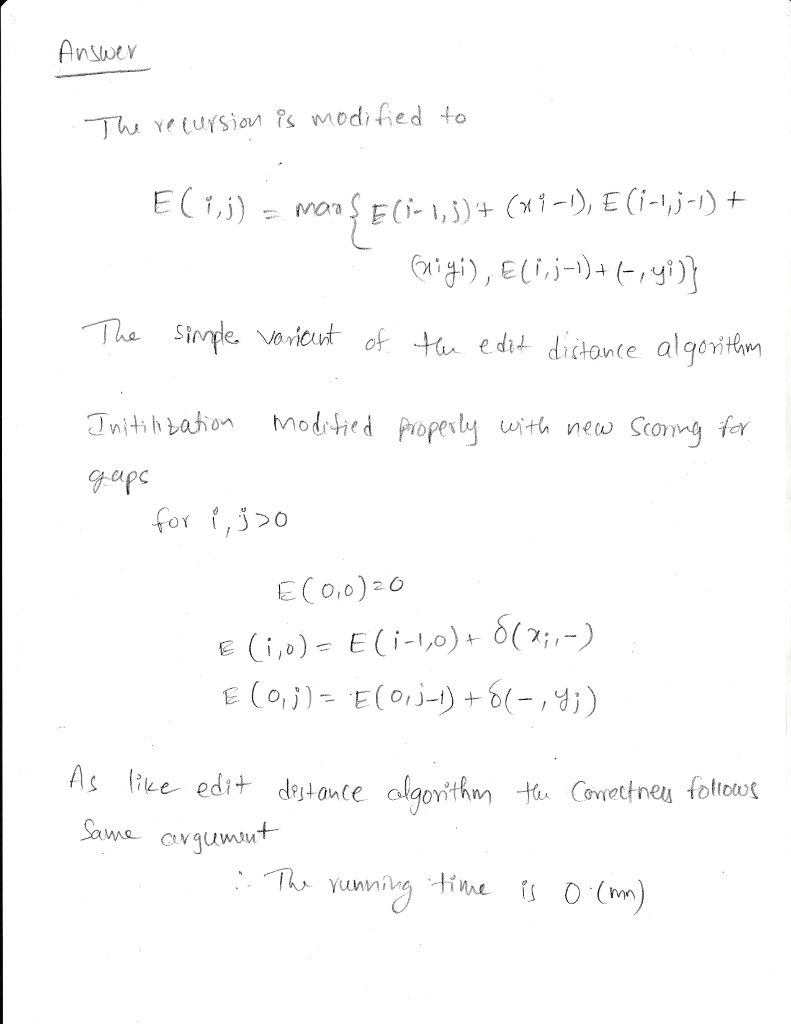
To solve this problem using Dynamic Programming first we will have to define recurrence relation.  
Let F[n] is the array which will contain the maximum sum at n for any given n. The recurrence relation will be.

F(n) = max{Coins[n] + F[n − 2], F[n − 1]} for n > 1,  
F(0) = 0, F(1) = Coins[1].

This is very easy to understand. While calculating F[n] we are taking maximum of coins[n]+the previous of preceding element and the previous element.  
For example F[2] = max{coins[0]+ F[2-2], F[2-1]} // No consecutive

Complexity: O(nm)

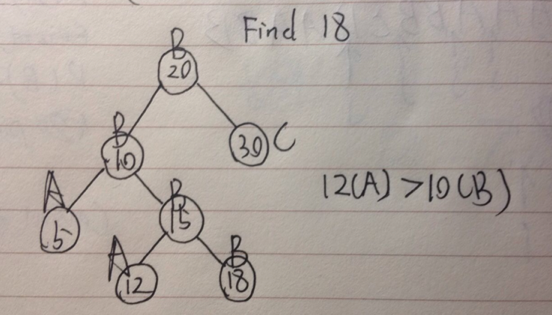




1. (CLRS 17.2-1) A sequence of stack operations is performed on a stack whose size never exceeds k. After every k operations, a copy of the entire stack is made for backup purposes. Show that the cost of n stack operations, including copying the stack, is O(n) by assigning suitable amortized costs to the various stack operations.

Solution: Assign the following amortized costs: Push 3 Pop 1 Multipop 1 Push uses one credit to pay for itself and saves one credit for future pops and one for copying the stack. Pop and Multipop pay for their operations using saved Push credits and save a credit for stack copying. After k operations, we have saved k credits exclusively for stack copying and can copy the stack for free. Since each operation costs at most O(1) amortized and the credits are nonnegative, the cost for n operations is

5)b) Professor Bunyan thinks he has discovered a remarkable property of binary search trees. Suppose that the search for key k in a binary search tree ends up in a leaf. Consider three sets: A, the keys to the left of the search path; B, the keys on the search path; and C, the keys to the right of the search path. Professor Bunyan claims that any three keys a∈A, b∈B, and c∈C must satisfy a ≤ b ≤ c. Give a smallest possible counterexample to the professor’s claim.



5)c) Describe a red-black tree on *n* keys that realizes the largest possible ratio of red internal nodes to black internal nodes. What is this ratio? What tree has the smallest possible ratio, and what is the ratio?

* The largest ratio is 2, each black node has two red children.
* The smallest ratio is 0