```swift
// Minimum change of amount given denominations
// Method: Dynamic Programming
// Space complexity: O(Amount)
// Time complexity: O(Amount * |{Denominations}|)
struct MoneyChanger {
    let denominations: [Int]

    func minChange(for amount: Int) -> [Int]? {
        guard amount > 0 else { return nil }

        // sentinel for this problem.
        // when compared, any value would be less than this
        let sentinel = amount + 1

        // answer to subproblem i amount
        // dp[i] = min count required to change amount i
        var dp = [Int](repeating: sentinel, count: amount + 1)

        // stores back pointer (like in a linked list) for best path for i
        // 0 is a sentinel as the problem is already solved after reaching 1
        var previousIndices = [Int](repeating: 0, count: amount + 1)

        // stores last denominations for best path
        // 0 is a sentinel as it is not a valid denomination
        // no need to check sentinel while back tracking so any value is fine
        var lastChanges = [Int](repeating: 0, count: amount + 1)

        // 0 change needed for amount 0
        dp[0] = 0

        // for each subproblem i amount
        for i in 1...amount {
            // try each denomination to be the last change
            for j in 0..<denominations.count {
                guard denominations[j] <= i else { continue }

                let currentChange = denominations[j]
                let previousIndex = i - currentChange
                let minChangeAfterDeductingDenomination = dp[previousIndex]
                let newMinChange = minChangeAfterDeductingDenomination + 1

                // update min if this is a new best
                if newMinChange < dp[i] {
                    dp[i] = newMinChange
                    previousIndices[i] = previousIndex
                    lastChanges[i] = currentChange
                }
            }
        }

        // minimum number of change required
        let minCount = dp[amount]
        guard minCount < sentinel else { return nil }

        // minimum change
        var change = [Int]()
        change.reserveCapacity(minCount)

        // back track and build the change
        var k = amount
        while k > 0 {
            change.append(lastChanges[k])
            k = previousIndices[k]
        }

        return change
    }
}

let moneyChanger = MoneyChanger(denominations: [1, 2, 5, 10, 20, 50, 100, 200, 500])
let amount = 1882
if let change = moneyChanger.minChange(for: amount) {
    print(change)
} else {
    print("impossible to change \(amount) with denominations \(moneyChanger.denominations)")
}
```