Pirates' Gold: Solution

Problem: You have stolen a number of gold pieces of various values

$$S = \{s_0, s_1, \ldots, s_{n-1}\}$$

and a claim c on the amount you stole. You want to determine the **minimum sum** of a subset of values of S whose value is **at least** as large as c.

Example:

S = {7, 10, 13, 4, 10, 24},
c = 25,
Answer: 27 (= 7 + 10 + 10 or alternately 10 + 13 + 4).

Ideas that Don't Work

Best-Fit: Add in the value that brings you closest to the claim.

Counter-example: S = {7, 10, 13, 4, 10, 24}, c = 25

24 + 4 = 28 (**too big!**)

First-Fit Increasing: Start with the smallest amount and start adding values until we reach a value as large as the claim.

Counter-example: S = {7, 10, 13, 4, 10, 24}, c = 25

4 + 7 + 10 + 10 = 31 (**too big!**)

Add and Prune: First-fit, but prune unneeded items.

Example: S = {7, 10, 13, 4, 10, 24}, c = 25

4 + 7 + 10 + 10 = 31; remove 4; 7 + 10 + 10 = 27 (**maybe this works?**)

Counter-example: S = {7, 10, 13, 4, 10, 24}, c = 23

4 + 7 + 10 + 10 = 31; remove 7; 4 + 10 + 10 = 24.

But 10 + 10 + 13 = 23, and this is better. (**No this doesn't work**)

Bottom line: We need something that is provably correct.

Ideas that Do Work

Brute Force: Enumerate all subsets of coins, compute each sum, and return the smallest value exceeding the claim.

This should be too slow, if we had generated a large enough test case.

Foolishly, we didn't.

Our Solution Structure

Approach: We will construct **all the possible sums** that can be generated from the first i coins, where i = 0, 1, 2, …, n. Then we will select the smallest sum that is at least as large as the claim.

Example: S = {7, 10, 13, 4, 10, 24}

Sum[0] = {0}	No coins yet.
Sum[1] = {0, 7}	We may either use 7 or not.
Sum[2] = {0, 7, 10, 17}	= {0, 7} ∪ ({0, 7} + 10)
Sum[3] = {0, 7, 10, 17, 13, 20, 23, 30}	= {0, 7, 10, 17} ∪ ({0, 7, 10, 17} + 13)

General Rule:

Sum[i] = Sum[i-1] ∪ (Sum[i-1] + s[i])

Implementation: We will represent Sum as a 2-dimensional array boolean array, where sum[i][j] = true if j is an element of Sum[i].

Final result: Return the smallest j ≥ claim, such that sum[n][j] = true.

Computing sum[i][j]:

For i = 0: sum[0][j] ← true if and only if j = 0.

For i ≥ 1: sum[i][j] ← sum[i-1][j] || sum[i-1][j – s[i]]

Pseudo-code

```
M ← some large enough value (e.g. claim + largest stolen value).
sum[0][0] ← true;                 // initialize row 0
for (j ← 1 to M) sum[0][j] ← false;

for (i ← 1 to n) {                // construct rest of table
    for (j ← 0 to M) {
            sum[i][j] ← sum[i-1][j] || sum[i-1][j - s[i]];
            // Note: not quite correct – may generate negative subscript
        }
    }
}
for (j ← claim to M) {            // determine return value
    if (sum[n][j]) return j;      // return smallest after claim
}
return -1;                        // no feasible solution
```