Min Abs Sum

with help of Peng Cao

codility
WE TEST CODERS

We show how the Codility Challenge codenamed Delta-2011 can be solved. You can still give it a try, but no certificate will be granted. The problem asks you to find the lowest absolute sum of elements of the given array or their negatives.

Since we can arbitrarily choose to take the element or its negative, we can simplify the problem and replace each number with its absolute value. Then the problem becomes dividing the numbers into two groups and making the difference between the sums of the two groups as small as possible. It is a classic dynamic programming problem.

Assume the sum of absolute values of all the numbers is S. We want to choose some of the numbers (absolute values) to make their sum as large as possible without exceeding $\frac{S}{2}$. Why? Let P be the sum of the first group, Q be the sum of the other group and $P < Q$. We have $P \leqslant \frac{S}{2} \leqslant Q$ and $Q + P = S$. The larger is P, the smaller is Q and the difference $Q - P$. Hence, the largest possible $P \leqslant \frac{S}{2}$ gives the optimal result. Let M be the maximal element in the given array A. We create an array dp of size S.

Slow solution $O(N^2 \cdot M)$

Let dp_i equal 1 if it is possible to achieve the sum of i using elements of A, and 0 otherwise. Initially $dp_i = 0$ for all of i (except $dp_0 = 1$). For every successive element in A we update the array taking this element into account. We simply go through all the cells, starting from the top, and if $dp_i = 1$ then we also set dp_{i+A_j} to 1. The direction in which array dp is processed is important, since each element of A can be used only once. After computing the array dp, P is the largest index such that $P \leqslant \frac{S}{2}$ and $dp_P = 1$.

1: Slow solution.

```
def slow_min_abs_sum(A):
    N = len(A)
    M = 0
    for i in xrange(N):
        A[i] = abs(A[i])
        M = max(A[i], M)
    S = sum(A)
    dp = [0] * (S + 1)
    dp[0] = 1
    for j in xrange(N):
        for i in xrange(S, -1, -1):
            if (dp[i] == 1) and (i + A[j] <= S):
                dp[i + A[j]] = 1
    result = S
```

```
15      for i in xrange(S // 2 + 1):
16          if dp[i] == 1:
17              result = min(result, S - 2 * i)
18      return result
```

The time complexity of the above solution is $O(N^2 \cdot M)$, since $S = O(N \cdot M)$.

Golden solution $O(N \cdot M^2)$

Notice that the range of numbers is quite small (maximum 100). Hence, there must be a lot of duplicated numbers. Let $count_i$ denote the number of occurrences of the value i. We can improve the previous solution by processing all occurrences of the same value at once. First we calculate values $count_i$. Then we create array dp such that:

- $dp_j = -1$ if we cannot get the sum j,
- $dp_j \geqslant 0$ if we can get sum j.

Initially, $dp_j = -1$ for all of j (except $dp_0 = 0$). Then we scan through all the values appearing in A; we consider all a such that $count_a > 0$.

For every such a we update dp that dp_j denotes how many values a remain (maximally) after achieving sum j. Note that if the previous value at $dp_j \geqslant 0$ then we can set $dp_j = count_a$ as no value a is needed to obtain the sum j. Otherwise we must obtain sum $j - a$ first and then use a number a to get sum j. In such a situation $dp_j = dp_{j-a} - 1$.

Using this algorithm, we can mark all the sum values and choose the best one (closest to half of S).

2: Golden solution.

```
1   def golden_min_abs_sum(A):
2       N = len(A)
3       M = 0
4       for i in xrange(N):
5           A[i] = abs(A[i])
6           M = max(A[i], M)
7       S = sum(A)
8       count = [0] * (M + 1)
9       for i in xrange(N):
10          count[A[i]] += 1
11      dp = [-1] * (S + 1)
12      dp[0] = 0
13      for a in xrange(1, M + 1):
14          if count[a] > 0:
15              for j in xrange(S):
16                  if dp[j] >= 0:
17                      dp[j] = count[a]
18                  elif (j >= a and dp[j - a] > 0):
19                      dp[j] = dp[j - a] - 1
20      result = S
21      for i in xrange(S // 2 + 1):
22          if dp[i] >= 0:
23              result = min(result, S - 2 * i)
24      return result
```

The time complexity of the above solution is $O(N \cdot M^2)$, where M is the maximal element, since $S = O(N \cdot M)$ and there are at most M different values in A.