CS29003 ALGORITHMS LABORATORY
Assignment No: 9
Last Date of Submission: 07–October–2015

Ms. Zyzzyva receives m items in her office at Fast Courier Transform Private Limited. Each item has a cost (or weight). She needs to put the items in cartons for shipping to a foreign country. She must put the items in the same sequence as she has received them. Moreover, the total cost of the items packed in each carton must not exceed a capacity bound C (this may be the weight limit of a carton or a restriction on total cost imposed by the customs department of the destination country). In order to reduce packing and shipping costs, Ms. Zyzzyva needs to minimize the number of cartons to pack all the m items without violating the sequence and capacity constraints. Let the item costs and C be integers. Treat C as a constant. For this assignment, take $C = 100$. Finally, assume that each item cost is $\leq C$ (otherwise, the problem has no solution).

We can restate Ms. Zyzzyva's problem as follows. Let $A = [a_0, a_1, \ldots, a_{m-1}]$ be an array of m integers. Decompose the array into a concatenation of k subarrays ($A = A_1A_2 \ldots A_k$) such that no A_i is empty, and k is as small as possible.

Part 1 In this part, assume that each a_i is positive. Write a function *packpos*() to solve Ms. Zyzzyva's problem in this case. Your function should implement a greedy algorithm, and run in O(m) time.

Part 2 Now, allow the array elements a_i to be negative, that is, A[] is now a mixture of positive and negative integers. (If you need a physical interpretation, think of a helium-filled balloon having "negative" weight or a discount coupon having negative cost.) Write a function *packposneg*() to solve the packing problem. Your function should run in O(m^2) time. Use dynamic programming. Write a comment before the function, clearly stating what recurrence relation you are using in your dynamic-programming algorithm.

Part 3 Finally, suppose that there are two arrays $A = [a_0, a_1, \ldots, a_{m-1}]$ and $B = [b_0, b_1, \ldots, b_{n-1}]$. For simplicity, assume again that all a_i and b_j are positive integers. These two arrays stand for two different types of items (one from multi-national companies, the other from ulti-national companies). Ms. Zyzzyva may mix items of two types in a carton. But the total cost of the items in each carton must be $\leq C$, and the items of both types must be packed in the sequence they appear in the respective arrays. In other words, she needs a simultaneous decomposition of the two arrays as $A = A_1A_2 \ldots A_k$ and $B = B_1B_2 \ldots B_k$ (k is the same for both the arrays) such that for each i, either A_i or B_i (or both) is/are non-empty, and the sum of the elements of A_i and B_i is $\leq C$. The goal is to make k as small as possible. Write an O(mn)-time function *packtwo*() to solve the packing problem for two arrays. This function too should take a dynamic-programming approach. Write a comment before the function, clearly stating what recurrence relation you are using in your dynamic-programming algorithm.

Write a *main*() function to do the following:

- Read m and an array A of m positive integers. Call the function of Part 1 to optimally pack the items in A. Print which items are packed in each carton. Do not rewrite A. We need it again at the end.
- Read a second array A' of a mixture of m positive and negative integers. Call the function of Part 2 to optimally pack the items in A'. Print which items are packed in each carton.
- Read n and an array B of n positive integers. Call the function of Part 3 to optimally pack the items in A and B. Print which items are packed in each carton.

Submit a single C/C++ file solving all the parts. Do not use global/static variables.

Sample Output

```
m = 40

+++ Input array with positive entries
      14  15   6   5   5  15  12  13  13  12   6  18   8  19   7  14
      20  19  13  14   1  18  15  13   9  19  17   5   1   6   3  15
      20  20  11   5   6   2  17  18

+++ Packing items of positive costs
    ...

    Carton  3: Packing A[16 ... 22]: Total cost = 100
    ...

*** Total number of cartons needed = 5

+++ Input array with positive and negative entries
      -4  -2  -6  11  -9  16  11   7  17   1   5  13  17   1   5  -2
       9  13  18   1  12  16  16  14  16  -7   1  19  10  -7   2  16
      13  -8  -5  -3  16  19  -2  -6

+++ Packing items of both positive and negative costs
    ...

    Carton  3: Packing A[27 ... 39]: Total cost = 64
*** Total number of cartons needed = 3

n = 30

+++ Input array with positive entries
       9  18   2   1  10   2  19  18   3  10  16   9   3  16   4   5
      20   3   4   2  18  11  14  15  18  14  14  20   7  17

+++ Packing two types of items with positive costs
    Carton  1: Packing A[0  ...  8]              : Total cost = 98
    Carton  2: Packing A[9  ... 14] B[0  ...  3] : Total cost = 100
    Carton  3: Packing A[15 ... 21]              : Total cost = 99
    Carton  4: Packing A[22 ... 24] B[4  ...  9] : Total cost = 99
    Carton  5: Packing A[25 ... 25] B[10 ... 18] : Total cost = 99
    Carton  6: Packing A[26 ... 27] B[19 ... 24] : Total cost = 100
    Carton  7: Packing A[28 ... 32] B[25 ... 28] : Total cost = 100
    Carton  8: Packing A[33 ... 39] B[29 ... 29] : Total cost = 96
*** Total number of cartons needed = 8
```

A zyzzyva (Source: Youtube images)