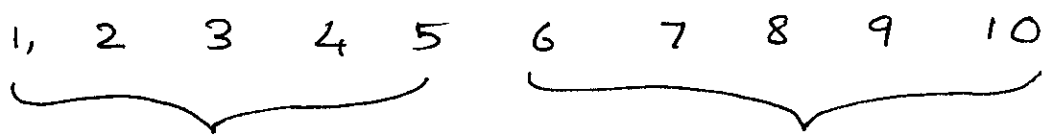
A Segmentation Problem.

Suppose that we have a sequence ~~t...n~~ ~~of~~ 1, 2, 3, ..., n of n elements, and a function that associates a cost $c[i, j]$ corresponding to each subsequence $i, i+1, \ldots, j-1, j$. We assume that $c[i,j]$ is a non-negative integer for each i and j.

The goal is to break up the input sequence into chunks so that the sum of the costs of the chunks is minimized.

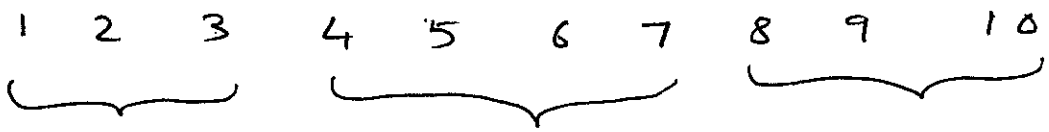
Let us clarify this by an example. Suppose that $n = 10$.



Then one possible segmentation, i.e, way of breaking up the sequence into chunks, is shown above.

It has a cost $c[1,5] + c[6,10]$.

Another possible segmentation is:



It has a cost $c[1,3] + c[4,7] + c[8,10]$.

Which one has smaller cost? That depends on the $c[\cdot,]$ function that's part of the input.

The overall goal is to find the segmentation of smallest cost.

This problem is related to Section 6.3 (Segmented Least Squares), Solved Exercise 2 in Chapter 6, and several exercises in chapter 6.

As always, we begin our attempt at solution by recursive thinking: How to express the optimal solution to the problem in terms of optimal solutions to smaller problems?

Suppose that the last chunk in the optimal solution is the chunk $j, j+1, \ldots, n$. What can we say about the rest of the chunks in the optimal solution?

They constitute an optimal segmentation of $1, 2, \ldots, j-1$.

So, assuming that the last chunk in an optimal solution is $j, \ldots, n$,

the cost of optimal segmentation of $1, 2, \ldots, n$

$=$ cost of optimal segmentation of $1, 2, \ldots, j-1$ $+$ $c[j, n]$.

Lets introduce some notation: $Opt(i)$ denotes cost of optimal segmentation of $1, 2, \ldots, i$.

So we have $Opt(n) = Opt(j-1) + c[j, n]$

assuming last chunk in an optimal solution for $1, 2, \ldots, n$ is $j, j+1, \ldots n$.

We don't know this j. But we can assert:

$$Opt(n) = \min_{1 \le j \le n} opt(j-1) + c[j, n]$$

We define $Opt(0) = 0$.

More generally, we have, for $1 \le i \le n$:

$$Opt(i) = \min_{1 \le j \le i} Opt(j-1) + c[j,i]$$

This recurrence suggests that we should compute $Opt(0), Opt(1), Opt(2), \ldots, Opt(n)$ in that order. We'll initialize an array $M[0..n]$, and use $M[i]$ to store $Opt(i)$.

```
M[0] ← 0
For i ← 1 to n
    M[i] ← ∞
    For j ← 1 to i
        if (M[i] > M[j-1] + c[j,i])
            M[i] ← M[j-1] + c[j,i]
        endif
    endfor
endfor
```

Note that $M[n]$ eventually has the cost of best segmentation of $1, 2, \ldots, n$.