(cont). designing dynamic prog. algo

→ notice there is a sharing of subproblems going on which prevents us from dividing the problem into completely independent subproblems

→ we can use the equation for $N_{i,j}$ to derive an algorithm by computing $N_{i,j}$ values in a bottom-up fashion, and storing the intermediate solutions in a table of values

→ we can build $N_{i,j}$ values up from previously computed values until we can finally compute value of $N_{0,n-1}$

```
def matrix-chain(d):
  n = len(d)-1                          // number of matrices
  N = [[0] * n for i in range(n)]       // init n-by-n result to 0
  for b in range(1,n):                  // num of products in subchain
    for i in range(n-b):                // start of subchain
      j = i+b                           // end of subchain
      N[i][j] = min(N[i][k] + N[k+1][j] + d[i] * d[k+1] * d[j+1]  for k in range(i,j))
  return N
```

DNA/Text Seq. Alignment/LCS

→ given string $X = x_0 x_1 x_2 \cdots x_{n-1}$, a subsequence of X is any string of form $x_{i_1} x_{i_2} \cdots x_{i_k}$, where $i_j < i_{j+1}$

→ the DNA and text similarity problem addressed here is longest common subseq. problem (LCS)

→ Components of Dynamic Prog. Soln

→ Simple Subproblems: must be some way of repeatedly breaking the global opt. problem into subproblems. There should be a way to parameterize subproblems with just a few indices.

→ Subprob Optimization: optimal solution to global problem must be a composition of optimal subproblems

→ Subprob Overlap: optimal solutions to unrelated subproblems can contain subproblems in common