Dynamic Programming - II

Sequence Alignment

Sequence Alignment

Input : Pair of strings X, Y ; |X| = m; |Y| = n

gap penalty δ , mismatch cost $\{\alpha_{pq} | p, q \in \Sigma\}$

Output : Minimum cost of alignment between X and Y

cost of alignment - sum of gap penalties and mismatch costs

Sequence Alignment

OPT[i,j] : cost of alignment of X[1,…,i] and Y[1,……,j]

Sequence Alignment

OPT[i,j] : cost of alignment of X[1,…,i] and Y[1,……,j]

Options :

1. X [i] is matched with Y[j]
2. X [i] is not matched
3. Y [j] is not matched

Sequence Alignment

OPT[i,j] =

$$\min(OPT[i-1,j]+\delta,$$

$$OPT[i,j-1]+\delta,$$

$$\alpha_{X[i]Y[j]}+OPT[i-1,j-1])$$

Sequence Alignment

Correctness of recurrence:

Sequence Alignment

Bottom-up implementation :

Sequence Alignment

Constructing the actual solution :

Sequence Alignment

Running Time :

0-1 Knapsack Problem

0-1 Knapsack Problem

Input : $\{w_1, w_2 \ldots, w_n\}$, W

Output : $S \subseteq [n]$, $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} w_i$ is maximised

0-1 Knapsack Problem

Sub-problem : OPT(i, w) - returns the value of the optimal solution using a subset of items $\{1,\ldots,i\}$ with maximum allowed weight w.

0-1 Knapsack Problem

Sub-problem : OPT(i, w) - returns the value of the optimal solution using a subset of items $\{1,\ldots,i\}$ with maximum allowed weight w.

OPT(n,W) - desired solution

0-1 Knapsack Problem

$$OPT(i,w) = max(OPT(i-1,w), w_i + OPT(i-1,w-w_i))$$

0-1 Knapsack Problem

Proof of correctness :

0-1 Knapsack Problem

Bottom-up implementation :

0-1 Knapsack Problem

Running Time :

Longest Palindromic Subsequence

Input : An array A[1,……,n]

Output : Longest palindromic subsequence of A

Longest Palindromic Subsequence

Input : An array A[1,……,n]

Output : Longest palindromic subsequence of A

Sub-problem ?

Longest Palindromic Subsequence

Input : An array A[1,……,n]

Output : Longest palindromic subsequence of A

Sub-problem : Pal[i,j] - longest palindromic subsequence of A[i,i+1,….,j]

Longest Palindromic Subsequence

Input : An array A[1,……,n]

Output : Longest palindromic subsequence of A

Sub-problem : Pal[i,j] - longest palindromic subsequence of A[i,i+1,….,j]

Base case?

Longest Palindromic Subsequence

Input : An array A[1,……,n]

Output : Longest palindromic subsequence of A

Sub-problem : Pal[i,j] - longest palindromic subsequence of A[i,i+1,….,j]

Guess ?

Longest Palindromic Subsequence

Pal[i,j] = 2+ Pal[i+1, j-1] if A[i] = A[j]

= max(Pal[i+1, j], Pal[i, j-1] otherwise

Longest Palindromic Subsequence

Proof of correctness :

Longest Palindromic Subsequence

Bottom-Up Implementation :

Longest Palindromic Subsequence

Running Time :