Longest Common Subsequence

– Motivated by problem of whether two proteins are similar.
– A subsequence of a sequence $a[1]a[2]\ldots a[n]$ is a sequence of the form $a[i_1]a[i_2]\ldots a[i_k]$ such that $1 \leq i_1 < i_2 < \ldots < i_k \leq n$.

Example: $ACAE$, AB, BA, AE, E are subsequences of $ABCADE$.
DA is not a subsequence of $ABCADE$.

– Longest common subsequence of two sequences ABC and BDC is BC.

Input: $x[1]x[2]\ldots x[n]$ and $y[1]y[2]\ldots y[m]$
Find the longest common subsequence of the above two sequences (strings).

If $x[n] = y[m]$, then the longest common subsequence is:
(the longest common subsequence of
$x[1]\ldots x[n-1]$ and $y[1]\ldots y[m-1]$)
followed by $x[n]$

If $x[n] \neq y[m]$, then the longest common subsequence is:
(the longest common subsequence of $x[1]\ldots x[n-1]$ and $y[1]\ldots y[m]$)
or
(the longest common subsequence of $x[1]\ldots x[n]$ and $y[1]\ldots y[m-1]$)

$F(i, j)$ length of the longest common subsequence of $x[1]x[2] \ldots x[i]$
and $y[1]y[2] \ldots y[j]$,
where $0 \leq i \leq n$ and $0 \leq j \leq m$
Base Case: $F[i, j] = 0$, for $i = 0$ or $j = 0$
Solve the recurrence equation given earlier to obtain other $F[i, j]$.

$LCS(x[1] \ldots x[n], y[1] \ldots y[m])$

For $i = 0$ to n { $F(i, 0) = 0$ }

For $j = 0$ to m { $F(0, j) = 0$ }

For $i = 1$ to n {

For $j = 1$ to m {

If $x[i] = y[j]$, then $F(i, j) = 1 + F(i - 1, j - 1)$.

If $x[i] \neq y[j]$, then $F(i, j) = max(F(i - 1, j), F(i, j - 1))$.

}

}

End

Complexity: $O(mn)$

LCSP($x[1] \ldots x[n], n, m, F$)

If $F(n, m) = 0$, then return.

If $F(n, m) = 1 + F(n - 1, m - 1)$, then return $LCSP(x, n - 1, m - 1, F) \cdot x(n)$

If $F(n, m) = F(n, m - 1)$, then return $LCSP(x, n, m - 1, F)$

If $F(n, m) = F(n - 1, m)$, then return $LCSP(x, n - 1, m, F)$

End

0/1 Knapsack problem

– Some objects $O_1, O_2, \ldots, O_n$
– Their weights $W_1, W_2, \ldots, W_n$ (assumed to be integral values)
– Their values $V_1, V_2, \ldots, V_n$
– Capacity C
– To find a set $S \subseteq \{1, 2, \ldots, n\}$ such that
$\Sigma_{i \in S} W_i \leq C$ (capacity constraint)
and
$\Sigma_{i \in S} V_i$ is maximsed

Note that here we cannot use fractional items! We either take the whole or nothing of each item.

Let $F(C, j)$ denote the maximum value one can obtain using capacity C such that objects chosen are only from $O_1, \ldots, O_j$.
Then, $F(C, 0) = 0$.
If $W_j \leq C$, then $F(C, j) = max(F(C, j-1), F(C-W_j, j-1)+V_j)$.
If $W_j > C$, then $F(C, j) = F(C, j-1)$.

For $s = 0$ to C $\{F(s, 0) = 0\}$.

    For $s = 1$ to C {

        For $j = 1$ to n {

            If $W_j \leq s$, then

                $F(s, j) = max(F(s, j-1), F(s - W_j, j-1) + V_j)$;

            Else $F(s, j) = F(s, j-1)$

        }

    }

Complexity: $O(C * n)$

For $s = 0$ to C $\{F(s,0) = 0; Used(s,0) = false\}$.

For $s = 1$ to C {

For $j = 1$ to n {

If $W_j \leq s$, then {

$F(s,j) = max(F(s,j-1), F(s-W_j, j-1)+V_j)$;

If $F(s,j) = F(s,j-1)$, then $Used(s,j) = false$

Else $Used(s,j) = true$ }

Else { $F(s,j) = F(s,j-1)$; $Used(s,j) = false$ }

}

}

$Left = C$

For $j = n$ down to 1 do {

    If $Used(Left, j) = true$, then {

        Print("Pick item" j); $Left = Left - W_j$;

}

All pairs shortest path algorithm.

Suppose A is the adjacency matrix of a weighted graph, that is, $A[i, j]$ gives the weight of the edge (i, j).
Here the vertices are numbered 1 to n.
Here we assume that $A[i, j]$ is non-negative.
If edge does not exist then the weight is taken to be ∞.
We want to find shortest path between all pairs of vertices.

Floyd's algorithm
$D_k[i, j]$ denote the length of the shortest path from i to j where the intermediate vertices (except for the end vertices i and j) are all $\leq k$.

$D_0[i, j] = A[i, j]$.
Here we assume $A[i, i] = 0$ for all i.
If initially not so, then update $A[i, i]$ to be made in this fashion.

To find, $D_k[i, j]$, for $1 \leq k \leq n$:
$D_k[i, j] = min(D_{k-1}[i, j], D_{k-1}[i, k] + D_{k-1}[k, j])$.

In the algorithm, $next[i, j]$ denotes the vertex which appears just after i in the shortest (known) path from i to j.

Note: $D_k[i, k] = D_{k-1}[i, k]$ and $D_k[k, j] = D_{k-1}[k, j]$.
So, we can do the computation in place!
(using D itself to compute $D_0, D_1, D_2, \ldots$).

```
For i = 1 to n {
For j = 1 to n {
  next[i, j] = j; D[i, j] = A[i, j]
} }
For i = 1 to n { D[i, i] = 0 }
For k = 1 to n {
    For i = 1 to n {
    For j = 1 to n {
        If D[i, j] > D[i, k] + D[k, j], then {
            D[i, j] = D[i, k] + D[k, j]
            next[i, j] = next[i, k]
            }
    }
    }
}
```

$Path(i, j)$

$current = i;$

Print(i);

While $current \neq j$ {

    $current = next(current, j);$

    Print($current$)

}