6.8

\[ x = x_1 \ldots x_n \]
\[ y = y_1 \ldots y_n \]

LLCS(x, y) = length of longest common substring of x and y.

\[ LLCS'(x_i, y_j) = \begin{cases} 
LLCS(x_{i-1}, y_{j-1}) + 1 & \text{if } x_i = y_j \\
0 & \text{otherwise} 
\end{cases} \]

LLCS' is defined as the length of the longest common substring ending at xi and yj.

\[ LLCS'(x_1, y_1, \epsilon) = 0 \quad \epsilon \text{ is an empty string} \]

\[ LLCS'(\epsilon, x_1, y_1) = 0 \]

\[ LLCS(x, y) = \max_{i, j} LLCS'(x_i, y_j) \]
0.15) Let $A(i, j)$ $(i, j \leq n)$ represent the prob. that $A$ wins if $i$ goes and $B$ wins if $j$ goes.

\[
A(n, j) = 1 \quad \text{if } j \leq n
\]

\[
A(i, n) = 0 \quad \text{for all } i < n
\]

Solve the subproblems in decreasing order of $i + j$. 

\[
A(i, j) = \frac{1}{2} \left( A(i, j+1) + A(i+1, j) \right)
\]
6.17.

\[ \text{change}(0) = \text{true} \]
\[ \text{change}(w) = \text{false} \quad \text{for all } 1 \leq w \leq u \]

\[ \text{for } i = 1 \text{ to } u \]
\[ \text{change}(i) = \bigvee \text{change}(w) \quad \text{if } \exists i \leq w \]

\[ \text{for } i = 1 \text{ to } u \]
\[ \text{change}(i) = \text{change}(i) \lor \text{change}(i-xj) \quad \text{if } xj \leq i \\
\lor j \]

6.18.

\[ \text{change}(0, i) = \text{true} \quad \text{for all } i = 1 \text{ to } n \]
\[ \text{change}(w, i) = \text{false} \quad \text{for all } w = 1 \text{ to } u \]

\[ \text{for } i = 1 \text{ to } n \]
\[ \text{for } j = 1 \text{ to } w \]
\[ \quad \text{if } xj \leq j \]
\[ \quad \text{change}(j, i) = \text{change}(j, i-1) \lor \text{change}(j, i-1) \]
\[ \text{else change}(j, i) = \text{change}(j, i-1) \]
6.19. Make a new set \( x_1, x_2, \ldots, x_n = y_1, \ldots, y_{3n} \)

\[ \text{Change}(0, i) = \text{true} \quad i = 0 \text{ to } 3n \]

\[ \text{Change}(w, i) = \text{false} \quad w = 1 \text{ to } 3n \]

\[ \text{for } i = 1 \text{ to } 3n \]

\[ \text{for } j = 1 \text{ to } w \]

\[ \text{if } y_i \leq j \]

\[ \text{Change}(j, i) = \text{Change}(j, i - 1) \lor \text{Change}(j - y_i, i - 1) \]

\[ \text{else} \]

\[ \text{Change}(w, i) = \text{Change}(j, i - 1) \]
\begin{align*}
\text{cost}(i) &= \min \left( C + \text{price}(j) + \text{cost}(i-j) \right) & \text{if } j \leq i \\
\text{cost}(1) &= C + \text{price}(1) \end{align*}
Tutorial notes for the Viterbi Algorithm problem

This problem is stated in CLR 15-5. In practice, Viterbi algorithm is used for speech recognition.

You are given a directed graph $G = (V, E)$ in which every edge is labeled by a letter from a finite alphabet $\Sigma$, that is there is a function $\sigma(u, v)$ that gives a label for the edge $(u, v)$ for every pair $(u, v) \in E$. You are also given a distinguished vertex $v_0$ and a string $s = l_1, \ldots, l_k$ of symbols from $\Sigma$. Design a DP algorithm to check whether there is a path starting at $v_0$ labeled with $s$.

For example, let $\Sigma = \{a, b, c, d, e\}$, $v_0 = 1$ and $s = abc$ and consider the following graph. It has a path $\{1, 5, 4, 2\}$ labeled with $s$.

Now we want to design a dynamic programming algorithm for finding whether there is a path from $v_0$ labeled with $s$.

**Step 1:** Let $0 \leq i \leq k$ and $1 \leq v \leq n$ (that is, $i$ indexes the letters of $s$ and $v$ the vertices). Then define $A(i, v) = 1$ if there is a path from $v_0$ to $v$ labeled with the prefix of length $i$ of $s$, and $A(i, v) = 0$ otherwise. The final answer is “yes” if there is $v$ such that $A(k, v) = 1$.

**Step 2:** Initialize with $A(0, v_0) = 1$ and for $v \neq v_0 A(0, v) = 0$. Now the recurrence becomes:

$$A(i, v) = \max\{A(i-1, u)|(u, v) \in E \text{ and } \sigma(u, v) = l_i\}$$

That is, to set the value of $A(i, v)$ we check whether there is an edge labeled with $l_i$ leading to $A(i, v)$ from a vertex that was reached in the previous step. If there is $u$ such that $A(i-1, u) = 1$, then it is possible to arrive to $u$ by a path labeled with the prefix of length $i-1$ of $s$; the condition $(u, v) \in E$ states that $(u, v)$ is an edge and the condition $\sigma(u, v) = l_i$ states that this edge is labeled with $l_i$. Note that there can be several paths leading to a vertex, but we are only interested in an existence of a path, not their number. This is why the recurrence has max in it: if there is a path, $A(i, v)$ becomes 1, otherwise $A(i, v) = 0$.

**Step 3:** Skipped. As usual, just compute the array and then find a non-zero value in the last row. If succeed, output “path exists”, otherwise output “no path”. For this problem, it is simplest to encode the graph by its adjacency matrix with labels instead of 0/1 (that is, if a matrix $M$ encodes the graph, $M(u, v) = \sigma(u, v)$ if there is an edge $(u, v)$, and some special symbol not in $\Sigma$ if there is no edge $(u, v)$ in the graph.)

**Step 4:** Suppose now that there is a path, and we want to output the vertices that constitute the path. The idea is just to retrace the steps. That is, starting with a non-zero entry in the last row, find an edge with the correct label that leads to a vertex $v$ for which $A(i-1, v)$. The most straightforward algorithm outputs vertices in reverse order; make a recursive procedure in style of PrintOpt in the notes for scheduling to output the vertices in the correct order.