(a) Denote by $\delta(i,j,h)$ the length of the shortest path from $i$ to $j$ by a path involving only the nodes $i, j$ and $0, \ldots, h-1$, and $\infty$ if no such path exists.

Claim: When the "h-loop" is executed with value $h$, $C[i,j]$ contains $\delta(i,j,h)$

Proof: For $h=0$, this is secured by the first for-loops which copy in direct links.

Suppose true for $h$ and the "h-loop" has just been executed with value $h$ (and so is about to be executed with value $h+1$).

Case (i) the shortest path from $i$ to $j$ via $0, \ldots, h$ passes through node $h$, and this is shorter than any path from $i$ to $j$ via $0, \ldots, h-1$. Then $\delta(i,j,h+1)\leq\delta(i,h,h+1)+\delta(h,h,h+1) < \delta(i,j,h)$

But by inductive hypothesis, on this iteration of loop, $C[i,h]+C[h,j]<C[i,j]$ and so $C[i,j]$ will be updated to $\delta(i,j,h+1)$.

Case (ii) $\delta(i,j,h)\leq\delta(i,j,h+1)$. In that case, $C[i,j]$ does not change.

At the end of the algorithm, $C[i,j]$ contains $\delta(i,i,h)$ as required.
6) Set up an array \( N[i, j] \) which records
the first node vertex on an optimal path from \( i \) to \( j \).

Start by initializing \( N[i, j] = j \) whenever \( A[i, j] \neq \infty \).

Then, replace \( C[i, j] \leftarrow \min (C[i, j], C[i, h] + C[h, j]) \)
by

\[ \text{if } C[i, h] + C[h, j] < C[i, j] \]

then

\[ C[i, j] \leftarrow C[i, h] + C[h, j] \]

\[ N[i, j] \leftarrow N[i, h] \]

Then we can recover such a path from \( N \) by
jumping from vertex to vertex.

c) Precede code snippet \( \text{\( \square \)} \) by

\[ \text{if } C[i, h] + C[h, j] \leq C[i, j] \]

then

add \( N[i, h] \) to \( N[i, j] \)

where \( N[i, j] \) now records a set of possible
next steps. All best paths are then recoverable
from \( N \) by following any of the suggested links.
2 a) Record $x$ as an array over $V_i \times x_{[0]} \ldots x_{[n]}$

We fill in a "chart" $Ch[i,j]$ recording the set of non-terminals which $x_{[i]} \ldots x_{[j]}$
can be assigned using the grammar.

Code:
- Initialize $Ch[i,j]$ to be $\emptyset$ for all $i,j$.
- For $i = 0$ to $n - 1$
  - For all $p \in P$ of form $A \to p x_{[i]}$
    - Add $A$ to $Ch[i,i]$
- For all $h = 1$ to $n - 1$
  - For all $i = 0$ to $n - h$
    - For all $j = i$ to $i + h - 1$
      - For all $p \in P$ of the form $A \to BC$
        - If $B \in Ch[i,j]$ and $C \in Ch[i+h,i+h]$
          - Then
            - Add $A$ to $Ch[i,i+h]$
- If $S \in Ch[0,n-1]$
  - Then
    - Return $Y$
- Else
  - Return $N$

6) Instead of adding $A$ to $Ch[i,i+h]$, add

$\langle A, B, C, j \rangle$

recording the producing production rule chosen
and the place where it splits the string

$x_{[i]} \ldots x_{[i+h]}$
3a) \[ 2a_0, a_{i+1}, a_{i+2}, \ldots, a_{n-1}, a_n \]

b) \[ 2a_n, a_{n-1}, a_{n-2}, a_{n-3}, \ldots, a_{i+1} \]

c) Let \( \text{Ch}[i,j] \) (1 \( i \leq n \), 1 \( j \leq n \)) and \( \text{N}[i,j] \) (1 \( i \leq n \), 1 \( j \leq n \)) be arrays. \( \text{Ch}[i,j] \) will store the minimum number of operations required to calculate \( A_i, \ldots, A_j \) and \( \text{N}[i,j] \) will record the best split \( k \):

\[ A_{i-1} \ldots A_{k-1} = (A_i \ldots A_{k-1}) \cdot (A_{k+1} \ldots A_{j+1}) \]

For \( i = 1 \) to \( n-1 \)

\( \text{Ch}[i, i+1] = 2a_{i+1} \)

\( \text{N}[i, i+1] = i \)

For \( h = 2 \) to \( n-1 \)

For \( i = 1 \) to \( n-h \)

For \( j = i + h - 1 \)

If \( \text{Ch}[i,j] > \text{Ch}[i,j] + \text{Ch}[j+1,n] + 2a_{j+1} \)

Then

\( \text{Ch}[i,j] \leftarrow \text{Ch}[i,j] + \text{Ch}[j+1,n] + 2a_{j+1} \)

\( \text{N}[i,j] \leftarrow j \)
c) If $n = 2$ then aa has at least $2^\frac{2^2 - 1}{2} = 1$ parse, namely

\[
\begin{array}{c}
A \\
/ \ \\
A - a \\
/ \ \\
A - a - a \\
/ \ \\
A - a - a - a \\
\end{array}
\]

If $n = 3$ then aaa has at least $2^\frac{3^2 - 1}{2} = \sqrt{2}$ parses, namely

\[
\begin{array}{c}
A \\
/ \ \\
A - a \\
/ \ \\
A - a - a \\
/ \ \\
A - a - a - a \\
\end{array}
\]

Suppose true for $n$. We show it is true for $n+2$. But $a^{n+2} = a^n a^n$, which can be parsed

\[
\begin{array}{c}
A \\
/ \ \\
A - a \\
/ \ \\
A - a - a \\
/ \ \\
A - a - a - a \\
\end{array}
\]

where $T$ is any parse tree for $a^n$. These are all different so the number of parse-trees for $a^{n+2}$ is at least twice as many as for $a^n$.

The algorithm in 6 still runs in polynomial time, because it stores these parse trees implicitly.