Lösningar

1. Konstruera matriskedjemultiplikation

Det enda vi behöver ändra i algoritmen är att göra tilldelningen
$breakpoint[i, j] \leftarrow k$
sist på sista raden, dvs efter att värdet på $M[i, j]$ uppdaterats.

PrintWithParentheses$(a, b, breakpoint[1..n, 1..n]) =$
    if $a = b$ **then** Write('A')
    else
        Write('(')
        PrintWithParentheses$(a, breakpoint[a, b], breakpoint)$
        PrintWithParentheses$(breakpoint[a, b] + 1, b, breakpoint)$
        Write(')')

2. Bevisa matriskedjemultiplikation

En lämplig invariant för slingan **for** $len \leftarrow 1$ **to** $n-1$ säger att M är korrekt beräknad (dvs enligt rekursionen) upp till avstånd $len-1$ till höger om diagonalen.

När man går in i slingan är $len = 1$ och invarianten säger då att M är korrekt beräknad upp till avstånd 0 från diagonalen, dvs bara själva diagonalen. Eftersom diagonalen enligt basfallet bara ska innehålla nollor och första satsen i funktionen sätter alla diagonalelement till noll så är invarianten uppfylld när man går in i slingan.

När man går ur slingan är $len = n$ och invarianten säger att M är korrekt beräknad upp till avstånd $n-1$ till höger om diagonalen, dvs alla element till höger om diagonalen är korrekt beräknade. Speciellt är $M[1,n]$ korrekt beräknat, så när funktionen returnerar $M[1,n]$ så är det det minimala antalet multiplikationer som behövs för att räkna ut matrisprodukten $A_1 \cdots A_n$.

En lämplig invariant för den inre slingan **for** $i \leftarrow 1$ **to** $n - len$ behöver säga att $M[a,b]$ är beräknad enligt rekursionen för $1 \leq a \leq n$, $a \leq b \leq \min(a+len-1, n)$ och $M[a, a+len]$ är beräknad enligt rekursionen för $1 \leq a < i$.

När man går in i slingan är $i = 1$ och invarianten säger då bara att $M[a,b]$ är beräknad enligt rekursionen för $1 \leq a \leq n$, $a \leq b \leq \min(a+len-1, n)$, vilket är samma sak som att säga att M är korrekt beräknad upp till avstånd $len-1$ från diagonalen, alltså det som gäller enligt den yttre invarianten.

När man går ur slingan är $i = n - len + 1$ och invarianten säger att $M[a,b]$ är beräknad enligt rekursionen för $1 \leq a \leq n$, $a \leq b \leq \min(a+len-1, n)$ och $M[a, a+len]$ är beräknad enligt rekursionen för $1 \leq a < n - len + 1$, vilket är samma sak som att säga att M är korrekt beräknad upp till avstånd len från diagonalen, alltså det som ska gälla för den yttre invarianten i slutet på den yttre slingan.

Så om den inre invarianten gäller (för den inre slingan) så betyder det att den yttre invarianten är korrekt (för den yttre slingan), och då är algoritmen korrekt.

Det enda som återstår i korrekthetsbeviset är att visa att den inre slingans invariant är korrekt, dvs att den gäller i början av varje varv i slingan.

Detta är ganska lätt att se eftersom det bara är elementet $M[i, i+len]$ i M som påverkas inuti slingan, och det elementet sätts korrekt enligt rekursionen med hjälp av en standardminimiberäkning över alla k mellan i och $i+len-1$ där dom andra elementen från M som används i beräkningen redan är korrekt beräknade enligt invarianten.

3. Implementera hållbart fiske

Lämplig beräkningsordning är att beräkna matrisen F radvis (från vänster till höger i varje rad) uppifrån och ner. Pseudokoden blir som följer:

```
for i ← 1 to n do F[i, 0] ← 0
F[1, 1] ← k_{1,1}
for j ← 2 to k do F[1, j] ← −∞
for i ← 2 to n do
    for j ← 1 to k do
        if F[i − 1, j] > F[i − 1, j − 1] + k_{i,j} then F[i, j] ← F[i − 1, j]
        else F[i, j] ← F[i − 1, j − 1] + k_{i,j}
kmax ← k
for j ← k − 1 downto 1 do
    if F[n, j] > F[n, kmax] then kmax ← j
S ← new Stack
j ← kmax
for i ← n downto 2 do
    if F[i − 1, j] < F[i − 1, j − 1] + k_{i,j} then
       S.push(i)
       j ← j − 1
       if j = 0 then break // alla biljetter har använts
if j = 1 then S.push(1)
print "Maximal mängd fisk ", F[n, kmax], " fås på följande sätt:"
for j ← 1 to kmax do print "Fiska vid ", S.pop()
```

Algoritmen tar tid $O(nk)$ eftersom det som mest är två nästlade slingor och dessa går $n - 1$ respektive k varv.