Uppgifter till föreläsning 3 om dynamisk programmering, DD2350 adk20

1. Konstruera matriskedjemultiplikation

Bakgrund

Matriskedjemultiplikation är problemet att hitta bästa sättet att multiplicera ihop en kedja av matriser av varierande storlek. Genom att multiplicera ihop matriserna med varandra i en listig ordning kan antalet operationer som krävs minimeras. Att multiplicera en $a \times b$-matris och en $b \times c$-matris med vanlig matrismultiplikation tar abc talmultiplikationer.

Vi vill beräkna matrisprodukten $A_1 A_2 \cdots A_n$ där matris A_i har dimension $r_{i-1} \times r_i$.

Låt $M[i,j]$ = minimala antalet multiplikationer som behövs för att räkna ut matrisprodukten $A_i A_{i+1} \cdots A_j$.

Rekursionsekvationen för $M[i,j]$ är:

$$M[i,j] = \begin{cases} 0 & \text{om } i = j \\ \min_{i \le k < j}(M[i,k] + M[k+1,j] + r_{i-1} r_k r_j) & \text{om } i < j \end{cases}$$

Basfallen är $M[i,i]$, det vill säga diagonalen i (den triangulära) matrisen M. Beräkningsordning: Beräkna elementen efter stigande avstånd från diagonalen, alltså så att indexen i och j skiljer sig mer och mer.

```
OptimalMatmul(M[1..n, 1..n], r) =
    for i ← 1  to n do M[i, i] ← 0
    for len ← 1  to n − 1 do
        for i ← 1  to n − len do
            j ← i + len
            M[i, j] ← ∞
            for k ← i  to j − 1 do
                t ← M[i, k] + M[k + 1, j] + r_{i−1} · r_k · r_j
                if t < M[i, j] then M[i, j] ← t
    return M[1, n]
```

Det minimala antalet multiplikationer beräknas av `OptimalMatmul(M, r)`. Svaret finns då i $M[1,n]$. Algoritmen har komplexiteten $O(n^3)$.

Uppgift

Man vill inte bara veta det minimala antalet multiplikationer utan också i vilken ordning man ska multiplicera ihop matriserna för att uppnå det minimala antalet. Låt oss bygga ut `OptimalMatmul` så att man kan få reda på ordningen. Konstruera en hjälpmatris $breakpoint[i,j]$ som innehåller var gränsen går mellan dom två matriserna som bildar produkten $A_i A_{i+1} \cdots A_j$. Exempel: $breakpoint[7,9] = 7$ betyder att man ska multiplicera $A_7(A_8 A_9)$ och $breakpoint[7,9] = 8$ betyder att man ska multiplicera $(A_7 A_8)A_9$.

Ändra i algoritmen ovan så att $breakpoint[i,j]$ sätts till rätt värden.

Skriv allra sist i algoritmen ut hur multiplikationen ska ske i form av ett parentesuttryck där matriserna symboliseras med A, till exempel $(((AA)A)(AA))$, genom att anropa en rekursiv funktion PrintWithParentheses$(1, n, breakpoint)$.

2. Bevisa matriskedjemultiplikation

Konstruera en lämplig invariant för slingan **for** $len \leftarrow 1$ **to** $n-1$. Det som ska uttryckas i invarianten är att en viss del av M är korrekt beräknad (dvs enligt rekursionen).

Visa sedan att invarianten är uppfylld när man går in i slingan, och att om invarianten gäller när man går ut ur slingan så returneras rätt värde från funktionen.

Konstruera nu en lämplig invariant för den inre slingan **for** $i \leftarrow 1$ **to** $n-len$ och visa att den är uppfylld när man går in i den inre slingan.Visa sedan med hjälp av invarianten att den första invarianten är korrekt.

3. Implementera hållbart fiske

Den som har tid: Hitta en lämplig beräkningsordning för hållbart fiske (på förra uppgiftsbladet, återpublicerad härunder) och implementera sedan algoritmen enligt den. Vad är tidskomplexiteten för algoritmen?

Experter på hållbar utveckling har utformat ett ganska komplicerat kvotsystem för att begränsa fiske, ett system som bygger på *fiskebiljetter*. Varje fiskare får k stycken fiskebiljetter numrerade 1 till k och som ska användas i nummerordning. Varje gång fiskaren fiskar gör han eller hon av med en fiskebiljett. Hur mycket fisk som fiskaren får dra upp då beror på för vilken gång i ordningen fiskaren fiskar, alltså fiskebiljettens nummer. Man får bara fiska en gång vid varje fiskeplats.

Anta att en fiskare åker en rutt som i tur och ordning passerar n stycken lämpliga fiskeplatser, numrerade 1 och n, och fiskaren vill fiska vid k av dessa fiskeplatser. Fiskaren vill veta vilka k platser hon ska fiska vid för att få mest fisk.

Problemet är att givet fiskekvoterna $K_{i,j}$ (som står för hur många fiskar man får ta upp vid fiskeplats i om man använder fiskebiljett j), där $1 \leq i \leq n$

och $1 \leq j \leq k$, beräkna den maximala mängden fisk som kan fångas om man väljer dom k fiskeplatserna optimalt. Alla fiskebiljetterna behöver inte gå åt.

Låt $F[i,j]$ vara maximala antalet fiskar som kan fås vid fiskeplats $1 \ldots i$ med användning av dom j första biljetterna för alla $1 \leq i \leq n$ och $0 \leq j \leq k$.

Rekursionen blir

$$F[i,j] = \begin{cases} 0 & \text{om } j = 0 \\ K_{1,1} & \text{om } i = j = 1 \\ -\infty & \text{om } i = 1,\ j > 1 \\ \max(F[i-1,j], F[i-1,j-1] + K_{i,j}) & \text{om } i > 1,\ j > 0 \end{cases}$$

Lösningen till problemet är $\max_{1 \leq j \leq k} F[n,j]$.

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.