Algoritmer og Datastrukturer (NDAA04010U)
Ugeopgave 3 med svar og forklaringer

Københavns Universitet

2024

1 Amortiseret analyse

I denne opgave betragter vi potentialfunktionen der bruges til at analysere dynamiske tabeller i CLRS sektion 16.4.1. Lad Φ_i betegne potentialfunktionens værdi efter i TABLE-INSERT operationer, uden nogen TABLE-DELETE operationer. Hvilke egenskaber har Φ_i? Vælg ét eller flere korrekte svar og beskriv hvordan du kom frem til dem (både positive og negative svar).

1. $\Phi_i = \Omega(1)$.✓
2. $\Phi_i = O(1)$.
3. $\Phi_i = \Omega(i)$.
4. $\Phi_i = O(i)$.✓
5. $\Phi_i \geq \Phi_{i-1}$.
6. $\Phi_i \geq 0$. ✓

Svar: $\Phi_i = 2 \cdot num_i - size_i$, hvor $num_i = i$ og $size_i \geq 1$ er den aktuelle kapacitet. Vi har altid $\Phi_i \leq i/2$, så $\Phi_i = O(i)$, men lige efter en tabeludvidelse har vi $\Phi_i \leq 2$, dvs. det gælder ikke at $\Phi_i = \Omega(i)$. Lige efter en tabeludvidelse har vi $\Phi_i < \Phi_{i-1}$. Potentialfunktionen er gyldig, så den er ikke-negativ (og den er positiv fra første skridt og fremefter). △

2 Dynamisk programmering

En *stigende delsekvens* af en tabel A, der indeholder heltal $A[1], \ldots, A[n]$, er en delmængde af indekser $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ således at $A[i_1] < A[i_2] < \cdots < A[i_k]$. Længden af delsekvensen er antal indekser, k. En *ikke-aftagende delsekvens* defineres på samme måde bortset fra at kravet er $A[i_1] \leq A[i_2] \leq \cdots \leq A[i_k]$. Betragt følgende funktion, i pseudo-kode notationen fra CLRS, hvor input A er en tabel af heltal.

```
MaximumMystery(A, n)
1  let r[1...n] be a new array
2  r[1] = 1
3  for j = 2 to n
4       q = 1
5       for i = 1 to j − 1
6            if not A[j] < A[i]
7                 q = max(q, r[i] + 1)
8       r[j] = q
9  return max_{1≤ℓ≤n} r[ℓ]
```

Hvad beregner proceduren MaximumMystery? Vælg præcis ét svar og beskriv hvordan du kom frem til det. Formulér en invariant for den ydre løkke som en del af dit svar.

1. Indekserne i en længste stigende delsekvens.
2. Indekserne i en længste ikke-aftagende delsekvens.
3. Længden af en længste stigende delsekvens.
4. Længden af en længste ikke-aftagende delsekvens. ✓

Svar: En invariant for den ydre løkke er at efter hver iteration gælder for $\ell = 1 \ldots, j$ at $r[\ell]$ er længden af den længste ikke-aftagende delsekvens, der indeholder indekset ℓ. Den indre løkke beregner $r[j]$ ved rekursionsformlen $r[j] = 1 + \max_{i<j,\, A[i]\leq A[j]} r[i]$, hvor maximum af den tomme mængde defineres som 0. Hvis $A[j]$ er mindre end alle tidligere værdier i A beholdes initialiseringsværdien $q = 1$. △

3 Længste fælles delsekvens

Vi betragter algoritmen til længste fælles delsekvens (eng. *longest common subsequence*) i CLRS sektion 14.4.

a) Tegn tabellen c som algoritmen udfylder på følgende input:

$$x = \langle N, A, A, N\rangle \text{ og } y = \langle A, N, N, A\rangle.$$

Tilføj pile ($\uparrow$, $\nwarrow$, $\leftarrow$), der forklarer tallene i tabellen, som i CLRS Figur 14.8 og angiv en længste fælles delsekvens.

Svar: For overskuelighedens skyld inkluderer vi de to strenge i kanten af tabellen c, ligesom i CLRS Figur 14.8.

		A	N	N	A
	0	0	0	0	0
N	0	$\uparrow 0$	$\nwarrow 1$	$\nwarrow 1$	$\leftarrow 1$
A	0	$\nwarrow 1$	$\uparrow 1$	$\uparrow 1$	$\nwarrow 2$
A	0	$\nwarrow 1$	$\uparrow 1$	$\uparrow 1$	$\nwarrow 2$
N	0	$\uparrow 1$	$\nwarrow 2$	$\nwarrow 2$	$\uparrow 2$

(I nogle tilfælde er der et valg mellem $\uparrow$ og $\leftarrow$, det lægger vi ikke vægt på.) En længste fælles delsekvens er NA som udledes af ovenstående, en anden er AN. $\triangle$

4 Vægtet længste fælles delsekvens

Vi betragter følgende variant af længste fælles delsekvens (eng. *longest common subsequence*) problemet fra CLRS sektion 14.4. Lad Σ betegne en mængde af tegn (et "alfabet"). Input er to vektorer $x = \langle x_1, \ldots, x_m \rangle \in \Sigma^m$ og $y = \langle y_1, \ldots, y_n \rangle \in \Sigma^n$. I lighed med længste fælles delsekvens er målet at finde en optimal vektor $z = \langle z_1, \ldots, z_k \rangle$, men i stedet for at maksimere længden af z skal *vægten* af tegnene i z maksimeres. Vi antager at vægtene er ikke-negative. Vægten er givet ved en datastruktur w, således at vægten af $z_i \in \Sigma$, der betegnes med $w[z_i]$, kan findes i konstant tid. Vægten af z er summen af vægte, $\sum_{i=1}^{k} w[z_i]$, og målet er et finde en fælles delsekvens med størst mulig vægt.

Eksempel. Antag at vi har $x = \langle A, H, A, A \rangle$, $y = \langle A, A, R, G, H \rangle$, og $w[A] = 2$, $w[G] = 8$, $w[R] = 10$, $w[H] = 5$. Delsekvensen $\langle A, A \rangle$ med vægt $2 + 2 = 4$ findes i både x og y. Delsekvensen $\langle A, H \rangle$ med vægt $2 + 5 = 7$ findes også i både x og y, og er den delsekvens, der har størst vægt.

a) Skriv en rekursionsligning for den optimale vægt af en fælles delsekvens af x og y.

Svar: I forhold til det almindelige længste fælles delsekvens problem skal vi ændre omkostningen i det tilfælde hvor to karakterer matcher så det afspejler vægten:

$$c[i,j] = \begin{cases} 0 & \text{hvis } i = 0 \text{ eller } j = 0 \\ c[i-1, j-1] + w[\alpha] & \text{hvis } i, j > 0 \text{ og } x_i = y_j = \alpha \\ \max(c[i-1, j], c[i, j-1]) & \text{hvis } i, j > 0 \text{ og } x_i \neq y_j \end{cases}$$

$\triangle$

b) Lav en analyse af køretiden af den algoritme, der bruger rekursionsligningen og dynamisk programmering til at beregne den optimale vægt. Analysen skal give en øvre grænse på køretiden som funktion af n og m i store-O notation, der er så præcis som mulig.

Svar: Analysen for længste fælles delsekvens kan genbruges, da rekursionsligningen har samme struktur. Ændringen påvirker kun værdierne og tiden for hvert skridt øges med

højst en konstant faktor (da en vægt kan findes i konstant tid). Derfor er køretiden $O(nm)$. $\triangle$

c) Beskriv hvordan algoritmen kan modificeres til at håndtere det tilfælde hvor vægte kan være negative uden at påvirke den asymptotiske køretid. Argumentér for køretid og korrekthed af din løsning.

Svar: Tegn med negative vægte vil aldrig være del af en optimal løsning, hvilket kan ses med et modstridsargument: Hvis z indeholder et tegn med negativ vægt, så har delsekvensen der udelader dette element højere vægt. Tegn med negativ vægt kan derfor fjernes fra input i tid $O(n+m)$, hvorefter det resterende problem er uden negative vægte. Den samlede tid er $O(n+m+nm)$, hvilket er $O(nm)$ som inden. $\triangle$

5 Binomialkoefficienter

Lad n og k være heltal, $n \geq k \geq 0$. Binomialkoefficienten $\binom{n}{k}$ er antal måder at tage k forskellige elementer ud af en mængde med n elementer. Der gælder:

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!} = \prod_{i=0}^{k-1} \left(\frac{n-i}{k-i} \right)$$

a) Vis at følgende ligheder gælder:

$$\binom{n}{k} = \begin{cases} 1 & \text{hvis } k = 0 \text{ eller } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{hvis } 0 < k < n \end{cases}$$

Svar: Tilfældene $k = 0$ og $k = n$ følger af at $0! = 1$. I det sidste tilfælde har vi:

$$\begin{aligned} \binom{n-1}{k-1} + \binom{n-1}{k} &= \frac{(n-1)!}{(n-k)!(k-1)!} + \frac{(n-1)!}{(n-1-k)!k!} \\ &= \frac{k(n-1)!}{(n-k)!k!} + \frac{(n-k)(n-1)!}{(n-k)!k!} \\ &= \frac{n!}{(n-k)!k!} = \binom{n}{k} \ . \end{aligned}$$

Alternativt kan man argumentere kombinatorisk: Enten tager man det første element, og så er der $\binom{n-1}{k-1}$ muligheder for at vælge $k-1$ fra de sidste $n-1$, eller også tager man ikke det første element og så er der $\binom{n-1}{k}$ muligheder for at tage k elementer fra de sidste $n-1$. (**Kommentar**: Denne opgave kan ses som en analyse af "Pascal's trekant".) $\triangle$

b) Brug del a) og dit kendskab til dynamisk programmering til at skrive pseudokode for en algoritme, der beregner $\binom{n}{k}$ ved brug af højst nk additioner, uden brug af multiplikation.

Svar: Vi kan lave en dynamisk programmeringstabel A med $n+1$ rækker og $k+1$ søjler, hvor $A[i,j]$ er tiltænkt værdien $\binom{i}{j}$. Lighederne i del a) gør at vi kan beregne basistilfældene først, og derefter de øvrige indgange række for række, fra venstre til højre.

```
BINOMIAL(n, k)
1  Let A[0 : n, 0 : k] be a new table
2  for i = 0 to n
3        A[i, 0] = 1
4  for i = 1 to k
5        A[i, i] = 1
6  for i = 1 to n
7        for j = 1 to min(i − 1, k)
8              A[i, j] = A[i − 1, j − 1] + A[i − 1, j]
9  return A[n, k]
```

Ifølge del a) sikrer tildelingerne i linie 3, 4 og 8 at $A[i,j] = \binom{i}{j}$. I de to løkker er der højst nk iterationer hver der laves én addition i hver. $\triangle$