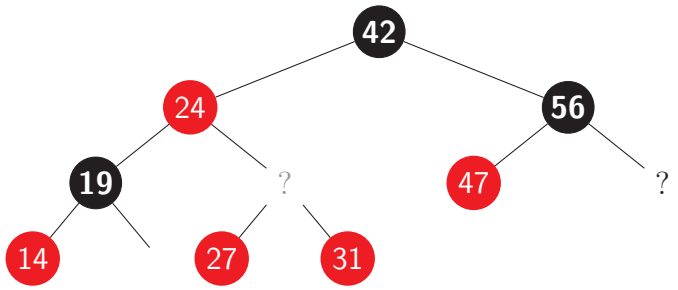
Algoritmer og Datastrukturer (NDAA04010U)
Ugeopgave 4 med svar og forklaringer

Københavns Universitet

2024

1 Rød-sorte søgetræer

Betragt et rød-sort binært søgetræ T (eng. *red-black binary search tree*), som beskrevet i CLRS kapitel 13. Betragt nedenstående træ, hvor alle blade er udeladt (som på CLRS figur 13.1.c) og to indre knuder er uspecificerede:



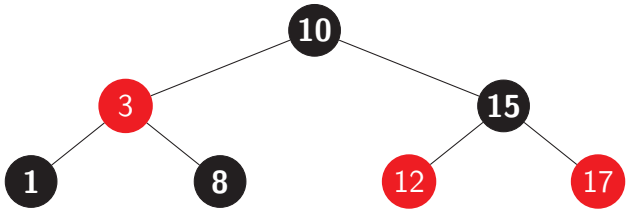
Hvilke knuder er mulige på mindst én af positionerne markeret med ?, hvis træet skal overholde invarianterne for et rød-sort binært søgetræ? Vælg ét eller flere korrekte svar og beskriv hvordan du kom frem til dem (både positive og negative svar).

1. En rød knude med nøglen 28.
2. En rød knude med nøglen 44.
3. En rød knude med nøglen 58. ✓
4. En sort knude med nøglen 25.
5. En sort knude med nøglen 30. ✓
6. En sort knude med nøglen 66.

Svar: Knuden til venstre skal være sort (fordi røde knuder har sorte børn) og ligge mellem 27 og 31. Derfor er kun en sort knude med nøglen 30 en mulighed hér. Knuden til højre skal være rød (fordi alle stier fra roden skal have samme antal sorte knuder) og større end 56, så kun en rød knude med nøglen 58 er mulig hér. △

2 Indsættelse i rød-sorte søgetræer

Betragt dette rød-sorte binære søgetræ (eng. *red-black binary search tree*), hvor bladene er udeladt på tegningen:



Antag at vi bruger indsættelsesalgoritmen RB-Insert fra CLRS kapitel 13 til at indsætte nøglen 9. Hvad sker med træet? Vælg præcis ét svar og beskriv hvordan du kom frem til det.

1. 9 indsættes som højre barn til knude 8 og farves rød. ✓
2. 9 indsættes som højre barn til knude 8 og farves sort.
3. 9 indsættes som venstre barn til knude 12 og farves rød.
4. 9 indsættes som venstre barn til knude 12 og farves sort.
5. Ingen af ovenstående.

Svar: RB-Insert finder det rette sted i træet til nøglen 9, som det højre barn under knude 8. Knuden farves rød og derefter kaldes RB-Insert-Fixup, men den terminerer uden at ændre noget fordi forældreknuden 8 er sort. △

3 Køretid for rød-sorte søgetræer

Betragt et rød-sort binært søgetræ T (eng. *red-black binary search tree*), som beskrevet i CLRS kapitel 13, hvorpå der (startende med et tomt træ) udføres n_1 Insert operationer, n_2 Delete operationer, og n_3 Search operationer (sidstnævnte kaldes også "access" operationer). Det totale antal operationer er $N = n_1 + n_2 + n_3$. Hvilke af følgende udsagn er altid sande? Vælg ét eller flere korrekte svar og beskriv hvordan du kom frem til dem (både positive og negative svar).

1. Den samlede tid for operationerne er $\Omega(N)$. ✓
2. Den samlede tid for operationerne er $O(N \lg N)$. ✓
3. Den samlede tid for operationerne er $O(n_3 + (n_1 + n_2) \lg N)$.
4. Dybden af T er højst $2 \lg(n_1 + 1)$. ✓

5. Alle blade i T er i dybde mindst $\lfloor \lg(n_3 + 1) \rfloor$.

6. Alle sorte knuder i T har en afstand til roden, der er et lige tal.

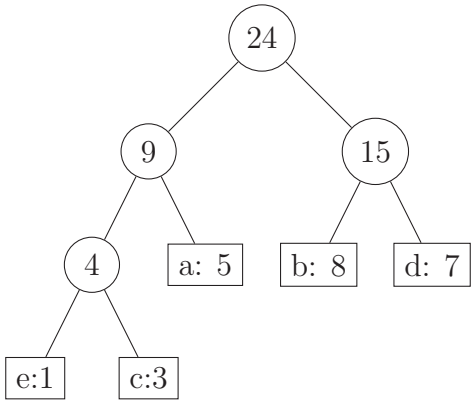
Svar: Hver operation tager mindst konstant tid, så den samlede tid er $\Omega(N)$. Antal knuder n i træet er på ethvert tidspunkt højst $n_1 \leq N$. Hver operation tager altså højst $O(\lg N)$ tid, så den samlede tid er $O(n \lg N)$. I værste fald tager søgeoperationerne tid $\Omega(\lg n_1)$, dvs. tid $\Omega(n_3 \lg n_1)$ totalt, og tiden kan altså ikke begrænses af $O(n_3 + (n_1 + n_2) \lg N)$. Et rød-sort binært søgetræ kan have blade i dybde 2, bladenes dybde er ikke større end $\lfloor \lg(n_3 + 1) \rfloor$ generelt. Sorte knuder kan optræde på alle niveauer. $\triangle$

4 Huffman koder

Vi betragter algoritmen til konstruktion af Huffman koder i CLRS sektion 15.3.

a) Tegn Huffman træet som algoritmen beregner for alfabetet $C = \{a, b, c, d, e\}$ med frekvenser $f_a = 5$, $f_b = 8$, $f_c = 3$, $f_d = 7$, $f_e = 1$. Det er ikke nødvendigt at angive labels (0 og 1) på kanterne, men angiv frekvenser i blade såvel som indre knuder (se eksempel i CLRS figur 15.5).

Svar: Algoritmen beregner dette træ (tegnet uden 0-1 labels på kanterne):



(Bemærk at rækkefølgen af børnene til hver knude er arbitrær, så den grafiske repræsentation kan se anderledes ud.) $\triangle$

5 Dynamisk programmering

Vi et givet en liste $x_0, x_1, \ldots, x_n$ med n positive heltal. Betragt følgende rekursive definition af værdier $m_{i,j}$, hvor $1 \leq i \leq j \leq n$:

$$m_{i,j} = \begin{cases} 1 \text{ hvis } i = j \\ \max\{m_{i,k} + m_{k+1,j} + x_{i-1}x_k x_j \mid i \le k < j\} \text{ hvis } i < j \end{cases}$$

En rekursiv algoritme der direkte følger rekursionsligningen, uden at gemme beregnede værdier, bruger tid $O(t_{i,j})$ til at beregne $m_{i,j}$, hvor

$$t_{i,j} = \begin{cases} 1 \text{ hvis } i = j \\ 1 + \sum_{k=i}^{j-1} t_{i,k} + t_{k+1,j} \text{ hvis } i < j \end{cases}$$

a) Argumentér for at $t_{i,j} \ge 2^{j-i+1} - 1$. Brug gerne induktion i $\ell = j - i$.

Svar: I tilfældet $\ell = j - i = 0$ har vi $t_{i,j} = 2^{j-i+1} - 1 = 1$ per definition. Antag i induktionsskridtet $\ell > 0$ at $t_{i,j} \ge 2^{j-i+1} - 1$ for alle i, j med $j - i < \ell$ og betragt i, j med $j - i = \ell$. Vi har at

$$t_{i,j} = 1 + \sum_{k=i}^{j-1} t_{i,k} + t_{k+1,j} \ge 1 + t_{i,j-1} + t_{i+1,j} \ge 1 + (2^{(j-1)-i+1} - 1) + (2^{j-(i+1)+1} - 1) = 2^{j-i+1} - 1,$$

hvor induktionsantagelsen bruges i den sidste ulighed. $\triangle$

b) Forklar, gerne med pseudokode, hvordan dynamisk programmering kan bruges til effektivt at beregne $m_{i,j}$ for alle $1 \le i \le j \le n$. Analysér plads og køretid for algoritmen, i store-O notation, som funktion af n.

Svar: Rekursionsformlen for $m_{i,j}$ svarer til rekursionsformlen for MATRIX-CHAIN-ORDER i CLRS afsnit 14.2, blot med mininum erstattet af maksimum og et ændret basistilfælde, hvilket ikke påvirker algoritmen. Værdierne med $\ell = j - i$ kan beregnes for $\ell = 0, 1, \ldots, n$, og totalt n^2 værdier. Derfor er køretid og plads $O(n^2)$. (Se evt. pseudokode på CLRS s. 378, der dog har en ekstra tabel s til at holde styr på den optimale løsning.) $\triangle$

6 Grådige algoritmer

Vi betragter igen `MaxProduct` problemet fra Ugeopgave 2:

Givet en liste af tal $x_1, \ldots, x_n \in \mathbf{R}$ (kan være både positive og negative), find en delmængde $I^* \subseteq \{1, \ldots, n\}$ hvor produktet $\prod_{i \in I^*} x_i$ er så stort som muligt, dvs. for alle mængder $I \subseteq \{1, \ldots, n\}$ skal gælde at $\prod_{i \in I} x_i \le \prod_{i \in I^*} x_i$.

Eksempel. På input $x_1 = -4$, $x_2 = -0.5$, $x_3 = 0.5$, $x_4 = 0.5$, $x_5 = 1.5$, $x_6 = 6$ kan vi vælge $I = \{5, 6\}$ og få et produkt på $1.5 \cdot 6 = 9$, men et bedre valg er $I = \{1, 2, 5, 6\}$ som giver produktet $(-4) \cdot (-0.5) \cdot 1.5 \cdot 6 = 18$. Den tomme mængde $I = \emptyset$ giver per definition $\prod_{i \in \emptyset} x_i = 1$.

Et mulig tilgang til at løse `MaxProdukt` er at grådigt føje elementer til I, ét ad gangen. Antag at input er sorteret således at $x_1 \leq x_2 \leq \cdots \leq x_n$.

a) Argumentér for at for en optimal løsning I^* vil der altid vil være et lige antal elementer i mængden $I^*_- = \{i \in I^* \mid x_i < 0\}$, dvs. $|I^*_-|$ er deleligt med 2.

Svar: Vi ved at $\prod_{i \in I^*} x_i \geq 1$ da den tomme mængde er et muligt valg. For at få et produkt, der er positivt, er antal negative elementer i produktet nødt til at være lige. $\triangle$

b) Foreslå en effektiv algoritme til at finde en optimal løsning I^*. Argumentér for korrekthed og køretid. Der lægges vægt på, at argumentationen er klar og koncis. (Det er *ikke* et krav at argumentere via "greedy choice property" og "optimal substructure".)

Svar: Problemet kan deles i to uafhængige problemer, der kan løses separat. Antag at k er indekset på det største negative input, dvs. $x_1, \ldots, x_k < 0$ og $x_{k+1}, \ldots, x_n \geq 0$. Vi ønsker at finde $I^*_- \subseteq \{1, \ldots, k\}$ og $I^*_+ \subseteq \{1, \ldots, k\}$ således at $I^* = I^*_- \cup I^*_+$ er optimal. Vi ved at alle produkter der kan opnås med indekser i I^*_+ er positive, så den optimale løsning kan ikke indeholde et negativt produkt over indekser i I^*_-. Derfor er det optimalt at maksimere de to delproblemer separat. Et optimalt valg for de positive tal er $I^*_+ = \{i \mid x_i > 1\}$ hvilket kan ses ved et modstridsargument. Definér $q = |\{i \mid x_i < -1\}|$. For de negative tal er $I = \{1, \ldots, q\}$ ikke nødvendigvis optimalt, da det kan indeholde et ulige antal elementer q (i modstrid med spørgsmål b). Hvis q er ulige er der to kandidater til en optimal løsning I^*_-: $\{1, \ldots, q-1\}$ og $\{1, \ldots, q+1\}$. For at se at det ikke er muligt at finde en bedre løsning betragt en vilkårlig mængde $I_- \subseteq \{1, \ldots, k\}$ af lige størrelse. Hvis to elementer i I_- er større end q så kan vi få en bedre løsning ved at fjerne dem. Hvis to elementer mindre end eller lig med q ikke er i I_- så kan vi få en bedre løsning ved at tilføje dem dem. Derfor kan en optimal løsning højst mangle ét element fra $\{1, \ldots, q\}$ og højst indeholde ét element der ikke er i $\{1, \ldots, q+1\}$. Specifikt fås de optimale værdier ved at tilføje $q+1$ eller ved at fjerne q. Hvis input er sorteret tager det lineær tid at finde I^*_- og I^*_+. $\triangle$