Assignment 3 - AlgDat

Nicklas Warming Jacobsen
Christian Enevoldsen
Simon Warg
Robert Rasmussen

20. maj 2014

Task 1

Algorithm 1 Algorithm to CanDistribute

for $i = 1$ to $n - 1$ **do**
  $distance = p_{i+1} - p_i$
  if $b_i < \hat{b}$ **then**
    $b_{i+1} = b_{i+1} - (\hat{b} - b_i) - 2 * distance$
    $b_i = \hat{b}$
  end if
  if $b_i > \hat{b}$ **then**
    $b_{i+1} = max(0, b_i - \hat{b} - 2 * distance)$
    $b_i = \hat{b}$
  end if
end for
return $b_n \geq \hat{b}$

Task 2

Hvis $b_1 < \hat{b}$, så vil den billigste, og dermed den beste, løsningen være at tage øl fra b_2. Hvis b_2 derefter får mindre end $\hat{b}$ øl så vil den endeste måde være

at tage øl fra b_3. Den anden case er; Hvis $b_1 > \hat{b}$, så vil den beste løsningen være at sørge for at b_2 får så mange øl fra b_1 som mulig indtil $b_1 = \hat{b}$. Dermed gælder det at $\hat{b} \leq b_1 \leq b_2 ... \leq b_{n-1}$, efter algoritmen har kørt. Hvis $\hat{b} \leq b_n$, ved vi at det gælder at $\hat{b} \leq b_1 \leq b_2 ... \leq b_n$ og vi kan konkluderer at det kan lade sig gøre at distribuere $\hat{b}$, men hvis $\hat{b} \geq b_n$ kan vi konkluderer at det ikke er muligt at distribuere $\hat{b}$.

Task 3

Algorithm 2 Find maximum number of beer that can be distributed among all bars

```
left = 0
right = B
while right − left > 1 do
    if CanDistribute(⌊(right+left)/2⌋) then
        left = ⌊(right+left)/2⌋
    else
        right = ⌊(right+left)/2⌋
    end if
end while
if CanDistribute(right) then
    return right
else
    return left
end if
```

Funktionen kører i $O(\log B) + O(n)$ fordi i hvert iteration så tjekkes det hvis maximum m, $0 \leq m \leq B$, i et søgområde der halveres efter hver iteration. Hver gang den søger i søgområdet bruger den $O(n)$ tid, altså kører funktion i $O(n \log B)$.
Funktionen er korrekt siden den hver gang tjekker hvis maximum er under eller over $B/2$. Er den under så fortsætter den at undersøge om maximum er under eller over $B/2 - B/4$, er den over så tjekker den hvis maximum er over eller under $B/2 + B/4$ indtil søgområdet er kun to tal, hvor den tjekker begge for maximum.

Exam outline Nicklas W. Jacobsen qmr656

En grådig algoritme tager et valg og så løser et under-problem, i modsætning til DP algoritme. Når en grådig algoritme har taget et valg efterlader den kun et (mindre) underproblem.

Steps for udvikling af en grådig algortime

1. Find den optimale sub-struktur af problmet
2. Udvikle en rekursiv løsning.
3. Vis at ved at lave et grådigt valg, efterlader man kun ét delproblem.
4. Hvis at det altid er "sikkert" at lave et grådigt valg.

Huffmans algoritme

Huffmans algortime laver et optimalt parsetræ for en given sekvens, hvor hver blad repræsenterer et word og dens frekvens i sekvensen. Hver knude i træet repræsenterer summen af sine børns frekvens. I et Huffmanstræ er de blade med mindst frekvens længst nede i træet, og får derved længste prefix-kode. længden af indkodningen er givet ved: $B(T) = \sum_{s \in S} f(s) \cdot d_T(s)$, hvor $d_T(s)$ er dybden af s i parsetræet.

Bevis for at der kan laves et optimalt grådigt valg

Lad S være en sekvens af words hvor hver word $s \in S$ har en frekvens $f(s)$. Lad x og y være to words i S med de laveste frekvenser.

Lemma: Der eksisterer en optimal prefix-kodning af S hvor x og y har den samme prefix-indkodnings længde og kun differencerer i deres sidste bit.

Bevis: Idéen er at antage et træ T som repræsenterer et vilkårligt optimalt parsetræ for prefix indkodning, og modificerer det til et andet træ som også repræsenterer en optimal prefix inkodning, hvor x og y er har samme parent og ligger i maksimum dybde af træet.
Steps:

1. Lad a og b være to words som er søskendeblade i T og som ligger i maksimum dybde.

2. Vi antager $f(a) \leq f(b)$ og $f(x) \leq f(y)$. Og da x og y er de to words med lavest frekvens gælder det $f(x) \leq f(a)$ og $f(y) \leq f(b)$.

3. For at gøre det non-trivielt antager vi $f(x) \neq f(b)$, da det således vil betyde at det gælder $f(x) = f(y) = f(a) = f(b)$.

4. Vi bytter om på x og a og konstruerer et nyt parsetræ T'

5. Vi beregne differencen mellem indkodningslængden $B(T)$ og $B(T')$

6. $$\begin{aligned}
&\sum_{s \in S} f(s) \cdot d_T(s) - B(T) - \sum_{s \in S} f(s) \cdot d_{T'}(s) \\
&= f(x) \cdot d_T(x) + f(a) \cdot d_T(a) - f(x) \cdot d_{T'}(x) + f(a) \cdot d_{T'}(a) \\
&= f(x) \cdot d_T(x) + f(a) \cdot d_T(a) - f(x) \cdot d_T(a) + f(a) \cdot d_T(x) \\
&= (f(a) - f(x)) \cdot (d_t(a) - d_t(x))
\end{aligned}$$

   Både $f(a) - f(x)$ og $d_t(a) - d_t(x)$ er non-negativt da $f(x)$ er den laveste frekvens, og $d_t(a)$ er den største dybde (per difinition), og derfor gælder det at:

   $$(f(a) - f(x)) \cdot (d_t(a) - d_t(x)) \geq 0$$

7. Hvis vi bytter om på y og b i T' og konstruerer et nyt træ T'', så gælder samme ræsonnement som i forrige punkt. Det gælder derfor at:

   $$B(T') - B(T'') \geq 0.$$

   Det følger deraf $B(T'') \leq B(T)$, men da T er optimalt gælder det også $B(T) \leq B(T'')$, hvilket er ensbetydende med $B(T'') = B(T)$

Exam outline - Christian Enevoldsen

Optimal delstruktur
Grådige algoritmer opnår optimal delstruktur, ved at foretage en række valg, til at løse et problem Ved hvert valg, vælger den det der ser bedst ud for

delstrukturen/problemet. Det hedder det grådige valg.

Typiske steps, fra CLRS
1. Vælg en optimal delstruktur af problemet
2. Udvikle en rekursiv løsning
3. Vis at hvis der foretages det grådige valg, så er der kun et delproblem tilbage
4. Bevis at det altid er sikkert at foretage sig det grådige valg
5. Udvikle en rekursiv algoritme som foretager sig disse valg
6. Udvikle den rekursive algoritme til en iterativ en.

Det grådige valg
Grådige algoritmer foretager sig et valg, ved hvert delproblem for at opnå optimal delstruktur, hvis muligt. Dette er et valg som foretages lokalt i en delstruktur for at løse det globale problem.

DP vs. GA

DP og GA er ofte metoder som bruges når man skal optimerer noget. Som regel er DP dog langsommere og mere ineffektiv når GA kan bruges. I stedet for at løse samtlige delproblemer og lave en masse delstrukturer vælger GA den delstruktur der ser bedst ud på tidspunktet, så den kun ender med en delstruktur og dermed kun et delproblem at løse.

Huffmann
Huffman koder er en komprimeringsmetode, som oftest bruges til strenge. Den bruger en tabel, som indeholder alle tegn og hvor tit dit opstår i en strengesekevens til at repræsentere den som en binary streng.
Den opbygges ved at lave et træ. Træet opbygges af knuder, som har en venstre og en højre værdi og en frekvens. Algoritmen itererer fra 1 til n-1, hvor den i hver itereration sætter z's (ny knude) venstre side y til minimumsværdien i C (et sæt af alle karakterer, med frekvens) og højresiden y til den næste minimumsværdi i C. Derefter sætter den knudens frekvens til x.freq + y.freq, og indsætter det i det opbyggende træ.

Exam outline - Robert Schannong Rasmussen

Points

- Greedy-choice property
  - The algorithm makes the best choice available to it in a given subproblem.
- Optimal substructure
  - "A problem exhibits optimal substructure if an optimal solution to the problem contains optimal solutions to the sub-problems."
- Greedy strategy
  - Six steps on how to develop a greedy algorithm.
- Traveling salesman problem
  - How greedy algorithms is affected by this problem.
- Coin example
  - Given three types of coins (25-cent, 10-cent and 4-cent) and why a greedy algorithm won't be able to give 41 cents back in change.
- Matroid
  - Using combinatorial structures to show that a greedy method yields an optimal solution.

1 Problem Instance

A set of frequencies to be compressed using Huffman encoding or activity selection algorithm