

Grådige algoritmer

Mads Ohm Larsen

6. april 2008

1 Hvad er en grådig algoritme?

En grådig algoritme, er en algoritme der altid tager den optimale løsning på et givent tidspunkt. Det vil sige at den altid gør det der er bedst lige her og nu. Vi håber altså på at den løsning vi vælger fører til en optimal løsning på hele problemet.

2 Relation mellem grådige algoritmer og dynamisk programmering

I modsætning til dynamisk programmering, hvor vi udregner et valg på baggrund af hvad vi tidligere har gjort, vælger grådige algoritme den optimale løsning til et underproblem og håber at det fører til en optimal global løsning. Med dynamisk programmering løser vi altså problemerne fra bunden og op (fra små problemer til større). Med grådige algoritmer løser vi problemet fra toppen og ned (fra store problemer til små).

3 Korrekthed

Vi kan ikke bare sådan vise at en grådige algoritme løser problemet, men vi har 2 egenskaber vi kan sige noget ud fra.

3.1 Grådige-valg egenskab

Det grådige-valg egenskaben viser sig ved at vi kan lave optimale (grådige) løsninger til underproblemer på problemet.

3.2 Optimal substruktur

Et problem har optimal substruktur, hvis en optimal global løsning basere sig på optimale lokale løsninger.

4 Aktivitetsskeduleringproblemet

En række aktiviteter, alle med et start- og et sluttidspunkt skal passe ind i et lokale.

4.1 Optimal substruktur

Antag at der er fra 1.. n aktiviteter. Lad den 0. aktivitet slutte før alle andre og den $n + 1$ starter efter alle andre. Kigger vi på $S_{ij} = \{a_k \in S : f_i < s_k < f_k < s_j\}$, altså mængden af aktiviteter der starter efter a_i slutter og slutter før a_j starter, ser vi at $S = S_{0,n+1}$, $S = \emptyset$ for $i \leq j$ og $a_k \in S_{ij} \Rightarrow i < k < j$.

Den optimale løsning til dette problem må være den optimale løsning til S_{ik} og den optimale løsning til S_{kj} . Ergo er der optimal substruktur.

4.2 Rekursiv ligning

Lad $c[i, j]$ være den største submængde af kompatible aktiviteter i S_{ij} .

$$c[i, j] = \begin{cases} 0 & \text{for } S = \emptyset \\ \max_{a_k \in S_{ij}} \{c[i, k] + c[k, j] + 1\} & \text{for } S \neq \emptyset \end{cases} \quad (1)$$

4.3 Grådig algoritme

Samme som før, men vi kigger kun på den aktivitet, $a_m \in S_{ij}$, der slutter tidligst.

1. a_m bliver brugt i en optimal løsning til et subproblem i S_{ij}
2. Subproblemet S_{im} er tomt

Bevis:

1. I en hver optimal løsning til S_{ij} kan vi skifte den aktivitet med tidligst sluttidspunkt ud med a_m uden at lave overlap.
2. Kommer direkte af at a_m har det tidligste sluttidspunkt af alle aktiviteterne.

Med dette kan vi nu løse problemet fra toppen og ned og lave grådige valg, hele tiden med mindre underproblemer!

5 Huffman-kode problemet

En Huffman-kode er en algoritme der tager en tekststreng og ud fra hyppigheden laver den et binært træ således at alle tegn for en unik bitsekvens. Dem med højst hyppighed vil have den korteste. Det mest hyppige tegn kunne have koden 0, og derfor kan ingen andre tegn have et prefix 0. Dette kaldes prefix-koder.

5.1 Grådig algoritme til løsning af Huffman-koden

Vi sortere tegnstrengen efter deres hyppighed i en minimumskø. Efter vær iteration sætte vi de to sammen der har de mindste hyppigheder i et træ. Roden i træet for værdi der er summen af dets børn. Derefter sætter vi dette træ ind i minimumskøen hvor det skal være.

Når algoritmen har kørt $n - 1$ gange vil vi have et element på venstre side og et træ på højre side. Elementet vil være det element der har den højeste hyppighed.

5.2 Køretid

Minimumskøenen kan initialiseres for n tegn på $O(n)$ tid, hvis vi bruger min-hob operationer. Herefter vil vi i hver iteration finde det mindste og sætte det sammen med resten af hobben. Da alle operationer med hobbe tager $O(\lg n)$ tid, vil hver iteration tage $O(\lg n)$ tid. For n tegn giver dette $O(n \lg n)$ tid.

6 Andre problemer

Både mindst udspændende træer og korteste vej algoritmer gør brug af grådige algoritmer!