Algoritmer (datastrukturer) och komplexitet, våren 2010

Uppgifter till övning 4

Dekomposition och lådpackning

Max och min med dekomposition I vektorn $v[1..n]$ ligger n tal. Konstruera en dekompositionsalgoritm som tar reda på det största och det minsta talet i v. Algoritmen ska använda högst $\lceil 3n/2 \rceil - 2$ jämförelser mellan v-element. Antalet tal i v behöver inte vara en tvåpotens.

Snabb lådpackning *Lådpackningsproblemet* (*the bin packing problem* på engelska) är följande problem. Du får n stycken metallprylar som var och en väger mellan 0 och 1 kg. Du får också ett antal stora men sköra lådor. Målet är att hitta det minsta antal lådor som behövs för att förvara alla n metallprylarna utan att någon låda innehåller mer än 1 kg.

Detta är ett känt problem som är svårt att lösa exakt (det är ett så kallat *NP-fullständigt* problem). Därför ska vi nu nöja oss med att hitta en lösning som inte är optimal med hjälp av följande enkla algoritm:

Anta att både metallobjekten och lådorna är numrerade från 1 till n. Ta en metallpryl i taget (i tur och ordning) och stoppa den i den första låda som rymmer den (dvs den låda med lägst nummer som inte blir för tung om man stoppar i prylen).

Din uppgift är att beskriva hur denna algoritm kan implementeras så att den går i tid $O(n \log n)$ (i värsta fallet och med enhetskostnad). För att uppnå detta kommer du att behöva konstruera en heapliknande datastruktur i vilken du snabbt kan söka upp den första låda som rymmer den aktuella prylen.

Matrismultiplikation Strassens algoritm multiplicerar två $n \times n$-matriser i tid $O(n^{2.808})$ genom dekomposition i 2×2-blockmatriser. Anledningen till att Strassens algoritm går snabbare än $O(n^3)$ är att den gör sju multiplikationer istället för åtta för att bilda produktmatrisen.

En annan idé är att göra dekomposition i 3×3-blockmatriser istället. Viggo försökte för ett par år sedan hitta det minimala antalet multiplikationer som krävs för att multiplicera två 3×3-matriser. Han lyckades nästan komma fram till 22 multiplikationer. Om han hade lyckats, vilken tidskomplexitet hade det gett för multiplikation av två $n \times n$-matriser?

Komplex multiplikation Om man multiplicerar två komplexa tal $a + bi$ och $c + di$ på det vanliga sättet krävs fyra multiplikationer och två additioner av reella tal. Eftersom multiplikationer är dyrare än additioner (och subtraktioner) lönar det sig att minimera antalet multiplikationer om man ska räkna med stora tal. Hitta på en algoritm som bara använder tre multiplikationer (men fler additioner) för att multiplicera två komplexa tal.

Majoritet med dekomposition Indata är i denna uppgift en array A med n element. Konstruera och analysera en algoritm som tar reda på om något element i arrayen A är i majoritet, det vill säga förekommer mer än $n/2$ gånger, och i så fall returnerar det. Algoritmen ska vara en dekompositionsalgoritm och ha tidskomplexiteten $O(n \log n)$. Enda tillåtna jämförelseoperationen på element i A är $=$. Det finns alltså ingen ordningsrelation mellan elementen.

Lösningar

Lösning till Max och min med dekomposition
När man bara har två tal räcker det med en enda jämförelse för att både hitta det största och det minsta talet.

```
MinMax(v,i,j)=
  if i=j then return (v[i],v[i])
  else if i+1=j then
     if v[i]<v[j] then return (v[i],v[j])
     else return (v[j],v[i])
  else
     m:=Floor((j-i)/2)
     if Odd(m) then m:=m+1;
     (min1,max1):=MinMax(v,i,i+m-1);
     (min2,max2):=MinMax(v,i+m,j);
     min:=(if min1<min2 then min1 else min2);
     max:=(if max1>max2 then max1 else max2);
     return (min,max);
```

Beräkningsträdet kommer att få $\lceil n/2 \rceil$ löv och $\lceil n/2 \rceil - 1$ inre noder. Om n är jämnt är alla $n/2$ löv tvåelementsföljder och gör därför en jämförelse. Om n är udda är ett löv (det högraste) ett enstaka element som inte kräver någon jämförelse. I löven görs alltså $\lfloor n/2 \rfloor$ jämförelser. I varje inre nod görs två jämförelser, alltså sammanlagt $2\lceil n/2 \rceil - 2$ stycken. Totalt får vi $\lfloor n/2 \rfloor + 2\lceil n/2 \rceil - 2 = \lceil 3n/2 \rceil - 2$ stycken jämförelser.

Det går faktiskt att visa att problemet inte kan lösas med färre jämförelser. □

Lösning till Snabb lådpackning
Eftersom n prylar ska stoppas ner i lådorna i tid $O(n \log n)$ så söker vi en metod som letar reda på i vilken låda en pryl ska stoppas ner i tid $O(\log n)$. Eftersom antalet lådor kan vara upp till n så måste sökalgoritmen i varje steg förkasta ungefär hälften av lådorna. I så fall har man efter $\log n$ steg förkastat alla lådor utom en, och då vet man i vilken låda prylen ska ligga.

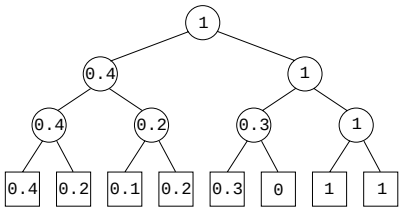
Det finns bara två kriterier efter vilka man kan förkasta lådor:

1. om en låda inte rymmer prylen,

2. om en låda har högre ordningsnummer än den första låda som rymmer prylen.

För att kunna förkasta hälften av lådorna med en sökning så måste vi hålla reda på hur tung den tyngsta prylen får vara som kan stoppas ner i den första hälften av lådorna respektive i den andra hälften av lådorna. Detta måste vi hålla reda på rekursivt för varje hälft.

Datastrukturen blir alltså ett fullständigt binärt träd i $\log n$ nivåer, där trädets löv utgörs av lådorna. I varje löv lagrar vi hur mycket motsvarande låda rymmer (inledningsvis 1). I varje inre nod i trädet lagrar vi det största av sönernas värden. Datastrukturen ser nu ut som en heap med det största värdet överst. Exempel med åtta lådor som är fyllda med 0.6, 0.8, 0.9, 0.8, 0.6, 1, 0 och 0 kg:



Algoritmen som stoppar ner en pryl med vikt x i den första låda som rymmer den blir nu:

```
void FindBin(double x, int i)
{ if (i >= n)                    /* Är detta ett löv (dvs en låda)? */
    H[i] = H[i] - x;
  else {
    if (H[2*i] >= x)             /* Rymmer vänstra sonen x? */
      FindBin(x, 2*i);           /* Ja, fortsätt i vänster delträd. */
    else
      FindBin(x, 2*i+1);         /* Nej, fortsätt i höger delträd. */
    H[i] = max(H[2*i],H[2*i+1]); /* Uppdatera aktuell nod. */
  }
}
```

Man anropar proceduren med `FindBin(x,1)`. □

Lösning till Matrismultiplikation
Rekursionsekvationen blir $T(n) = 22 \cdot T(n/3) + O(n^2)$. Mästarsatsen ger lösningen $T(n) = O(n^{\log_3 22}) = O(n^{2.814})$. □

Lösning till Komplex multiplikation
Egen övning. □

Lösning till Majoritet med dekomposition
Om det finns ett majoritetselement måste det vara i majoritet i åtminstone ena halvan av arrayen. Rekursiv tanke: Kolla majoritet rekursivt i vänstra och högra halvan och räkna sedan hur många gånger halvarraysmajoritetselementen förekommer i hela arrayen. Om något element är i total majoritet returneras det.

Majority($A[1..n]$) =
  if $n = 1$ **then return** $A[1]$
  $m \leftarrow \lceil (n+1)/2 \rceil$
  $v \leftarrow$ Majority($A[1..m-1]$)
  $h \leftarrow$ Majority($A[m..n]$)
  if $v = h$ **then return** v
  $vn \leftarrow 0; hn \leftarrow 0$
  for $i \leftarrow 1$ **to** n **do**
    if $A[i] = v$ **then** $vn \leftarrow vn + 1$
    else if $A[i] = h$ **then** $hn \leftarrow hn + 1$
  if $vn \geq m$ **then return** v
  if $hn \geq m$ **then return** h
  else return NULL

Tidskomplexitet: Två rekursiva anrop med halva arrayen följt av efterarbetet $O(n)$ ger med mästarsatsens hjälp tidskomplexiteten $O(n \log n)$. □
