

Algoritmik eksamen vinteren 1997/98

4 timers skriftlig prøve

Opgave 1 (30 %)

En *binomialhob* er en implementation af en *prioritetskø*. Dvs en datastruktur til lagring af en samling elementer fra en ordnet mængde.

Binomialhobe understøtter følgende operationer:

lavhob(e) Returnerer en binomialhob med elementet e .

findmin(h) Returnerer en pointer til det minimale element i hoben h .

indsæt(h, e) Tilføjer elementet e til hoben h .

sletmin(h) Sletter det minimale element i h .

foren(h_1, h_2) Returnerer en binomialhob, der indeholder elementerne fra h_1 og h_2 .

Det antages for nemheds skyld at alle elementer, der lagres i det følgende, er forskellige.

En binomialhob h består af en række *binomiale hobtræer*, en pointer min_h samt en heltalsværdi max_h .

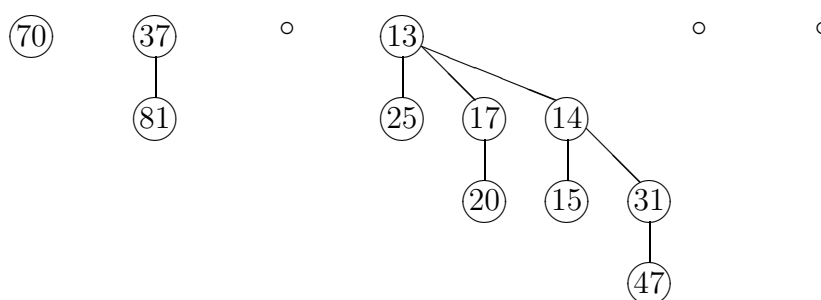
Binomiale hobtræer af *type* i er defineret rekursivt ved:

1. Et binomialt hobtræ af type 0 består af en knude.
2. Et binomialt hobtræ af type i består af en rod med i efterfølgere, der er rødder i binomiale hobtræer af type $0, 1, \dots, i-1$. Elementet i roden er mindre end elementerne i efterfølgerne.

Der er højst et træ af hver type i en binomialhob. De binomiale hobtræer i hoben h er givet ved en række pointere $h(i)$, $i = 0, 1, 2, \dots$ som enten peger på roden i et binomialt hobtræ af type i eller er **nil**.

min_h peger på det mindste element lagret i h . (Bemærk, at pga. (2) ovenfor er det mindste element rod i et af hobtræerne i h .)

max_h angiver den maksimale type hobtræ i h .



Eksempel på en binomialhob med 11 elementer. Den indeholder 3 hobtræer af type 0,1 og 3. min peger på element "13" og max er 3.

Spørgsmål 1A

1. Vis, at et binomialt hobtræ af type i har 2^i knuder.
2. Vis, at alle knuder i et binomialt hobtræ er rod i et undertræ, der er et binomialt hobtræ.

□

Implementationen af nogle af operationerne ovenfor benytter sig af **join** af to binomielle hobtræer af samme type. **join**(t_1, t_2) fungerer som følger:

join(t_1, t_2): Hvis t_1 og t_2 er af type i returneres et binomialt hobtræ af type $i + 1$ dannet ved at træet med det største element i roden gøres til undertræ af roden i det andet. Denne operation tager tid $O(1)$.

- **lavhob**(e) implementeres simpelt ved at oprette et hobtræ af type 0 med e som knude og sætte max og min til de rigtige værdier.
- **findmin**(h) implementeres simpelt ved at returnere min_h .
- Implementationen af **foren**(h_1, h_2) minder meget om addition af to binære heltal:

```

foren( $h_1, h_2$ ):
   $min := \min\{max_{h_1}, max_{h_2}\};$ 
   $carry := \mathbf{nil};$ 
   $i := 0;$ 
  while ( $i \leq min$ )  $\vee$  ( $carry \neq \mathbf{nil}$ ) do
    if  $carry, h_1(i)$  og  $h_2(i)$  alle er nil then
       $h(i) := \mathbf{nil};$ 
    fi;
    if præcis en af  $carry, h_1(i)$  og  $h_2(i)$  er forskellig fra nil then
      sæt  $h(i)$  til dette træ;
    fi;
    if præcis to af  $carry, h_1(i)$  og  $h_2(i)$  er forskellig fra nil then
      sæt  $carry$  til join af disse to træer;
    fi;
    if  $carry, h_1(i)$  og  $h_2(i)$  alle er forskellige fra nil then
       $h(i) := carry;$ 
       $carry := \mathbf{join}(h_1(i), h_2(i))$ 
    fi;
     $i := i + 1$ 
  od;
   $max_h := \max\{max_{h_1}, max_{h_2}\};$ 
  if  $max_h < i - 1$  then
     $max_h := i - 1$ 
  fi

```

- **indsæt**(h, e) implementeres ved: **foren**(**lavhob**(e), h)
- **sletmin**(h) implementeres ved:

sletmin(h):

Dan en binomial hob h_1 af efterkommerne af roden i træet,
som min_h peger på;

Slet træet min_h i h ;

Find hvilket af de tilbageværende træer i h , der har den mindste rod
og sæt min_h til at pege på denne rod;

Opdater om nødvendigt max_h ;

foren (h_1, h);

Bemærk at hvis min_h peger på roden af et træ t , der er af type i , så bliver h_1 en binomial hob, hvis hobtræer netop er undertræerne af roden i t .

Det antages, at træer af type i er lagret på en sådan måde, at undertræerne til roden kan gennemløbes i en tid $O(i)$.

Spørgsmål 1B

Lad h være binomialhoben i eksemplet ovenfor.

Tegn h efter udførelse af operationen **indsæt**($h, 43$) og tegn h efter yderligere udførelse af operation **sletmin**(h). □

Spørgsmål 1C

Vis, at alle operationerne har worst case udførelsestid $O(\log n)$, hvor n er antal elementer i hoben efter udførelse af operationen. □

Spørgsmål 1D

Vis, at udførelsestiden for en lovlig sekvens s bestående af a_{lav} **lavhob** operationer, a_{find} **findmin** operationer, a_{ind} **indsæt** operationer, a_{slet} **sletmin** operationer og a_{for} **foren** operationer, på en fra start tom mængde af hobe

$$O(a_{lav} + a_{find} + a_{ind} + (a_{slet} + a_{for}) \log n),$$

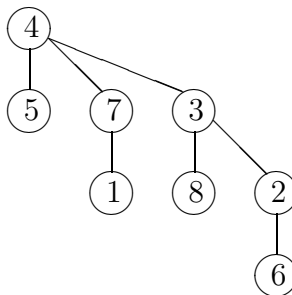
hvor n er det maksimale totale antal elementer, der er i hobene under udførelse af sekvensen s . \square

Opgave 2 (30 %)

I det følgende repræsenteres orienterede træer med n knuder ved arrays af størrelse n . Kanterne i træerne er orienteret mod roden.

Et træ t med knuder $1, 2, \dots, n$ og rod r er repræsenteret ved array $P[1..n]$, hvis $P[r] = r$ og faderen til knuden $i \neq r$ er $P[i]$.

Eksempel:



Dette træ er repræsenteret ved $P = [7, 3, 4, 4, 4, 2, 4, 3]$.

Følgende CRCW PRAM algoritme har som input array P , der repræsenterer et træ t . Som output produceres et array D , hvor $D[j]$ er dybden af knude j i t .

```

procedure dybde( $P, n$ )[ $p_1, p_2, \dots, p_n$ ];
  for  $i := 1$  to  $n$  pardo
    if  $P[i] = i$  then  $D[i] := 0$  else  $D[i] := 1$  fi;
  od
   $a := 1$ ;
  while  $a < n$  do
    for  $i := 1$  to  $n$  pardo
       $D[i] := D[i] + D[P[i]]$ ;
       $P[i] := P[P[i]]$ ;
    od;
     $a := 2 \cdot a$ 
  od

```

For P i eksemplet ovenfor bliver $D = [2, 2, 1, 0, 1, 3, 1, 2]$.

Spørgsmål 2A

Udfør proceduren dybde på eksemplet ovenfor ($P = [7, 3, 4, 4, 4, 2, 4, 3]$). Angiv P og D efter hver udførelse af kroppen i **while**-løkken.

Spørgsmål 2B

Vis, at de korrekte dybder bliver beregnet i D af proceduren dybde.

Spørgsmål 2C

Hvad er arbejdet for proceduren dybde?

Spørgsmål 2D

Vis, at array D kan beregnes ud fra P (og n) i tid $\sqrt{n} \log n$ ved hjælp af \sqrt{n} processorer på en CRCW PRAM.

Spørgsmål 2E

Gør rede for at array D kan beregnes ud fra P (og n) i tid $\sqrt{n} \log n$ ved hjælp af \sqrt{n} processorer på en EREW PRAM. \square

Opgave 3 (20 %)

Lad $CIRCUIT_{\vee}-VALUE$ være $CIRCUIT-VALUE$ problemet for netværk, der kun har \vee -knuder.

Spørgsmål 3A

Vis, at $CIRCUIT_{\vee}-VALUE \in NL$. \square

Lad $CIRCUIT_{\wedge}-VALUE$ være $CIRCUIT-VALUE$ problemet for netværk, der kun har \wedge -knuder.

Spørgsmål 3B

Vis, at $CIRCUIT_{\wedge}-VALUE \in NL$. \square

Opgave 4 (20 %)

Lad $G = (V, E)$ være en orienteret graf. $A \subseteq V$ er et *uafhængigt anker* for G hvis A er en uafhængig mængde af knuder (der er ingen kanter mellem knuder i A) og for alle $u \in V \setminus A$ er der en knude $v \in A$ (et anker for u) således at $(v, u) \in E$.

Spørgsmål 4A Vis at alle orienterede grafer med to knuder har et uafhængigt anker. Angiv for hver graf med to knuder, de mulige ankre. \square

Spørgsmål 4B Hvis, at der findes en orienteret graf med tre knuder, der har et uafhængigt anker. \square

Problemet *ANKER* er problemet:

Givet en orienteret graf $G = (V, E)$.
Findes der et uafhængigt anker for G ?

Spørgsmål 4C Vis, at *ANKER* er *NP* fuldstændig. \square