Diskret Matematik og Algoritmer

Aflevering 7g

Adam Ingwersen,
Aske Fjellerup,
Peter Friborg

Datalogisk Institut
Københavns Universitet

December 5, 2016

1

Det skal i denne delopgave vises hvordan man kan implementere træstrukturerede disjunkte mængder, ved hjælp af en tabel.

1.1

Der opskrives pseudokode til de 3 funktioner:

- **Find(x)** - Denne funktion skal finde repræsentanten for den mængde, x befinder sig i.
- **Init(n)** - Denne funktion skal oprette en tabel med n elementer, der hver består af en mængde indeholdende sig selv alene
- **Union(x, z)** - Denne funktion skal sammenlægge mængderne x og z til en samlet mængde.

Først opskrives pseudokoden for **Find(x)**. Denne opbygges rekursivt:

Algorithm 1 Find(x) - Finder repræsentanten for den mængde x er i.

```
function FIND(x)
    if x ≠ x.p
        Find(x.p)
    else Return x.p
```

Den næste funktion, **Init(n)**, laves ved hjælp af en hjælpefunktion n kald til **MakeSet(x)**, der opretter en mængde med x som rod, og x.p peger tilbage på x.
MakeSet(x) kan se ud som følger:

```
MakeSet(x)
    x.p = x
    x.rank = 0
```

Algorithm 2 Init(n) - Opretter en tabel med n elementer.

```
function INIT(n)
    for i=0 to n-1
        A[i] = MakeSet(i)
```

Den sidste funktion, **Union(x, z)**, skal kunne tage to mængder og forene dem, hvis de ikke allerede er i samme mængde. Funktionen skal tage den mindste mængde, og sætte på den større mængde. Dette gøres ikke ved hurtig forening. Derfor er der brug for en hjælpefunktion **ChangeAll(x, z, P)**.
ChangeAll(x, z, P) udskifter alle elementer **x** med **z** i tabellen P. Den kan se ud som følger:

```
ChangeAll(x, z, P)
    for each element in P
        if x == P[i]
            P[i] = z
```

Algorithm 3 Union(x, z) - Forener x og z til en enkelt mængde.

```
function UNION(x, z)
    rx = Find(x)
    rz = Find(z)
        if rx.rank > rz.rank
            ChangeAll(rz, rx, A)
        else ChangeAll(rx, rz, A)
            if rx.rank == rz.rank
                rz.rank = rz.rank + 1
```

Det noteres at **ChangeAll(x, z, P)** kører igennem hele P ved hvert kald. Dog bliver **Find(x)** hurtigere, og tabellen, A, bliver nemmere at se på. Alle tal på tabellen i samme mængde vil pege på samme repræsentant.

1.2

Det skal nu afprøves, hvordan forskellig operationer vil efterlade tabellen A:

$$\begin{aligned}
\text{Init}(7) &= A[0,1,2,3,4,5,6] \\
\text{Union(3, 4)} &= A[0,1,2,3,3,5,6] \\
\text{Union(5, 0)} &= A[0,1,2,3,3,0,6] \\
\text{Union(4, 5)} &= A[0,1,2,0,0,0,6] \\
\text{Union(4, 3)} &= A[0,1,2,0,0,0,6] \\
\text{Union(0, 1)} &= A[0,0,2,0,0,0,6] \\
\text{Union(0, 4)} &= A[0,0,2,0,0,0,6] \\
\text{Union(6, 0)} &= A[0,0,2,0,0,0,0]
\end{aligned}$$

Med denne metode er det tydeligt at se hvilke mængder, der har samme repræsentant.

1.3

Den væsentlige forskel mellem **Union(x, z)** der gennemgås i denne opgave og den **Union(i, j)** der gennemgås i noterne, er måden hvorved repræsentanten udskiftes. I noterne er det kun en enkelt peger, der skal skiftes ud, hvorimod her i opgaven skiftes alle pegere ud i hele tabellen. Metoden, hvor alle pegere bliver skiftet ud på, sker ved at kigge hele tabellen igennem, dette er tidskrævende, derimod er **Find(x)** hurtigere. Tabellen bliver lettere at se på, da alle tallene er roden i et træ. Dette gør det lettere at se, hvor mange elementer der er i hver enkelt mængde, samt at se hvor mange forskellige mængder der er.

2

Det skal i denne opgave vises, at højden for et træ, der er forekommet ved **UNION** med `union by rank` heuristikken, højest kan være logaritmen af antallet af knuder i træet.

2.1

I `union by rank` heuristikken gælder det, at der for hver node vedligeholdes en rank - denne starter ved for alle noder. I det tilfælde, hvor vi betragter `union by rank` uden `path-compression`, vil enhver **UNION** følge logikken fremført i **Link(x,y)**, CLRS s. 571. Det følger direkte af **Link**-funktionen, at rank af et vilkårligt træ `t` som er resultatet af **UNION** af 2 andre vilkårlige træer, `t` og `t` højest kan være steget med 1 og mindst må være lig rank for det af de to træer med højest rank.

2.2

2.3

- **Første markeing** : viser P(1) er sand

  For at P(1) er sand må der gælde at $rank(t) \leq lg(1)$. Da der kun er et element i træet (roden) har kan Udtrykket omskrives til $0 \leq lg(1) = 0$. vi kan derfor se at P(1) er sand

Overstående opgave at hvis $rank(t_1) \neq rank(t_2)$ gælder $rank(t) \leq max(rank(t_1), rank(t_2))$. det bruger vi som udgangspunkt til at lave induktion.
rank(t) kan ikke være støre end 2-tals logaritmen til t da det er et binært træ. Det skyldes at lg af de j elementer i træet giver en længde tilsvarende til højden af træet.

$$rank(t) \leq max(rank(t_1), rank(t_2)) \leq max(lg(i), lg(l))$$

Da lg(i+j) altid vil være mere end hver for så længe de begge er positive

$$rank(t) \leq max(lg(i), lg(l)) \leq lg(i + j) = lg(n)$$

her får vi logaritmen af log(n)

2.4