Assignment 4 - AlgDat
Genaflevering

Nicklas Warming Jacobsen
Christian Enevoldsen
Simon Warg
Robert Rasmussen

06-06-2014

Task 1

Datastruktur

Datastrukturen for vores Online Incremental Connectivity graf gør brug af Path Compression metoden og Rank metoden, vi kan tillade os at bruge Path Compression da det ikke bliver fjernet kanter i grafen. Ved Path compression bliver parent'en, hos de elementer der indgår i en Find-Path (dvs. også i Query), substitueret med dis-join sættets repræsentant, hvilket betyder at den næste query bliver hurtigerer hvis nolge af de samme elementer indgår. Rank metoden er en metode som forgår under Link og Make-Set operationenerne. Hver gang der laves et nyt dis-join set med ét element via Make-Set bliver rank for elementet sat til 0. Under Link operationen kan der ske to forskellige udfald: 1; I tilfælde af at de to elementer der skal linkes har samme rank, bliver et arbitrært element valgt, det valgte elements rank bliver sat én op og det andet elements parent bliver sat til det valgte element. 2; I tilfælde af at de to elementer der skal linkes ikke har samme rank, bliver elementet med størst rank sat som det andet elements parent. Ved at bruge Rank og Path Compression metoderne bliver køretiden for en operation $O(lg^*(n))$ og $\Omega(1)$.

Algorithm 1 Online incremental connectivity

```
function MAKE-SET(x)
    x.parent = x
    x.rank = 0
end function

function FIND-SET(x)
    if x.parent ≠ x then
        x.parent = Find-Set(x.parent)
    end if
    return x.parent
end function

function QUERY(x, y)
    return Find-Set(x) == Find-Set(y)
end function

function LINK(x, y)
    if x.rank < y.rank then
        x.parent = y
    else
        y.parent = x
        if x.rank == y.rank then
            x.rank = x.rank + 1
        end if
    end if
end function
```

Task 2

Algorithm 2 CountComponents - Retunerer antallet af connected components i en graf G(V, E)

V: En liste af trivielle sæt
E: En liste af tubler med to trivielle sæt som er forbundet i grafen G

```
function CountComponents(V, E)
    c = V.length()
    for e in E do
        if Not Query(e[0], [1]) then
            Link(e[0], e[1])
            c = c-1
        end if
    end for
    for v in V do                    ▷ Vi laver v'erne om til trivielle sæt igen
        Make-Set(v)
    end for
    return c
end function
```

Algorithm 3 MakeMatrix - laver en n*n relations matrise over unlink operationer, dvs. m[i][j] = 1 hvis der er en unlink operation mellem knude i og j

n: er størrelsen af matrisen (antallet af knuder) U: er en liste af tubler med to trivielle sæt som skal unlinkes (Unlink operationer)

```
function MakeMatrix(n, U)
    m[n][n];
    for u in U do
        m[u[0].val()][u[1].val()] = 1
        m[u[1].val()][u[0].val()] = 1
    end for
    return m
end function
```

Algorithm 4 Unlinker - Retunerer antallet connected components efter hver unlink operation

V: En liste af knuder (1,2,3...)
E: En liste af kanter, fx. ([1,2],[4,5]...)
U: En liste af kanter som skal unlinkes (Unlink operationer)

```
function UNLINKER(V, E, U)
    for v in V do
        Make-Set(v)
    end for
    m = MakeMatrix(V.length(), U)
    c = CountComponents(V, E) - 1
    for e in E do
        if m[e[0].val()][e[1].val()] = 1 And Not Query(e[0], e[1]) then
            c = c +1
        else
            Link(e[0], e[1])
        end if
    end for
    R[] = c
    for u in U.reversed() do
        if Not Query(u[0],[1]) then
            Link(u[0], u[1])
            c = c-1
        end if
        R.push(c)
    end for
    return R
end function
```

Task 3

Bevis forCountComponents

Vi ser på antallet af connected components i delgrafen G' af G(E, V). Hvor vi i hver iteration tilføjer en kant til G' fra G. Invariancen er at variable c er antallet af connected components i G'.

Initialization

Vi ser på delgrafen G' af G(V, E) uden nogle kanter, så vil antallet af knuder være lig antallet af connected components. c sættes derfor på linje 2 til længden af V. Invariancen holder derfor inden loop'ets start.

Iteration

Hvis det gælder at de to knuder mellem kanten e, ikke er i samme component i G', må det gælde at $\{e\} \cup G'(E)$, har en mindre component end G'. Ved at tilføjde kanten til G' og mindske c med en, holder iterationen loop-invariancen.

Termination

Iterationen stopper når vi har itereret over alle e i $G(E)$ og derved tilføjet alle e i $G(E)$ til $G'(E)$, og da c er antallet af connected components i G' må c også være antallet af connected components i G.

MakeMatrix

Funktioen laver en n*n relations matrise over unlink operationerne. Vi beviser ikke denne da den er triviel.

Bevis for Unlinker

Funktionen består af to iterationer. Den første iteration konstruerer og tæller connected components efter alle U_n unlink operationer af lavet på grafen G, den næste iteration finder antallet connected components efter U_{n-i} unlink operationer

Initialization 1

c er antallet af connected components før unlink operationerne. Eftersom der er lavet 0 unlink operationer før loopet, holder vores invariance.

Iteration 1

Hvis e_i er den eneste forbindelse mellem knuderne forbundet af e_i, så gælder det at $G' = G(E, V \setminus \{e_i\})$ må have $c + 1$ connected components

Termination 1

Loopet stopper når alle $e \in E$ er blevet blevet sammenholdt med U, og da vi har talt c op når det gælder at $e \in U$ og der ikke eksistere andre veje til knuderne i mellem ved vi at c er antallet af components i $G(V, E - U)$

Initialization 2

Det gælder inden iterationen at $R = \{c\}$, hvilket er korrekt da $i = 0$.

Iteration 2

Hvis der ikke er nogle forbindelse mellem kanterne unlinket af u_i i G', så gælder det at $G'(V, \{u_i\} \cup E)$ må have $c - 1$ connected components

Termination 2

Loppet stopper når alle $u \in U$ er tilføjet til G'. det må derfor gælde ved terminering at $G' = G$ og alle c_i for alle i er fundet.

Tidskompleks

CountComponents

Vi går ud fra at vi kan tælle V i linær tid, linje 2 tager derfor $O(n)$. På linje 3 til 8 er et loop der køre funktionerne Query m gange og Link maks m gange, loopet tager derfor $O(2m \cdot \alpha(n))$ hvilket også er $O(m \cdot \alpha(n))$. På linje 9 til 11 er et loop der køre n gange og kalder funktionen Make-Set, som tager konstant tid, loopet tager derfor $O(n)$.

Til sammen er tidskompleksen for funktionen $O(n + m \cdot \alpha(n) + n)$ hvilket også er $O(m \cdot \alpha(n))$

MakeMatrix

Funktionen består af et loop der udføre konstant tid operationer, loopet køre maks m gange. Tidskompleksen for funktionen er derfor $O(m)$.

Unlinker

På linje 2 til 4 er et loop der køre n gange, og udføre en konstanttids funktion, loopet tager derfor $O(n)$. Linje 5 kalder MakeMatrix som tager $O(m)$. Linje 6 kalder CountComponents der tager $O(m \cdot \alpha(n))$. Linje 7 til 13 er et loop der bliver kørt m, loopet slår op i en matrise i konstant tid, samt kalder funktionenerne Query og Link som begge tager $O(\alpha(n))$. loopet på linje 7 til 13 tager derfor $O(m \cdot \alpha(n))$. På linje 15 til 21 er et loop der bliver kørt m gange. Loop på linje 15 til 21, kalder Query og Link som begge tager $O(\alpha(n))$, loopet tager defor $O(m \cdot \alpha(n))$.
Tidskompleksen for hele funktionen er derfor: $O(n+m+m\cdot\alpha(n)+m\cdot\alpha(n)) = O(2(m\cdot\alpha(n))+m+n$, og da m af asymptotisk mindre $m\cdot\alpha(n)$, og konstanter ikke har en betydning i store O notation, er tidskompleksen

$$O(m \cdot \alpha(n) + n)$$