Sommeren 2001, opgave 1

Spørgsmål a

Vi antager at $k \geq 3$, da det ellers er uklart hvordan trekanterne kan sættes sammen i en kreds. Vi ser nu at for hver trekant er der en knude i kredsen, og en spids. Derfor er $n = 2k$. For hver trekant er der indlysende 3 kanter. Derfor er $m = 3k$. Dette giver at $n = \frac{2}{3}m$.

Spørgsmål b

Ikke relevant for dADS 2 eksamen, da Ω ikke gennemgået i dADS 1.

Spørgsmål c

Korteste veje

Vi identificerer først den trekant som s hører til i, og kalder hjørnerne a og b.

I konstant tid kan vi finde den korteste afstand til a og b som kun går gennem trekanten s, a, b.

Trekant s, a, b markeres besøgt.

Startende med $v = a$ udføres følgende nu iterativt

while $v \neq b$

> find den næste trekant ved at finde de 2 kanter fra v som ikke går til en besøgt knude.
>
> marker denne trekant besøgt.
>
> korteste afstand til knuderne i trekanten er korteste afstand til v
> + korteste afstand gennem trekanten.
>
> $v \leftarrow$ det andet hjørne i trekanten.

Herefter gentages samme princip startende i b, indtil man når a. Den korteste afstand defineres til at være den mindste af de 2 fundne afstande.

Argument for korrekthed: Enhver korteste vej fra s til t må fra s gå gennem enten a eller b, og herefter følge den korteste vej gennem knuderne i kredsen enten den ene eller anden vej til den trekant nås som indeholder t, og vælge den korteste vej gennem denne trekant.

Men det er præcist hvad algoritmen ovenfor gør, da den finder korteste vej begge veje rundt gennem kredsen, og vælger den korteste vej gennem alle trekanter.

Argument for udførselstid: Der laves konstant-tids initialisering + 2 gange konstant arbejde for hver trekant. Tid: $O(1 + 2k) = O(k) = O(2n) = O(n)$.

Mindste udspændende træ

Vi starter med at finde en start-trekant. Det gøres ved at vælge en tilfældig knude. Hvis den ikke har grad 2 vælges en af dens naboer med grad 2.

Fra denne knude med grad 2 kan man let identifisere den tilhørende trekant. Vi kalder dens hjørner i kredsen for a og b. I denne trekant sletter vi den tungeste kant.

Der er nu 2 tilfælde: Den slettede kan var fra kredsen, eller den slettede kant var *ikke* fra kredsen.

Hvis den slettede kant var fra kredsen huskes i variablen *tungeste* den tungeste af de to andre. Ellers huskes kanten fra kredsen i variablen *tungeste*. Vi fortsætter nu således, startende med $v = a$:

while $v \neq b$

find den næste trekant ved at finde de 2 kanter fra v som ikke går til en besøgt knude.

marker denne trekant besøgt.

slet den tungeste kant i denne trekant.

if slettet kan var i kredsen

then *tungeste* $\leftarrow$ den tungeste af de 2 andre kanter i trekanten,
hvis den er tungere end *tungeste*.

else *tungeste* $\leftarrow$ trekantens kant i kredsen, hvis den er tungere end *tungeste*.

$v \leftarrow$ det andet hjørne i trekanten.

Til sidst slettes *tungeste*.

De tilbageværende kanter udgør et mindste udspændende træ.

Argument for korrekthed: Algoritmen sletter een kant i hver trekant. Da disse er den tungeste i en cykel er det korrekt at undlade denne kant ifølge sætningen i opgaven.

Algoritmen sletter desuden kanten *tungeste*. Men denne er den tungeste kant i en cykel, nemlig den cykel der består af kreds-kanter, hvor de ikke er slettet, og begge de andre kanter i en trekant hvor de er. Derfor er det også korrekt at slette denne kant.

Når disse kanter er slettet er der $m - k - 1$ kanter tilbage. Men $m - k - 1 = 3k - k - 1 = 2k - 1 = n - 1$ kanter, og derfor har vi et træ, og alle tilbageværende kanter er medlem i mindste udspændende træ. Derfor giver algoritmen et mindste udspædende træ.

Argument for udførselstid: Der laves konstant-tids initialisering, og konstant arbejde for hver trekant. Tid: $O(1 + k) = O(k) = O(2n) = O(n)$.

Sommeren 2003, opgave 3

Spørgsmål a

$U = ST$ er en super-sekvens for S og T af længde $|U| = |S| + |T|$, d.v.s. korteste fælles super-sekvens har længde $\leq |S| + |T|$.

$S =$`A` og $T =$`C` har to korteste fælles super-sekvenserer `AC` og `CA`.

Spørgsmål b

i) $i = 0 \ \vee \ j = 0$

Hvis den ene streng er tom så udgør den anden streng den korteste fælles super-sekvens, d.v.s. $C(i, j) = \max\{i, j\}$

ii) $S[i] = S[j]$

Generelt findes der en kortest fælles super-sekvens der slutter med $S[i]$ eller $T[j]$ (ellers kunne vi undlade sidste symbol, og opnå en kortere fælles super-sekvens).

iii) Det er muligvis ikke ligegyldigt om korteste fælles super-sekvens slutter med $S[i]$ eller $T[j]$, derfor tages minimum af de to tilfælde

Spørgsmål c

for $i = 0..n$ do $C[i, 0] = i$
for $j = 1..m$ do $C[0, j] = j$
for $i = 1..n$ do
    for $j = 1..m$ do
        if $S[i] = T[j]$ then $C[i, j] = 1 + C[i - 1, j - 1]$
        else $C[i, j] = 1 + \min(C[i - 1, j], C[i, j - 1])$
return $C[n, m]$

De to sidste næstede for-løkker dominerer udførselstiden og tager tid $O(nm)$

Spørgsmål d

Efter at have beregnet $C[i, j]$-tabellen udfyldes et array $U[1..t]$ hvor $t = C[n, m]$

Algoritme

$j = m$
$i = n$
for $k = C[n, m]..1$ do
    if $j > 0 \wedge C[i, j] = C[i, j - 1] + 1$ then $U[k] = T[j], j = j - 1$
    else if $i > 0 \wedge C[i, j] = C[i - 1, j] + 1$ then $U[k] = S[i], i = i - 1$
    else $U[k] = S[i]; i = i - 1; j = j - 1$
return U

Sommeren 2003, opgave 4

Spørgsmål a

$n = k^2$

$m = 2k(k-1) + (k-1)k = 3k(k-1)$

Spørgsmål b

Dijkstra's algoritme tager tid $O(m \log n)$, d.v.s. $O(k^2 \log(k^2))$ som er $O(k^2 \log k)$

Spørgsmål c

Det ses at være tilstrækkeligt at finde alle korteste veje fra s til knuder i række r givet alle korteste vej til knuder i række $r-1$ allerede er kendt.

Da alle vægte er ikke-negative er der altid en korteste vej uden cykler. Der findes derfor en korteste vej til en knude $v_{r,t}$ fra række $r-1$ på formen

$$v_{r-1,j}, v_{r,j}, v_{r,j+1}, v_{r,j+2}, \ldots, v_{r,t} \text{ for et } j \leq t$$

eller på formen

$$v_{r-1,j}, v_{r,j}, v_{r,j-1}, v_{r,j-2}, \ldots, v_{r,t} \text{ for et } j > t.$$

Det er derfor tilstrækkeligt at *relaxere* (side 343 i GT) hver kant én gang: først de vertikale kanter mellem række $r-1$ og r. Dernæst alle højrerettede kanter fra venstre mod højre og til sidst alle venstrerettede kanter fra højre mod venstre.

Algoritme

$D[s] = 0$
forall $v \neq s$ do $D[v] = \infty$
for $j = 1$ to $k-1$ do relax($v_{1,j}$, $v_{1,j+1}$)
for $i = 2$ to k do
    for $j = 1$ to k do relax($v_{i-1,j}$, $v_{i,j}$)
    for $j = 1$ to $k-1$ do relax($v_{i,j}$, $v_{i,j+1}$)
    for $j = k-1$ downto 1 do relax($v_{i,j+1}$, $v_{i,j}$)

Da hver kant højest relaxeres én gang er tiden $O(m)$.

Spørgsmål d

De samme betragtninger som i 4C gælder stadig med én modifikation: I de vandrette rækker skal hver kant relaxeres *to gange*, idet en kortestes sti (fra række $r-1$ til række r) kan indeholde kanten $(v_{r,k}, v_{r,1})$ eller $(v_{r,1}, v_{r,k})$ og have en af formerne

$$v_{r-1,j}, v_{r,j}, \ldots, v_{r,1}, v_{r,k}, v_{r,k-1}, \ldots, v_{r,t} \text{ for et } j < t$$

eller

$$v_{r-1,j}, v_{r,j}, \ldots, v_{r,k}, v_{r,1}, v_{r,2}, \ldots, v_{r,t} \text{ for et } j > t$$

Algoritme

$D[s] = 0$
forall $v \neq s$ do $D[v] = \infty$
for $j = 1$ to $k-1$ do relax($v_{1,j}$, $v_{1,j+1}$)
relax($v_{1,1}$, $v_{1,k}$)
for $j = k-1$ downto 2 do relax($v_{1,j+1}$, $v_{1,j}$)
for $i = 2$ to k do
    for $j = 1$ to k do relax($v_{i-1,j}$, $v_{i,j}$)
    for $j = 1$ to $k-1$ do relax($v_{i,j}$, $v_{i,j+1}$)
    relax($v_{i,k}$, $v_{i,1}$)
    for $j = 1$ to $k-1$ do relax($v_{i,j}$, $v_{i,j+1}$)
    for $j = k-1$ downto 1 do relax($v_{i,j+1}$, $v_{i,j}$)
    relax($v_{i,1}$, $v_{i,k}$)
    for $j = k-1$ downto 1 do relax($v_{i,j+1}$, $v_{i,j}$)

Da hver kant højest relaxeres to gange bliver tiden $O(m)$.