AD vejledende løsninger

David Lolck og Lukas Retschmeier

2024

Dette dokument kan bruges til at checke egne løsninger til opgaver stillet i kurset, og til at se eksempler på hvordan svar kan formuleres. Feedback (fx om trykfejl, uklarheder) er velkommen: Skriv gerne til Lukas Retschmeier (lure@di.ku.dk).
Opgaverne fra den fjerde udgave af CLRS er markeret med *CLRS4*, opgaver fra tredje udgave med *CLRS3*.

CLRS Løsninger

CLRS4 3-4

a) Forkert! Vi sætter $f(n) = n$ og $g(n) = n^2$.

Tag $c = 1, n_0 = 1$, så gælder for alle $n \geq n_0$: $n \leq n^2$ og dermed er $f(n) \in \mathcal{O}(g(n))$ triviel.

På den anden side antager vi at $g(n) \in \mathcal{O}(f(n))$ gælder. Vi antager igen et der findes $c > 0, n_0 > 0$ så $n^2 \leq c \cdot n$ gælder for alle $n > n_0$. Vi dividerer begge sider med n og får $n < c$, men det kan ikke være sandt for alle $n > n_0$. $\Rightarrow$ Modstrid.

c) Korrekt! Vi skal vise at hvis $f(n) = O(g(n))$, og $\lg g(n) \geq 1$ og $f(n) \geq 1$ for alle tilstrækkeligt store n, så er $\lg f(n) = O(\lg g(n))$.

Observer hvis $f(n) = O(g(n))$, så eksisterer der et $c > 0$ således at $f(n) \leq cg(n)$ for tilstrækkeligt store n. Tages $\lg$ på begge sider gives der

$$\lg f(n) \leq \lg cg(n) = \lg c + \lg g(n)$$

Det gælder at $\lg f(n) \geq 0$ siden $f(n) \geq 1$. Siden $\lg g(n) \geq 1$, så har vi at

$$\lg c + \lg g(n) \leq \lg c \lg g(n) + \lg g(n) = (1 + \lg c) \lg g(n),$$

som ønsket

e) Forkert! Modeksempel: $f(n) = 1/n$ og $f(n)^2 = 1/n^2$. For $c > 0$ ved vi at $1/n < c/n^2 \Leftrightarrow 1/c < 1/n$ som ikke gældes for alle $n \geq n_0 > 0$.

CLRS4 3-1

d)

Vi skal vise at for ethvert polynomium $p(n)$ med grad d, hvor

$$p(n) = \sum_{i=0}^{d} a_i n^i$$

og for alle $k > d$, så gælder det at $p(n) = o(n^k)$. Med andre ord, for alle $c > 0$, og tilstrækkeligt store n er $p(n) < cn^k$. Lad $\alpha = \sum_{i=0}^{d} a_i$. Da er

$$p(n) = \sum_{i=0}^{d} a_i n^i \leq \sum_{i=0}^{d} a_i n^d = \alpha n^d < \frac{c}{\alpha} n^{k-d} \alpha n^d = cn^k$$

for tilstrækkeligt store værdier af n. Den sidste ulighed følger af at $n^\epsilon = \omega(1)$ for $\epsilon > 0$, og at $k - d > 0$, og derved at $\frac{c}{\alpha} n^{k-d} > 1$.

e)

Vi skal vise at for ethvert polynomium $p(n)$ med grad d, hvor

$$p(n) = \sum_{i=0}^{d} a_i n^i$$

og for alle $k < d$, så gælder det at $p(n) = \omega(n^k)$. Med andre ord, for alle $c > 0$, eksistere der et n_0 således at for alle $n > n_0$ er $p(n) \geq cn^k$. Da er

$$p(n) = \sum_{i=0}^{d} a_i n^i \geq a_d n^d = \frac{a_d}{c} n^{k-d} cn^d > cn^k$$

for tilstrækkeligt store værdier af n. Den sidste ulighed følger af at $n^\epsilon = \omega(1)$ for $\epsilon > 0$, og at $k - d > 0$, og derved er $\frac{a_d}{c} n^{k-d} > 1$

CLRS4 4.3-1

Ideen er lige for alle delopgaver. Vi markerer brug af induktionshypotesis med **i.h.**.

b)

$T(n) = T(n/2) + \Theta(1)$ Vi observerer at $T(n/2)$ halvere n i hvert funktionsopkald og vi bruger kun konstant tid (uafhængig af n) per trin.

Derfor giver en gæt af $\mathcal{O}(\lg n)$ mening. Vi viser at $T(n) \leq c \lg n$ for et $c > 0$ og $n \geq 2$. vi antager at $T(1) = \Theta(1)$.

Basis $T(2) = T(1) + \Theta(1) = 2\Theta(1) \leq c \lg 2 = c$. Hvis vi sætter c afhængig af den skjulte konstanter i $\Theta(1)$ er vi færdig.

Induktionstrin Antager at $T(n') < c \lg n'$ gælder for alle $n' < n$. Så har vi $T(n) = T(n/2) + \Theta(1) \overset{\text{i.h}}{\leq} c \lg(n/2) + \Theta(1) = c \lg n - c \lg 2 + \Theta(1) \leq c \lg n$. I den sidste ulighed antager vi at $c \lg 2 \geq \Theta(1) > 0$.

f)

$T(n) = 4T(n/2) + \Theta(1)$ har løsning $T(n) = \Theta(n^2)$.
Vi er nødt til at sætte både nedere og øvere grænser.

Nedere grænse: Vi går ud fra at $T(n) \geq cn^2$ gælder og derfor får vi

$$\begin{aligned} T(n) &= 4T(N/2) + \Theta(1) \\ &\overset{\text{i.h.}}{\geq} 4cn^2/4 + \Theta(1) \\ &= cn^2 + \Theta(1) \\ &\geq cn^2 \end{aligned}$$

Observere at et direkte bevis af den øvre grænse ikke virker hvis vi antager at $T(n) \leq cn^2$: *Øvere grænse*: $T(n) \leq cn^2$ og vi får $T(n) = 4T(n/2) + \Theta(1) \overset{\text{i.h.}}{\leq} c4n^2/4 + \Theta(n) = cn^2 + \Theta(n) \not\leq cn^2$ som kan ikke gælde for noget $c \geq 0$. Den følgende teknik kan også findes i CLRS.
Vi antager at $T(n) \leq c(n-a)^2$ for et a. Så har vi

$$\begin{aligned} T(n) &= 4T(n/2) + \Theta(1) \\ &\overset{\text{i.h.}}{\leq} 4c(n/2 - a)^2 + \Theta(1) \\ &= c(n^2 - 2 \cdot 2 \cdot an + 2^2a^2) + \Theta(1) = c(n-2a)^2 + \Theta(1) \\ &\leq c(n-a)^2 \end{aligned}$$

Det sidste ulighed gælder fordi vi ved at $\Theta(1) \leq (n-a)^2 - (n-2a)^2 \leq 2an$ for store n hvis vi sætter a tilsvarende.

CLRS4 4.4-4

Vi kan tegne et rekursionstræ som vist i fig. 1. Som forklaret i CLRS antager vi, at $T(n)$ termineres hvis $n \leq 1$ og den tager tid $\Theta(n)$. Bemærk at α kontrollerer hvor mange elementer kommer til den venstre eller højre side. Som eksempel er rekursionstræet balanceret hvis $\alpha = 1/2$.

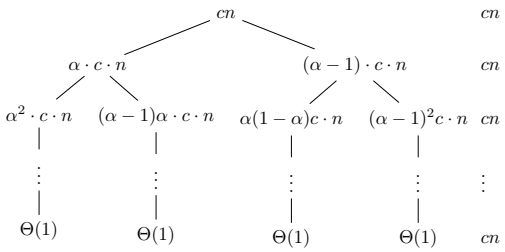


Figure 1: Rekursionstræet for $T(\alpha n)$ and $T((1-\alpha)n)$. Højden er $\log_{1/\alpha} n$ og ethvert level tager linear tid.

Antag at $\alpha \leq 1/2$ ($\alpha > 1/2$ er symmetrisk) og vi kan se at længste vejen ned til et blad (= højde af træet) er at altid følge $T((1-\alpha)n)$-vejen. Så er højden k:

$$\begin{aligned}(1-\alpha)^k n &= 1\\ \log_{1/\alpha}\left((1-\alpha)^k n\right) &= \log_{1/\alpha}(1) = 0\\ -k + \log_{1/\alpha} n &= 0\\ \log_{1/\alpha} n &= k\end{aligned}$$

Hele køretiden er summen af alle knuder plus nogen linear tid på blade.

$$\begin{aligned}T(n) &= \sum_{i=0}^{\log_{1/\alpha} n} cn + \Theta(n)\\ &= cn \log_{1/\alpha}(n) + \Theta(n)\\ &= \Theta(n \lg n)\end{aligned}$$

Det skal stadig bevises med induktion.

Ideskitse: Bevis både nedre grænse $T(n) \leq dn \lg n$ for et $d > 0$ og øvre grænse $T(n) \geq dn \lg n$.

$$\begin{aligned}T(n) &\leq T(\alpha n) + T((1-\alpha)n) + \Theta(n)\\ &\leq d\alpha n \lg(\alpha n) + d((1-\alpha)n \lg((1-\alpha)n)\\ &\vdots\\ &\leq dn \ln n\end{aligned}$$

for et d.

CLRS4 16.1-3

Vi er givet at for en sekvens af n operationer, så har den i'te operation følgende pris per operation

$$s_i = \begin{cases} i & \text{hvis der findes et } k \in \mathbb{Z} \text{ så } i = 2^k \\ 1 & \text{ellers} \end{cases}$$

Hvis vi tæller sammen den totale pris for den første type operationer $T_1(n)$, så får vi:

$$T_1(n) = 1 + 2 + 4 + \ldots + t = 2t - 1$$

Da vi ved at $t \leq n$, så følger det at $T_1(n) = O(n)$.

For prisen for de billige operationer $T_2(n)$, så følger det at $T_2(n) = n - r = O(n)$, hvor r er antallet af dyre operationer. I alt får vi at den amortiserede køretid for n operationer er $T_1(n)+T_2(n) = O(n)$, og derved er prisen for enkelte operationer amortiseret konstant.

CLRS4 16.2-2

Vi skal bruge accounting til den samme ptis på operationer som den forrige.

$$s_i = \begin{cases} i & \text{hvis der findes et } k \in \mathbb{Z} \text{ så } i = 2^k \\ 1 & \text{ellers} \end{cases}$$

Vi opsætter at for billige operationer, så er prisen $3 = O(1)$, mens den er uændret for de dyre. Dette giver et overskud for de billige operationer som så kan bruges til at betale for de dyre.

Observer at for en dyr operation $i = 2^k$, så gælder det at den forrige dyre operation var operationen 2^{k-1}. Mellem disse operationer er der derfor $2^{k-1} - 1$ billige operationer. Dette betyder at de skaber et overskud på $2(2^{k-1} - 1) = 2^k - 2$. Dette betyder at for den i'te operation skal der kun betales $2^k - (2^k - 2) = 2 = O(1)$.

Vi kan konkludere at prisen per operation er amortiseret konstant.

CLRS4 16.3-2

Vi gennemføre den samme opgave som de 2 forrige, dog denne gang med potentiale metoden.

Vi lader D_i være datastrukturen efter i operationer, og vi definere potentialefunktionen som

$$\Phi(D_i) = 2\left(i - 2^{\lfloor \lg i \rfloor} + 1\right)$$

Dette giver at $\Phi(D_0) = 0$, hvilket er hvad vi ønsker. Vi finder nu den amortiserede pris per operation $\hat{s}_i$. Det vides at

$$\hat{s}_i = s_i + \Phi(D_i) - \Phi(D_{i-1})$$

For billige operationer gælder det at $\lfloor \lg i \rfloor = \lfloor \lg i - 1 \rfloor$, så vi har

$$\hat{s}_i = 1 + 2\left(i - 2^{\lfloor \lg i \rfloor} + 1\right) - 2\left(i - 1 - 2^{\lfloor \lg i - 1 \rfloor} + 1\right) = 3$$

For dyre operationer, så har vi at $2^{\lfloor \lg i \rfloor} - 2^{\lfloor \lg i - 1 \rfloor} = i/2$. Derved har vi

$$\hat{s}_i = i + 2\left(i - 2^{\lfloor \lg i \rfloor} + 1\right) - 2\left(i - 1 - 2^{\lfloor \lg i - 1 \rfloor} + 1\right) = i - i + 2 = 2.$$

Altså har vi konstant amortiseret pris per operation.

CLRS3 19.3a): Ny Operation Fib-Heap-Change-Key

Vi vil gerne have den samme amortiserede kompleksitet som for decrease_key. En Ide er at kombinere operationer, som allerede er introduceret. Bemærk at vi kan øge en værdi ved først at sætte den midlertidigt som et minimum, derefter fjerne den og genindføre den med den korrekte værdi. Den samlede køretiden er simpelthen summen af hver operation: decrease_key bruger $\mathcal{O}(\log n)$ amortiseret, og resten er konstant.

Algorithm 1 Fib-Heap-Change-key

Require: Heap H, Node x, New Value v
  if $v <= x.key$ **then**
    decrease_key(H, x, v)
  else if $v > x.key$ **then**
    decrease_key(H, x, H. min -1) amortiseret $\mathcal{O}(\log n)$
    delete_min(H) a. $\mathcal{O}(1)$
    insert(H, v) a. $\mathcal{O}(1)$
  end if

CLRS3 19.4-1

Vi ønsker en sekvens af operationer, der resulterer i et enkelt træ bestående af kun en sti. Det kan vi f.eks. (fig. 2) gøre på følgende måde:

1. **Initialisere** en ny hob, tilføje tre elementer og kalde extract_min. Det vil give dig en simpel sti med to elementer.
2. Derefter kan vi **udvide** stien med et element hvis vi
   (a) tilføjer tre nye elementer hvor to er mindre end roden, og derefter
   (b) kalder extract_min.

Hvis vi ser bort fra antallet af operationer, kunne vi også først skabe et stort træ og derefter skære de enkelte blade af. Her er en god interaktiv visualisering: https://people.ksp.sk/~kuko/gnarley-trees/Fibonacci.html

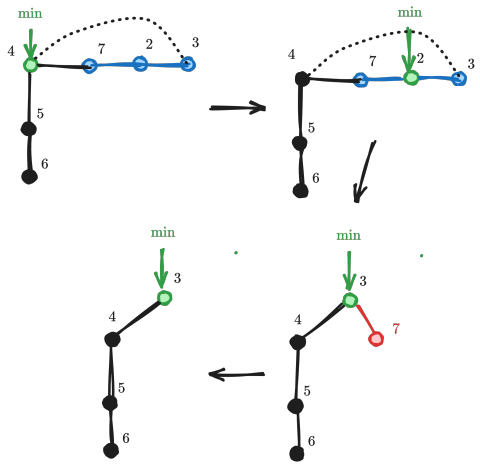


Figure 2: Skabe en enkel liste af lengden n

CLRS3 19.1

a) Kørtiden er ikke konstant fordi den er afhængig af antallet af børn fra x.

b) $\mathcal{O}(x.degree + c)$

c) Husk på, at vores potentialfunktion er $\Phi(H) = t(H) + 2 \cdot m(H)$ hvor $t(H)$ angiver antallet af træer og $m(H)$ antallet af markerede knuder i H. Kun CASCADING-CUT ændrer antallet af markerede knuder $m(H')$, og hvert opkald af CASCADING-CUT fjerner markeringen af en knude. Det sidste opkald kan markere en ny knude. Derfor $m(H') \leq m(H) - (c-1) + 1 = m(H) - c + 2$ (c er antallet af CASCADING-CUT opkaldene), og potentialet øges med $x.degree$:

$$\begin{aligned}\Phi(H') &= (t(H) + 2m(H)) + x.degree + 2 \cdot (-c + 2) \\ &= t(H) + x.degree + 2(m(H) - c + 2)\end{aligned}$$

d) Vi ved at $x.degree = \mathcal{O}(\lg n)$, og dermed har PISANO-DELETE den samme asymptotiske kørtid som FIB-HEAP-DELETE.

CLRS4 14.1-2

Table 1 giver et modeksempel.

	1	2	3	4
	1	6	10	1
d_i	1	9/3	10/3	1/4

Table 1: Modeksemplet 14.1-2. d_i er densiteterne

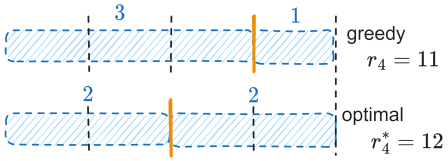


Figure 3: 14.1-2 Modeksempel. Den grådige algoritme vælger først et element med længden tre fordi den har det højeste densitet, og er derefter tvunget til at tage en blok med længden 1, hvilket ikke er optimalt.

CLRS4 14.1-3

Vi kan ændre vores algoritme og tilføje prisen pr. iteration. Desuden skal vi at være opmærksom når $i = j$. Det betyder at vi *bare* bruger **et** element ($p[i]$) til at dække alt, og vi har ikke noget snit. Derfor skal vi ikke trække c fra! Algoritme vises i algorithm 2.

Algorithm 2 Cut-With-Cost

Require: p, n, c
  Let $r[0:n]$ be array
  $r[0] = 0$
  for $j = 1$ **to** n **do**
    $q = p[j]$
    for $i = 1$ **to** $j - 1$ **do**
      $q = \max\{q, p[i] + r[j - i] - c\}$
    end for
    $r[j] = q$
  end for

CLRS4 14.3-5

fig. 4 giver en modeksempel hvor et subproblem kan ikke bliv genbrugt, fordi alle"2"s er allerede er i brug.

CLRS4 14.4-1

F.eks $(1, 0, 1, 0, 1, 0)$.

	1	2	3	4	5
p_i	1	5	1	8	1
l_i	∞	2	∞	∞	∞

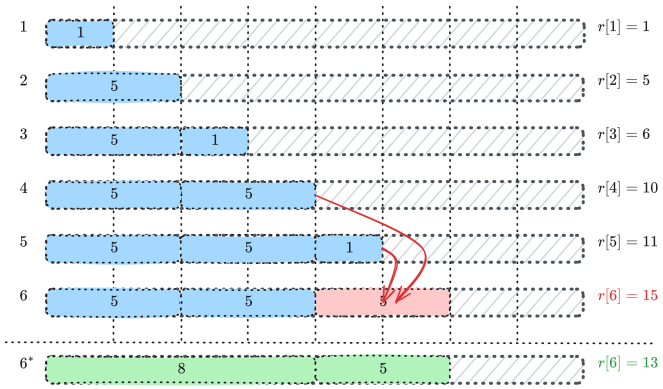


Figure 4: 14.3.5: At bruge et subproblem er ikke altid optimalt.

CLRS4 14.4-5

Lad $a \in \mathbb{R}^n$ være en sekvens. Vi bruger dynamisk programmering til at finde den længste monoton voksende delsekvens. Lad l_i være en matrix som gemmer løsningerne til den længste monoton voksende delsekvens af længden i. Derfor beregner vi l_i for alle $i \leq n$:

$$l_i = \begin{cases} 1 & \textbf{if } i = 1 \textbf{ eller } a_j > a_i \forall j < i \\ \max\limits_{1 \leq j < i, a_j \leq a_i} (l_{i-j}) + 1 & \textbf{ellers} \end{cases}$$

Kompleksiteten er igen $\mathcal{O}(n^2)$, fordi per i vores algoritme undersøger $i - 1$ delproblemer.

CLRS4 12.1-5

Vi antager at sammenligningsbaseret sortering er $\Theta(n \lg n)$. Hvis vi kunne generere et binært søgetræ i lineær tid, kunne vi udskrifter træet med en in-order algoritme som kører i $\mathcal{O}(n)$. Outputtet ville være sorteret, og samlede køretiden er kun $\mathcal{O}(n)$, hvilket er en modsigelse.

CLRS4 12.3-5

Se fig. 5.

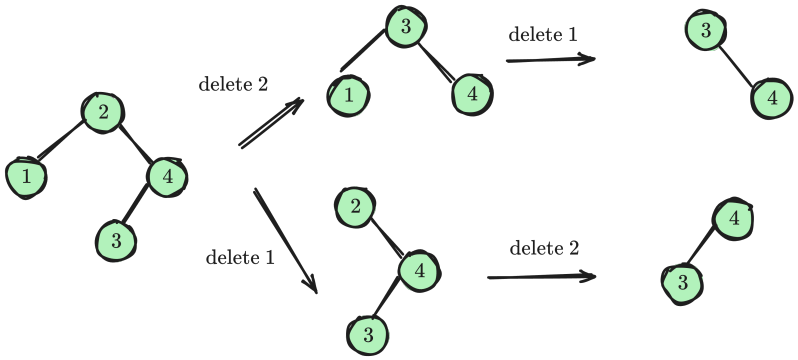


Figure 5: Modeksempel 12.3-5. Delete på binære søgetræer er ikke kommutativ.

CLRS4 13-3 AVL Træer

a) Vi beviser først at et AVL træ har højden $\mathcal{O}(\log n)$. Derfor viser vi at hver deltræ af højden h har mindst F_h mange knuder som en nedre grænse. (F_h er den hst fibonacci tal) med induktion over h. Vi kalder antallet af knuder i en deltræ med roden x som $s(x)$.
Basis: trivielt
Induktionstrin: Vi antager at den gælder for et træ af højden h og vi kalder roden x, xs børn $x.l$ og $x.r$. Desuden ved vi at $F_{n+1} = F_n + F_{n-1}$ per definition.

1. Hvis $h(x.l) = h(x.r) = h(x) - 1 \Rightarrow s(x) = 1 + \cdot s(x.l) + s(x.r) \geq 2 \cdot F_{h-1} \geq F_h$
   Den sidste ulighed holder fordi $F'_h = F_{h'-1} + F_{h'-2} \leq F_{h'-1} + F_{h'-1} = 2F_{h'-1}$ holder for alle h', og F er en monoton voksende funktion.
2. Hvis $h(x.l) = h - 1 \wedge h(x.r) = h - 2 \Rightarrow s(x) \Rightarrow 1 + s(x.l) + s(x.r) \geq 1 + F_{h-1} + F_{h-2} = 1 + F_h \geq F_h$
3. Symmetrisk til fald (2).

Vi kender en lukket form af fibonacci tallerne fra CLRS: $F_n = \lfloor \frac{\phi^n}{\sqrt{5}} + 1/2 \rfloor$. Alternativt kan vi se at $F_n = F_{n-1} + F_{n-2} \geq 2F_{n-2} \geq 2 \cdot 2 \cdot F_{n-4} \geq \cdots \geq 2^{\lfloor n/2 \rfloor}$, og dermed $F_n = \Omega(2^{n/2})$.

Lad h højden af hele træet. Så har vi $n \geq F_h$ fra første delen, og efter vi har løst denne her uligheden, får vi $h \in \mathcal{O}(\log n)$

b) Udladt.

d) Vi har bevist at højden er $\mathcal{O}(\log n)$ og vi har bare en konstant tal af roteringer per niveauet.

CLRS4 21.1-1

Lad G en sammenhængende graf, og $e = \{u, v\}$ en kant med minimal vægt. Antag at e ikke tilhøre noget mst. Vælg to forskellige kanter $\{u, u'\}$, $\{v, v'\}$ til både u og v som er del af et mindst udspændende træ og som er en del af stien mellem u og v (se også fig. 6). Fordi e er har mindre vægt end alle andre kanter kan vi udvskifte $\{v, v'\}$ med e og får et bedre mst. Modstrid.

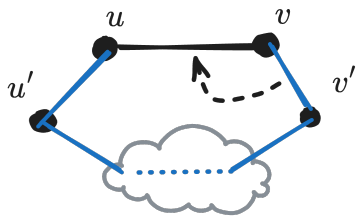


Figure 6: 21.1-1

CLRS4 21.1-3

Lad os antage, at $e = \{u, v\}$ er en del af et mst M. Lad L, R være de sammenhængende komponenter der resultere i at fjerne e fra M. Observer at siden L og R er sammenhængende, så vil vi kunne få et udspændende træ ved at erstatte e med en hvilken som helst anden kant krydser snittet (L, R). Derved må e have minimal vægt blandt alle kan kanter som krydser snittet da den tilhøre et mindst udspændende træ. Vi kan konkluderer at e er "light" Figure 7 skitser beviset.

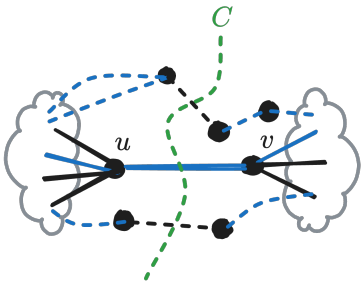


Figure 7: 21.1-3

CLRS4 21.1-5

Se fig. 8. Antag at $e \in$ MST og se på cyklen C. Lad (L, R) være de sammenhængende komponenter der resulterer i at fjerne e fra det mindst udspændende

træ. Siden C er en cykel, og e krydser (L, R) så eksisterer der mindst en kant $e' \neq e$ således at $e' \in$ MST og e' krydser (L, R). Da e har størst vægt i cyklen kan vi erstatte e med e' uden at foreværre vores MST.

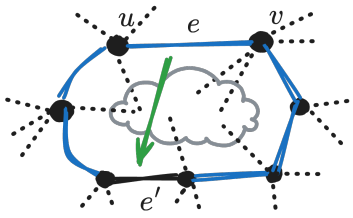


Figure 8: 21.1-3

CLRS4 21.2-4

Hvis vi ved at alle vægte er fra $[|V|]$, kan vi sortere i linær tid. (e.g. count sort) og derefter køre Kruskals algoritmen.

Kompleksiteten er derfor $\mathcal{O}(|E| + |E|\alpha(n)) = \mathcal{O}(|E|\alpha(n))$ hvor $\alpha(n)$ er *inverse ackermann* funktionen i union-find.

CLRS4 21.2-6

Forkert! Se fig. 9.

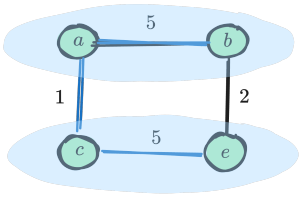


Figure 9: Modeksempel 21.2-6 Modeksempel

CLRS4 21-3

a) Er sand, fordi algoritmen kun fjerner kanter e som er del af en cykel c i T. T er fortsat forbundet uden e. Da vi kun fjerner de største kanter i cyklen, har alle andre kanter på c samme eller mindre vægt, og vi ved at der eksisterer mindst et andet mst uden e. (Se også opgaven 21.1-5 for det centrale argument)

b) Er forkert. Se fig. 10 for et modeksempel.

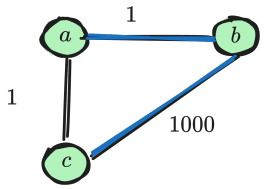


Figure 10: Modeksempel 21.3 b). Den vilkårlige rækkefølge kan give forkerte løsninger.

c) Er sand, fordi algoritmen kun har fjernet maximale kanter e på en cykel efter termineringen og der altid er en anden mst, der ikke bruger e (se også 21.1-5). Da vi fjerner kun kanterne af hver cykel i grafen, vil vi få en træ til sidst.

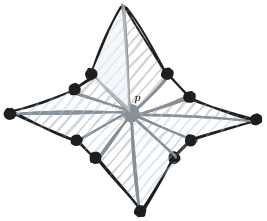
CLRS3 33.3-4

Flaskehalsen i Graham's Scan er sorteringen som tager $\mathcal{O}(n \log n)$. Hvis vi kunne springe det over, ville algoritmen kun behøve lineær tid. Vi vil snart se at for en star-shaped polygon, kan vi gøre det!

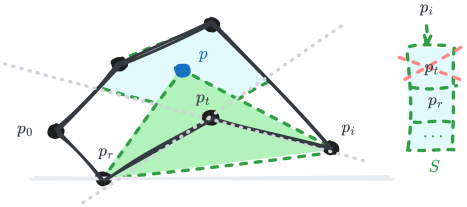
Vi antager, at kanterne af den givne polygon $P = (p_0, p_1, \cdots, p_n)$ er sorteret mod uret. Dette definerer rækkefølgen, som Graham-scanningen også bruger til at gå rundt om kanterne. Et præcist bevis følger den samme logik som beviset i bogen. Men den centrale ide er, at vi bruger star-shaped egenskaben hver gang vi fjerner en kant fra stakken og viser, at det er sikker fordi den nye kant forbedrer løsningen.

Antag at S er en stak, $(p_t, p_r, ..., p_0)$ er $S's$ de øverste kanter, og p_i er den nye kant i skridt i. Hvis algoritmen fjerner p_t fra stakken, er $\overline{p_r p_t}$ og $\overline{p_t p_i}$ et sving til højre. Lad p være et vilkårligt punkt i *kernen* (eng. *kernel*) af P. Vi bemærker den følgende: Unionen af alle trekanter $\triangle p_i p_{i+1} p$ giver præcist arealet af P og vi ved at alle disse trekanter må umsluttet af den konvekse hylde (se fig. 11a).

Vi ser, at $\triangle p_r p_i p$ indeholder $\triangle p_t p_r p$ og dermed også p_t (eller p_r ligger direkt på grænsen) i skidt i. Det betyder at p_t ikke er en del af den konvekse hylste og det er sikkert at fjerne p_t fra S. Vi gentage observeringen til alle punkter som bliver fjernt fra S i sløjfen. Du finder en skitse i fig. 11b.



(a) 33.3-4: Dekomponerer arealet af polygonen P.



(b) 33.3-4: $\triangle p_r p_i p$ umslotter p_t

CLRS3 33.3-5

Vi giver en $\mathcal{O}(n^2)$-tid online algoritme ved at opdatere et konvekst hylster hver gang der tilføjes en ny knude i lineær tid. Lad $C_{t-1} = \{p_1, p_2, ...\}$ den gamle konvekse hylster som er sortered mod uret, og p_t den nye punkt på tidpunktet t. Først tjekker vi om p_t allerede ligger i C_{t-1} med brug af krydsprodukten: Det sker hvis alle linjestykker $\overline{p_i p_{i+1}}$ og $\overline{p_i p_t}$ er et sving til venstre. (Det tager kun linear tid). Hvis C_{t-1} indeholder p_t, er vi færdige.

Nu antager vi, at p_t ligger uden fra C_{t-1} og vi observere: Hvis en $\overline{p_i p_{i+1}}$ og $\overline{p_i p_t}$ svinger til venstre, må p_{i+1} stadig være en del af C_t.

Derfor kan vi gennemgå C_t mod uret indtil vi finder en knude p_{i+1} hvor $\overline{p_i p_{i+1}}$ og $\overline{p_{i+1} p_t}$ svinger første gang til højre. Nu søger vi den første indeks j efter i hvor $\overline{p_t p_j}$ og $\overline{p_t p_{j+1}}$ svinger til venstre og sætter C_t. Hele algoritme er kun en lineær scanning over højst n knuder og derfor er samlet køretiden på n nye knuder $\mathcal{O}(n^2)$.

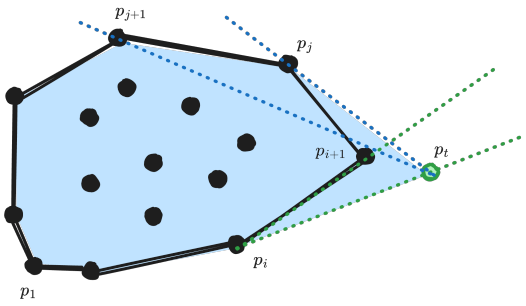


Figure 12: 33.3-5 Online Konveks Hylde

CLRS4 22.1-2

Vi skal bevise Korollar 22.3, der siger:

Corollary 0.0.1. *Let $G = (V, E)$ be a weighted, directed graph with source vertex s and weight function $w : E \to \mathbb{R}$. Then, for each vertex $v \in V$, there is a path from s to v if and only if* `BELLMAN-FORD` *terminates with $v.d < \infty$ when it is run on G.*

Proof. Vi skal her vise begge veje af "if and only if".

$\Rightarrow$: Det er givet at der eksistere en sti fra s til v. Per Lemma 22.2 vil det da gælde at $v.d = \delta(s, v) < \infty$.

$\Leftarrow$: Lad $p(v)$ være den første knude der opdatere $v.d$ for v gennem en `Relax` operation. Det holder desuden at siden hver knude x starter med $x.d = \infty$, så må det gælde at en knude x forskellig fra s kun kan opdatere en knude y hvis den selv er blevet opdateret tidligere i programmet. Observer nu grafen givet af knuderne V og kanterne givet ved $p(v)$. Der er tale om en acyclisk orienteret graf per algoritme eksekveringen. Dette betyder at vi har et rodfæstet træ, da

den eneste knude der kan være en rod er knuden s. Siden s er roden og v er indeholdt i træet medføre det at der eksisterer en sti fra s til v. □

CLRS4 22.1-3

Hvis vi kigger på beviserne for korrektheden af `BELLMAN-FORD`, så bygger de på at den korteste vej maksimalt har $|V| - 1$ kanter. Dette medføre at løkken skal gentages $|V| - 1$ gange. Hvis alle korteste veje derimod højest har m kanter, så ved vi at løkken kun ville behøves at gentages m gange for at opnå det korrekte resultat. Dette giver desuden også at efter m iterationer af den yderste løkke vil $v.d$ værdierne ikke ændre sig. Derfor kan man blot holde et boolsk værdi, der fortæller om man opdaterede nogle $v.d$ værdier. Hvis man ikke opdaterede nogen kan man blot terminere algoritmen, da man nu har udført m ændringer.

CLRS4 22.3-2

Se følgende eksempel: