

Introduktion til algoritmik og datastrukturer

Vejledende løsninger

IT-højskolen i København

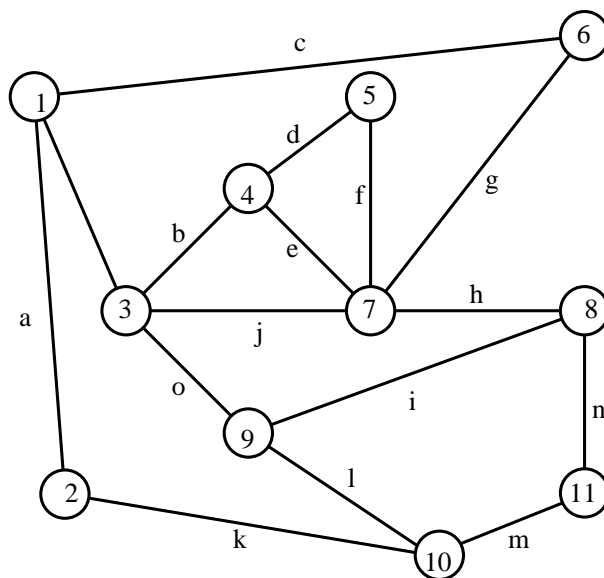
7. Januar 2000

Dette eksamenssæt består af 3 opgaver med tilhørende delopgaver, der tilsammen udgør 100 point. Delopgaverne i hver opgave vægtes ens. Du har i alt 4 timer til din rådighed. Husk at angive sidetal, navn og studienummer på alle sider i din besvarelse. Eksamensættet består af 9 nummererede sider.

CLR refererer til "Introduction to Algorithms" af Cormen, Leiserson og Rivest, 18. tryk, 1997.

Opgave 1 (30 point) Grafer og grafalgoritmer

Følgende opgave omhandler konstruktion af det mindst udspændende træ (minimum spanning tree) for en sammenhængende, uorienteret graf. Betragt den vægtede graf nedenfor. Hver knude er navngivet med et tal fra 1 til 11, og hver kant er navngivet med et bogstav fra *a* til *o*. I tabellen under grafen er hver kants vægt specificeret. F.eks. har kanten *j* vægten 13. Som bekendt kan man anvende Kruskals algoritme (CLR 24.2) til at finde det mindst udspændende træ for en vægtet graf.



Kant	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
Vægt	3	15	12	8	2	5	6	14	4	13	7	9	10	1	11

a) Angiv hvilke kanter, der er med i det mindst udspændende træ for grafen i figuren ovenfor, samt den rækkefølge disse træ-kanter bestemmes af Kruskals algoritme.

Svar a) *h, e, a, i, f, g, k, l, o, c.*

I det følgende betragter vi en sammenhængende og uorienteret graf $G = (V, E)$. Vi lader $n = |V|$ betegne antallet af knuder i G . Vi kalder en graf for en *tynd graf*, hvis den kun har lineært mange kanter, dvs. $|E|$ er $O(n)$. I de følgende opgaver b), c) og d) antages det, at G er en sammenhængende uorienteret tynd graf, der er repræsenteret ved **adjacency-list** repræsentationen (CLR 23.1).

b) Angiv køretiden for at finde det mindst udspændende træ for den tynde graf G ved hjælp af Kruskals algoritme.

Svar b) $O(n \lg n)$.

c) Betragt k kanter $e_1, e_2, \dots, e_k \in E$ i den tynde og sammenhængende graf G . Beskriv en algoritme, der afgør, om der findes et udspændende træ, der indeholder kanterne e_1, \dots, e_k , og i givet fald finder det mindst udspændende træ med disse kanter e_1, \dots, e_k . Angiv udførelsestiden for algoritmen.

Svar c) $E' = e_1, e_2, \dots, e_k \in E$ er indeholdt i et udspændende træ hvis og kun hvis grafen $G' = (V, E')$ ikke indeholder kredse. Dette kan undersøges ved hjælp af et dybde-først gennemløb af G' i $O(n)$ tid.

d) Antag, at alle kant-vægtene for G har en værdi mellem 1 og $3n$. Beskriv en algoritme, der finder det mindst udspændende træ for den tynde graf G i tiden $O(n \lg(\lg n))$. Det kan frit udnyttes, at enhver sekvens på $O(n)$ operationer (MAKE-SET, UNION og FIND-SET) for **disjoint-set** data strukturen kan udføres i tid $O(n \lg(\lg n))$.

Svar d) Vi anvender COUNTING-SORT til sorteringen i Kruskals-algoritme. Den samlede tid bliver da $O(n \lg \lg n)$.

Opgave 2 (40 point) Lister og søgning

Denne opgave omhandler bl.a. lister, søgning og halveringsteknik. Opgaven tager udgangspunkt i hægtede lister, som beskrevet i CLR 11.2

Lad L være en enkelthægtet liste (engelsk “singly linked list”). Lad $head[L]$ betegne pegeren på det første element i listen. Hvis $head[L] = \text{NIL}$, er listen tom. Hvert element har en peger, $next$, som peger på det næste element i listen. Hvis x er det sidste element i listen, så er $next[x] = \text{NIL}$. Hvert element har ligeledes et key felt, som er et tal. Det antages, at key felterne i listen er sorteret i stigende orden. Det vil sige, at for to nabo-elementer x og $y = next[x]$ i listen gælder det, at $key[x] \leq key[y]$.

Betragt følgende procedure TEST:

TEST(L, K)

1. $x \leftarrow head[L]$
2. **while** $x \neq \text{NIL}$ and $key[x] < K$
3. **do** $x \leftarrow next[x]$
4. **if** $x = \text{NIL}$
5. **then return** FALSE
6. **if** $key[x] = K$
7. **then return** TRUE
8. **else return** FALSE

a) Beskriv, hvad procedure TEST gør, såfremt L er sorteret, og angiv kompleksiteten for procedure TEST.

Svar a) TEST undersøger om der findes et element med nøglen K i L .

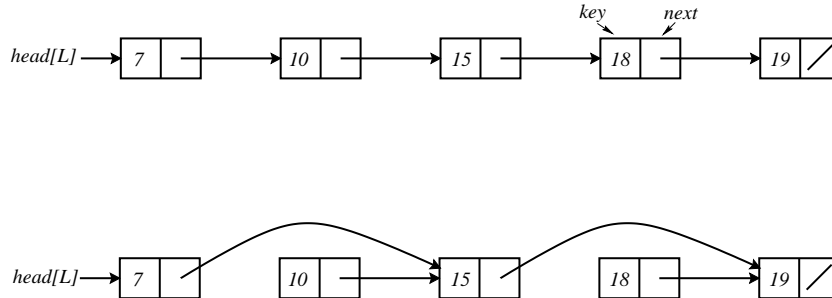
Betragt følgende to procedurer LÆNGDE og BYT:

LÆNGDE(L)

1. $x \leftarrow head[L]$
2. $k \leftarrow 0$
3. **while** $x \neq \text{NIL}$
4. **do** $k \leftarrow k + 1$
5. $x \leftarrow next[x]$
6. **return** k

BYT(L)

1. \triangleright Det antages, at proceduren LÆNGDE(L) returnerer et positivt, ulige heltal.
2. $x \leftarrow head[L]$
3. **while** $next[x] \neq \text{NIL}$
4. **do** $next[x] \leftarrow next[next[x]]$
5. $x \leftarrow next[x]$



I figuren ses et eksempel på en liste med 5 elementer før og efter et kaldt til BYT.

I den resterende del af opgaveteksten betragtes udelukkende lister med n elementer, hvor n kan skrives som $2^k + 1$ for et heltal $k \geq 0$. Det vil sige tallene 2, 3, 5, 9, 17, 33, ...

b) Lad L være en liste med $n = 17 = 2^4 + 1$ elementer. Først kaldes procedure BYT(L), og dernæst kaldes procedure LÆNGDE(L). Hvad returnerer procedure LÆNGDE(L)?

Svar b) Længde returnerer $n + 1/2 = 18/2 = 9$.

c) Lad L være en liste med $n = 2^k + 1$ elementer for et $k \geq 1$. Hvor mange gange skal procedure BYT(L) kaldes, før et kald til procedure LÆNGDE(L) returnerer 2?

Svar c) k .

Lad L være en enkelt hæftet liste med key og $next$ felt som beskrevet i ovenstående. Hvert element i listen lader vi nu endvidere have en tabel,

hop, af pegere. For en liste med $n = 2^k + 1$ elementer indeholder hoptabellen k pegere, $hop[1 \dots k]$. For ethvert element x er $hop[x][j]$ pegeren i den j 'te indgang i x 's hoptabel.

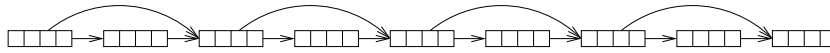
Betragt følgende procedure HOP1:

HOP1(L)

1. \triangleright Det antages, at proceduren LÆNGDE(L) returnerer et positivt ulige heltal.
2. $x \leftarrow head[L]$
3. **while** $next[x] \neq NIL$
4. **do** $hop[x][1] \leftarrow next[next[x]]$
5. $x \leftarrow hop[x][1]$

d) Givet er en liste L med $n = 9 = 2^3 + 1$ elementer. Antag, at $hop[x][i]$ for hvert x og i er initialiseret til NIL. Tegn datastrukturen efter et kald til procedure HOP1(L), hvor pegere, der er NIL, må udelades.

Svar d)



Betragt følgende to procedurer HOP og HOPTABEL:

HOP(L, i)

1. $x \leftarrow head[L]$
2. **while** $hop[x][i - 1] \neq NIL$
3. **do** $y \leftarrow hop[x][i - 1]$
4. $hop[x][i] \leftarrow hop[y][i - 1]$
5. $x \leftarrow hop[y][i - 1]$

HOPTABEL(L)

1. $n \leftarrow LÆNGDE(L)$
2. I nedestående antages det, at k er sat således, at $n = 2^k + 1$
3. **if** $k > 0$
4. **then** HOP1(L)
5. **if** $k \geq 2$

6. **then for** $i \leftarrow 2$ **to** k
7. **do** HOP(L, i)

e) Lad L være en liste med $n = 2^k + 1$ elementer for $k \geq 0$. Det antages, at $hop[x][i]$ for hvert x og i er initialiseret til NIL. Vis, at tidskompleksiteten for et kald til procedure HOPTABEL(L) er $O(n)$.

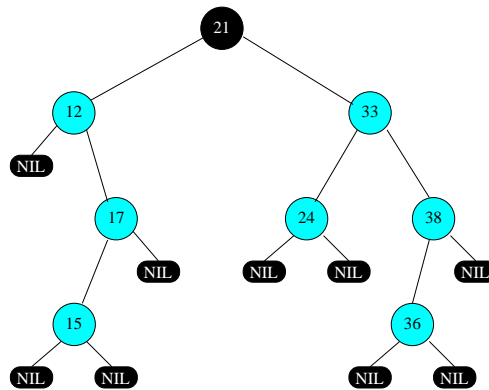
svaer e) Bemærk at HOP procedureren fordobler længden af det interval den springer hver gang. Tidskompleksiteten bliver følgelig $O(n/2 + n/4 + n/8 \dots) = O(n)$.

f) Lad L være en liste med $n = 2^k + 1$ elementer for $k \geq 0$, og med *key* felterne sorteret i stigende orden som beskrevet tidligere. Antag, at $hop[x][i]$ for hvert x og i er initialiseret til NIL, og at der dernæst er foretaget et kald til procedure HOPTABEL(L). Beskriv i ord, hvordan man kan konstruere en procedure NÆSTE(L, z), der returnerer et element, hvis *key* værdi er det mindste i listen, der er strengt større end z . Hvis et sådant tal ikke findes, skal programmet returnere -1 . Den beskrevne procedure NÆSTE skal have tidskompleksitet $O(k)$.

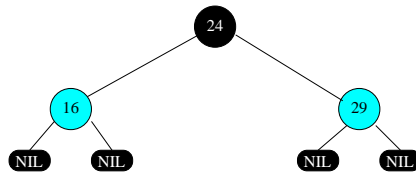
Svaer f) Ved hjælp af hoppegerne kan man udføre en binær søgning. Tiden bliver $O(\log n) = O(k)$.

Opgave 3 (30 point) Rød-sort-træer

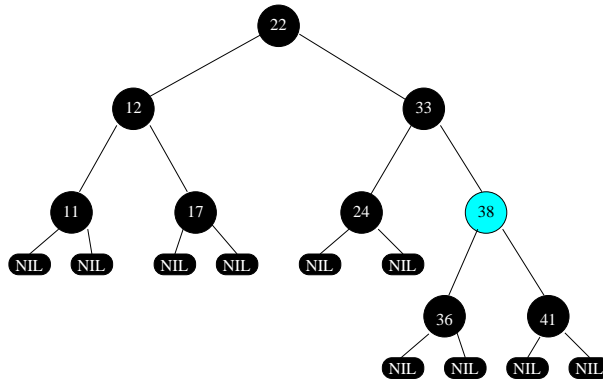
Denne opgave omhandler rød-sort-træer. Figurene, der hører til opgaven, følger konventionen i CLR. I besvarelsen skal knudernes farve tydeligt fremgå. For eksempel ved at skrive *r* for rød og *s* for sort ud for knuderne.



a) Træet T i figuren ovenfor overholder ikke egenskaberne for rød-sort-træer. Vis, hvordan T kan ændres og farvelægges uden at ændre roden, så træet overholder egenskaberne. Det modificerede træ skal indeholde de samme værdier som T .



b) Indsæt værdien 8 i rød-sort-træet i figuren med tre interne knuder ovenfor. Indsæt derefter værdien 13 i det træ, hvor 8 nu er indsat. Algoritmen til indsættelse i rød-sort-træer beskrevet i CLR skal benyttes. Beskriv resultatet ved at illustrere algoritmens virkemåde.



c) Slet værdien 22 i rød-sort-træet i figuren ovenfor. Algoritmen til at slette i rød-sort-træer beskrevet i CLR skal benyttes. Beskriv resultatet ved at illustrere algoritmens virkemåde.

d) Beskriv en algoritme $TÆL-SORT(r, v)$, hvor r er roden af et rød-sort-træ T , og v er en værdi. $TÆL-SORT$ returnerer antallet af sorte knuder i T , hvor værdien i knuderne er større end eller lig med v . Algoritmen skal have kompleksiteten $O(k + \lg n)$, hvor n er antallet af knuder i T , og k er antallet af knuder i T med værdi større end eller lig med v .

Svar d) $TÆL-SORT(r, v)$ kan implementeres ved først at lave en søgning efter r i T . Derefter finder vi alle knuder w , hvor $key[w] \geq r$ med kald til $TREE-SUCCESSOR$. Undervejs tæller vi antallet af sorte knuder. Søgningen tager $O(\lg n)$ og den samlede tid til $TREE-SUCCESSOR$ er $O(k)$.

e) Givet et rød-sort-træ T , hvor roden har black-height h . Hvad er det færrest mulige antal interne, sorte knuder, som T mindst skal indeholde? Hvad er det størst mulige antal interne sorte knuder, som T kan indeholde? Husk, at alle interne knuder har to børn.

Svar e) Minimalt skal T indeholde $2^{h-1} + 1$ interne sorte knuder. Dette opnås med et fuldstændigt binært træ af højden h . For at maximere antallet af sorte kan vi skiftevis farve hvert andet niveau af træet sort startende med bladene. Det maximale antal interne sorte knuder er dag $\sum_{i=0}^{h/2} 2^{2i}$ hvis h er lige og $1 + \sum_{i=0}^{h-1/2} 2^{2i+1}$ hvis h er ulige.

Svar a,b,c) Blade er undladt.

