Disjoint Sets

Partition

Reminder: en *Partition* (disjunkt opdeling) af en mængde S er en samling ikke-tomme delmængder A_i, $i = 1, \ldots, k$ som er disjunkte og tilsammen udgør S:

$$A_i \neq \emptyset \text{ for alle } i$$

$$A_i \cap A_j = \emptyset \text{ for } i \neq j$$

$$A_1 \cup A_2 \cup \cdots \cup A_k = S$$

Eksempel:

$\{a, b, e\}$, $\{f\}$, $\{c, d, g, h\}$ er en partition af $\{a, b, c, e, f, g, h\}$

Disjoint Sets operationer

Disjunkte mængder (partition) som datastruktur? Følgende er en samling operationer, som har vist sig relevante i anvendelser (anvendelser senere i kurset):

MAKE-SET(x):

Opret $\{x\}$ som en mængde.

UNION(x, y):

Slå $\{a, b, c, \ldots, x\}$ og $\{h, i, j, \ldots, y\}$ sammen til $\{a, b, c, \ldots, x, h, i, j, \ldots, y\}$.

FIND-SET(x):

Returner en ID for mængden indeholdende x.

Disjoint Sets operationer

Disjunkte mængder (partition) som datastruktur? Følgende er en samling operationer, som har vist sig relevante i anvendelser (anvendelser senere i kurset):

MAKE-SET(x):

Opret $\{x\}$ som en mængde.

UNION(x, y):

Slå $\{a, b, c, \ldots, x\}$ og $\{h, i, j, \ldots, y\}$ sammen til $\{a, b, c, \ldots, x, h, i, j, \ldots, y\}$.

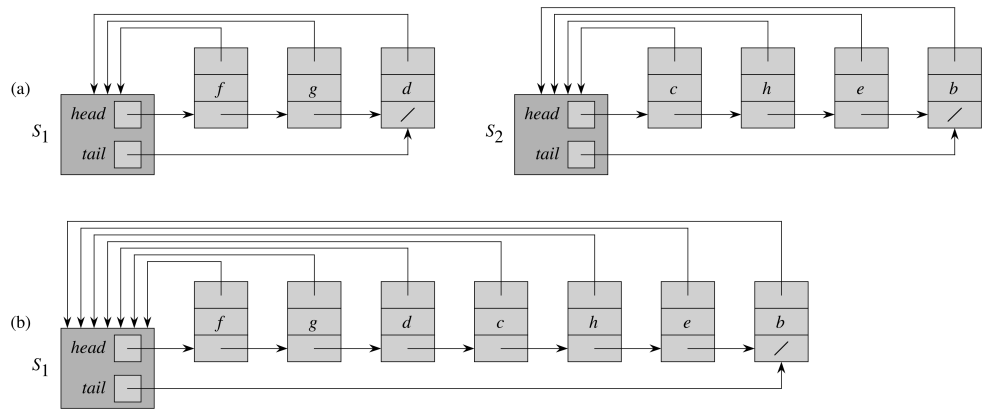
FIND-SET(x):

Returner en ID for mængden indeholdende x.

NB: Vi har ingen krav til ID'en. Skal blot være den samme for alle x i samme mængde, således at vi kan checke om to elementer x og y ligger i samme mængde.

Datastruktur for Disjoint Sets

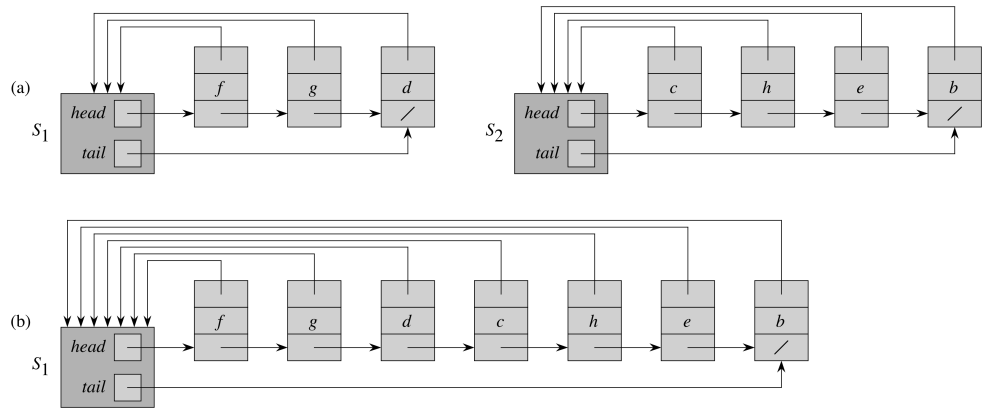
En simpel løsning:



- MAKE-SET(x): opret ny liste.
- UNION(x, y): slå lister sammen, behold én header, ændrer alle header-pointere i den anden liste.
- FIND-SET(x): returner pointer til header.

Datastruktur for Disjoint Sets

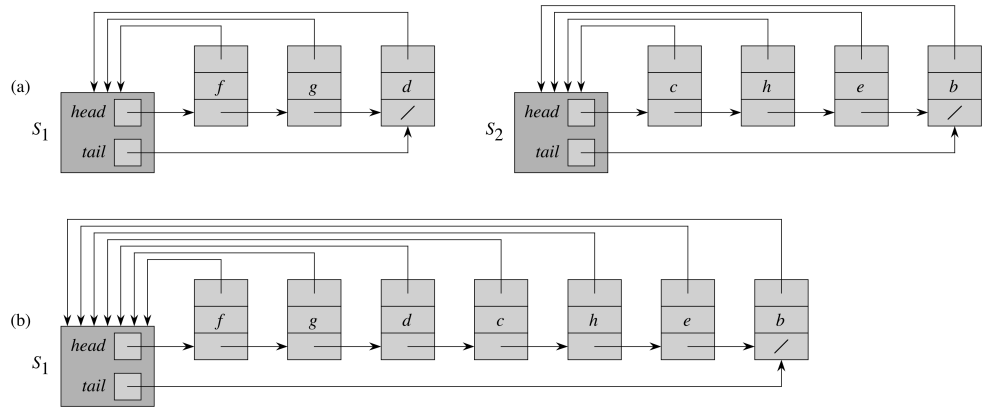
Køretid (n er antal elementer, dvs. antal MAKE-SETs udført):



- MAKE-SET(x): opret ny liste: $O(1)$.
- UNION(x, y): slå lister sammen, behold én header, ændre alle header-pointere i den anden liste: $O(n)$.
- FIND-SET(x): returner pointer til header: $O(1)$.

Datastruktur for Disjoint Sets

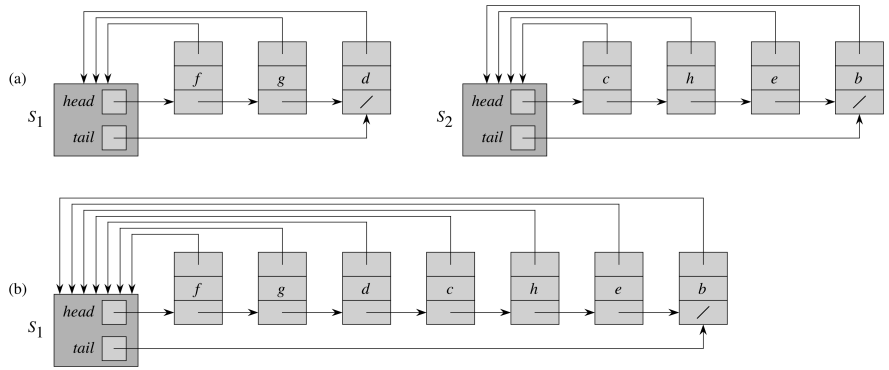
Køretid (n er antal elementer, dvs. antal MAKE-SETs udført):



- MAKE-SET(x): opret ny liste: $O(1)$.
- UNION(x, y): slå lister sammen, behold én header, ændre alle header-pointere i den anden liste: $O(n)$.
- FIND-SET(x): returner pointer til header: $O(1)$.

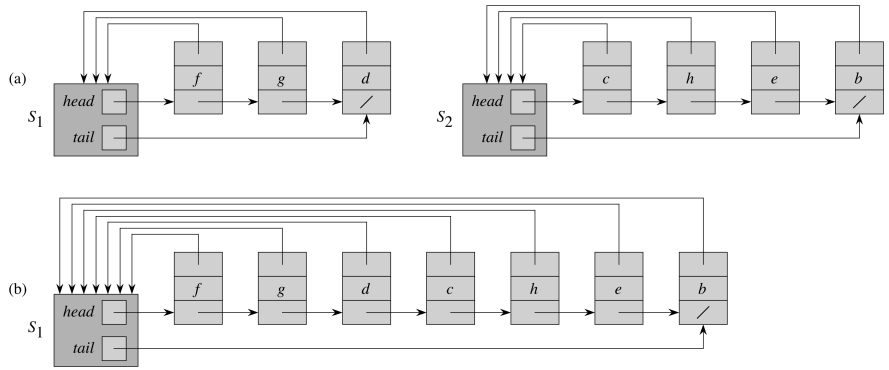
Naiv analyse: n MAKE-SET, op til $n-1$ UNION, og m FIND-SET koster $O(m+n^2)$.

Datastruktur for Disjoint Sets



- MAKE-SET(x): opret ny liste: $O(1)$.
- UNION(x, y): slå lister sammen, behold header af længste liste, ændre alle header-pointere i korteste liste: $O(n)$.
- FIND-SET(x): returner pointer til header: $O(1)$.

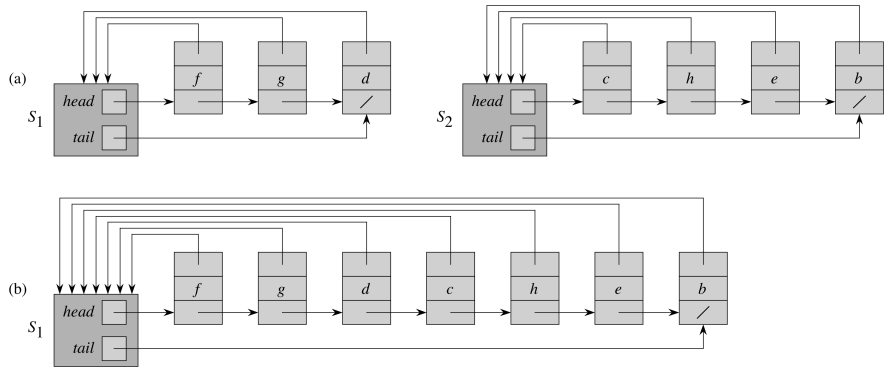
Datastruktur for Disjoint Sets



- MAKE-SET(x): opret ny liste: $O(1)$.
- UNION(x, y): slå lister sammen, behold header af længste liste, ændre alle header-pointere i korteste liste: $O(n)$.
- FIND-SET(x): returner pointer til header: $O(1)$.

Observer at nu gælder: en knude kan kun ændre sin header-pointer log n gange, da størrelsen af dens mængde hver gang vokser mindst en faktor to ($1 \cdot 2^k \leq n \Leftrightarrow k \leq \log n$).

Datastruktur for Disjoint Sets

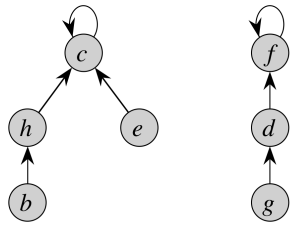


- MAKE-SET(x): opret ny liste: $O(1)$.
- UNION(x, y): slå lister sammen, behold header af længste liste, ændre alle header-pointere i korteste liste: $O(n)$.
- FIND-SET(x): returner pointer til header: $O(1)$.

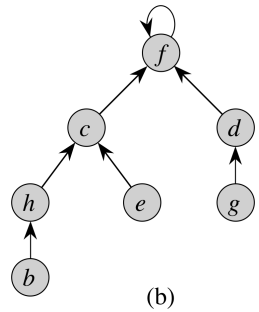
Observer at nu gælder: en knude kan kun ændre sin header-pointer log n gange, da størrelsen af dens mængde hver gang vokser mindst en faktor to ($1 \cdot 2^k \leq n \Leftrightarrow k \leq \log n$).

Så bedre analyse: n MAKE-SET, op til $n-1$ UNION, og m FIND-SET koster $O(m + n \log n)$.

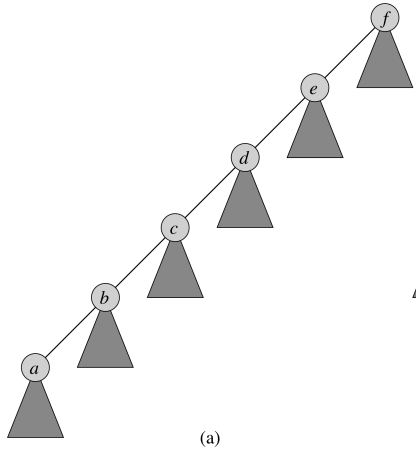
En anden datastruktur



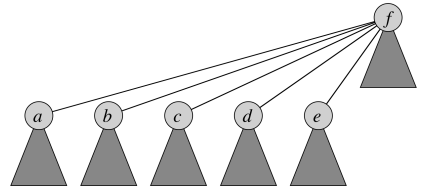
(a)



(b)

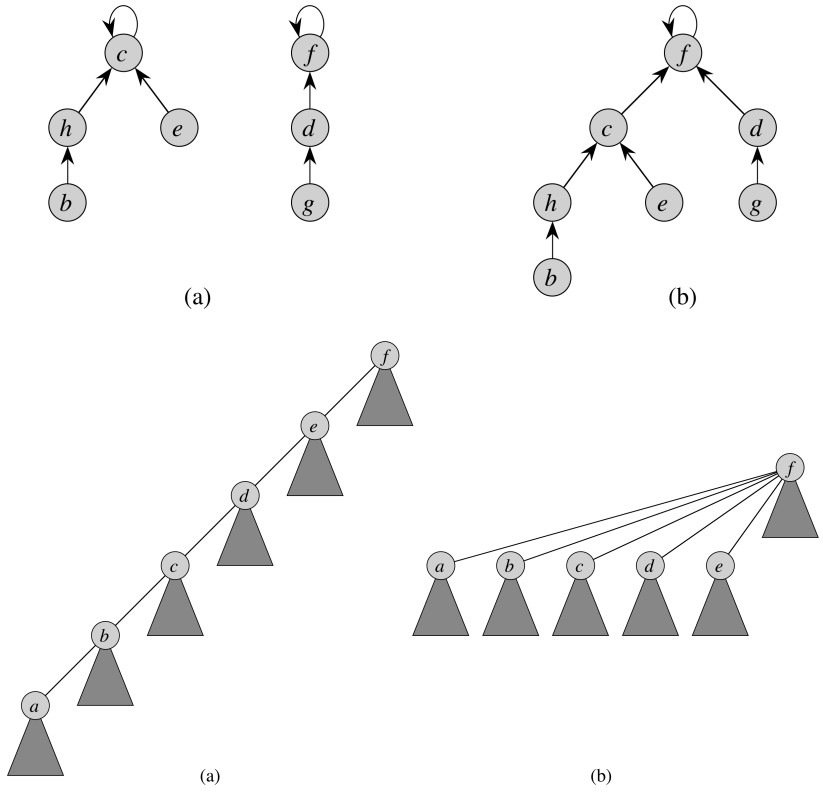


(a)



(b)

En anden datastruktur



Union by rank + *path compression* (se bog afsnit 21.3 for definition) $\Rightarrow$ meget tæt på $O(m+n)$ tid. Mere præcist $O(m \cdot \alpha(n))$, hvor $\alpha(n)$ er en meget langtsomt voksende funktion (defineret i afsnit 21.4, som ikke er pensum).

Analyse: i et senere kursus.