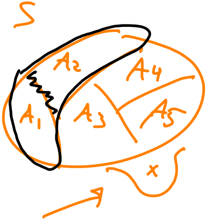
Disjoint Sets

Partition

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:



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

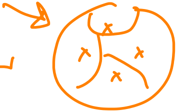
$$A_i \cap A_j = \emptyset \text{ for } i \neq j$$

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

Eksempel:

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

A_1 A_2 A_3



Disjoint Sets operationer

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

$\textsc{Make-Set}(x)$:

Opret $\{x\}$ som en mængde.

$\textsc{Union}(x, y)$:

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

$\textsc{Find-Set}(x)$:

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):

$\textsc{Make-Set}(x)$:

Opret $\{x\}$ som en mængde.

$\textsc{Union}(x, y)$:

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

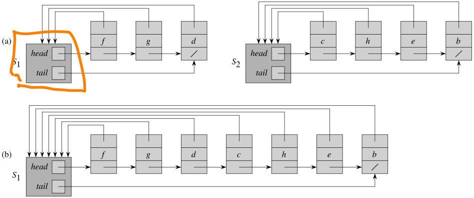
$\textsc{Find-Set}(x)$:

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 via lænkede lister

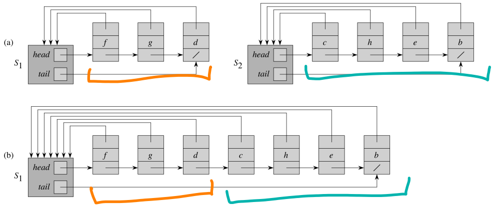
Hver mængde er en lænket liste af elementer, liste-header er ID for mængde:



Datastruktur for Disjoint Sets via lænkede lister

Hver mængde er en lænket liste af elementer, liste-header er ID for mængde:

Union



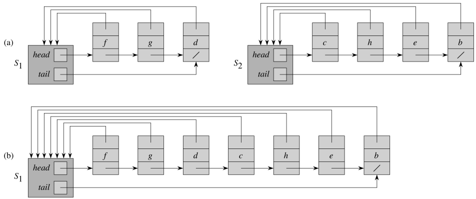
- $\textsc{Find-Set}(x)$: returner pointer til header.
- $\textsc{Make-Set}(x)$: opret ny liste.
- $\textsc{Union}(x, y)$: slå lister sammen, behold én header, ændrer alle header-pointere i den anden liste.

Datastruktur for Disjoint Sets via lænkede lister

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

Datastruktur for Disjoint Sets via lænkede lister

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



- FIND-SET(x): returner pointer til header: $O(1)$.
- 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: $O(n)$.

Datastruktur for Disjoint Sets via lænkede lister

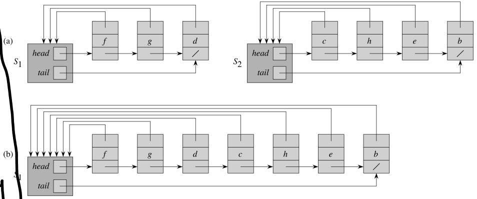
Køretid (n er antal elementer, dvs. antal Make-Sets udført)?

1

2

3

4

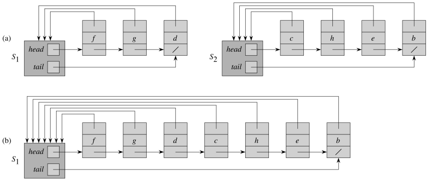


$\rightarrow 1 + 2 + \dots (n-1) = \Theta(n^2)$

- Find-Set(x): returner pointer til header: $O(1)$.
- 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: $O(n)$.

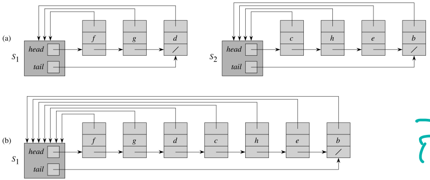
Naiv analyse: n Make-Set, op til $n-1$ Union, og m Find-Set koster $O(m + n^2)$.

Datastruktur for Disjoint Sets via lænkede lister



- $\textsc{Find-Set}(x)$: returner pointer til header: $O(1)$.
- $\textsc{Make-Set}(x)$: opret ny liste: $O(1)$.
- $\textsc{Union}(x, y)$: slå lister sammen, behold header af længste liste, ændre alle header-pointere i korteste liste: $O(n)$.

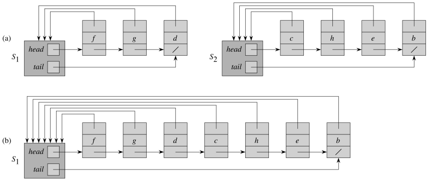
Datastruktur for Disjoint Sets via lænkede lister



- FIND-SET(x): returner pointer til header: $O(1)$.
- 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: $O(n)$.

Observér 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 via lænkede lister



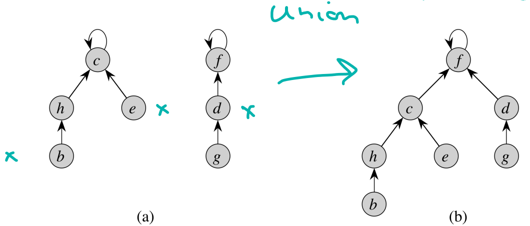
- $\textsc{Find-Set}(x)$: returner pointer til header: $O(1)$.
- $\textsc{Make-Set}(x)$: opret ny liste: $O(1)$.
- $\textsc{Union}(x, y)$: slå lister sammen, behold header af længste liste, ændre alle header-pointere i korteste liste: $O(n)$.

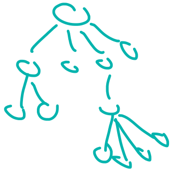
Observér at nu gælder: en knude kan kun ændre sin header-pointer $\log n$ gange, da størrelsen af dens mængde hver gang vokser mindst en faktor to ($1 \cdot 2^k \leq n \Leftrightarrow k \leq \log n$).

Så bedre analyse: n $\textsc{Make-Set}$, op til $n - 1$ $\textsc{Union}$, og m $\textsc{Find-Set}$ koster $O(m + n \log n)$.

Datastruktur for Disjoint Sets via træer

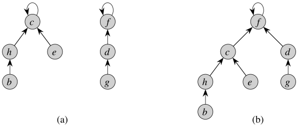
Hver mængde er et træ med elementer i knuder, rod er ID for mængde:



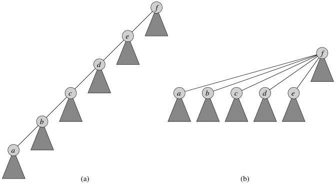


- $\textsc{Find-Set}(x)$: gå til rod.
- $\textsc{Make-Set}(x)$: opret nyt træ.
- $\textsc{Union}(x, y)$: gør rod af ét træ til barn af andet træ.

Datastruktur for Disjoint Sets via træer

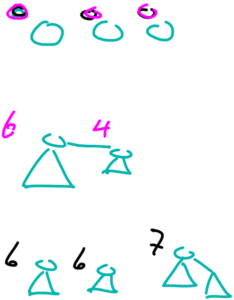


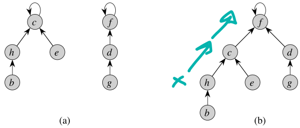
(a) (b)

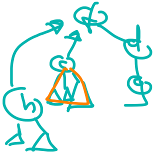


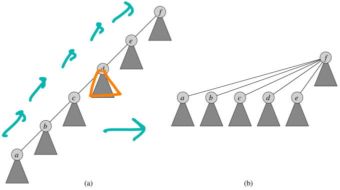
(a) (b)

Datastruktur for Disjoint Sets via træer

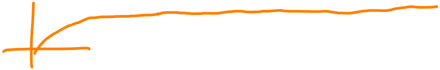








Union by rank + *path compression* (se bog afsnit 21.3 for definition) $\Rightarrow$ meget tæt på $O(m+n)$ tid. Mere præcist $O(m\ \alpha(n))$, hvor $\alpha(n)$ er en meget langtsomt voksende funktion (defineret i afsnit 21.4, som ikke er pensum).



Analyse: i et senere kursus.