

Disjoint Sets

Partition

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

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

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

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

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

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

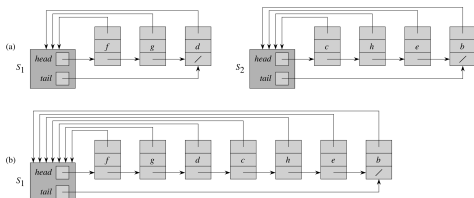
FIND-SET(x):

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 via l enkede lister

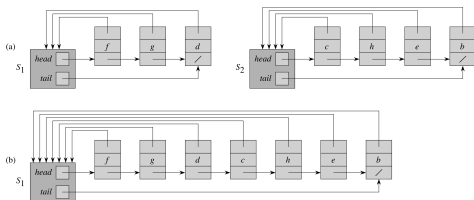
En simpel l osning:



- ▶ MAKE-SET(x): opret ny liste.
- ▶ UNION(x, y): sl  lister sammen, behold  n header,  ndrer alle header-pointere i den anden liste.
- ▶ FIND-SET(x): returner pointer til header.

Datastruktur for Disjoint Sets via lænkede lister

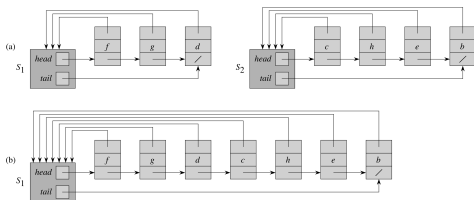
Køretid (n er antal elementer, dvs. antal MAKE-SETS udført):



- ▶ 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 via lænkede lister

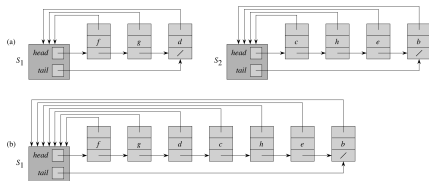
Køretid (n er antal elementer, dvs. antal MAKE-SETS udført):



- ▶ 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: $O(1)$.

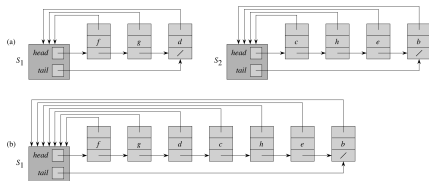
Naiv analyse: n MAKE-SET, op til $n - 1$ UNION, og m FIND-SET koster $O(m + n^2)$.

Datastruktur for Disjoint Sets via lænkede lister



- ▶ 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: $O(1)$.

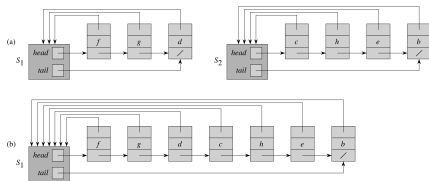
Datastruktur for Disjoint Sets via l enkede lister



- ▶ MAKE-SET(x): opret ny liste: $O(1)$.
- ▶ UNION(x, y): sl  lister sammen, behold header af **l ngste liste**,  ndre alle header-pointere i **korteste liste**: $O(n)$.
- ▶ FIND-SET(x): returner pointer til header: $O(1)$.

Observer at nu g elder: en knude kan kun  ndre sin header-pointer log n gange, da st rrelsen af dens m ngde hver gang vokser mindst en faktor to ($1 \cdot 2^k \leq n \Leftrightarrow k \leq \log n$).

Datastruktur for Disjoint Sets via l enkede lister

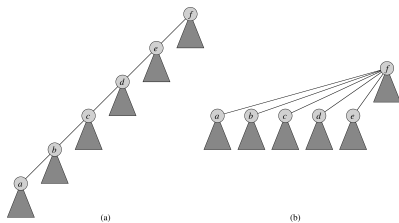
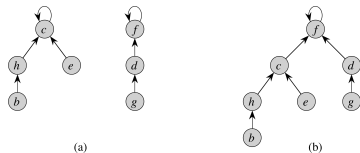


- ▶ MAKE-SET(x): opret ny liste: $O(1)$.
- ▶ UNION(x, y): sl  lister sammen, behold header af **l ngste liste**,  ndre alle header-pointere i **korteste liste**: $O(n)$.
- ▶ FIND-SET(x): returner pointer til header: $O(1)$.

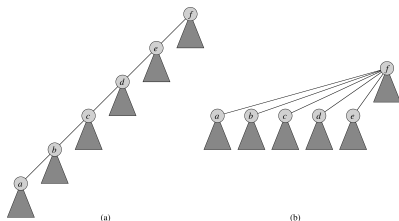
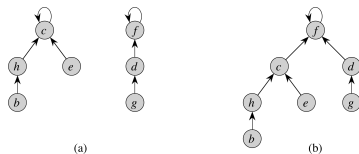
Observer at nu g elder: en knude kan kun  ndre sin header-pointer log n gange, da st rrelsen af dens m ngde hver gang vokser mindst en faktor to ($1 \cdot 2^k \leq n \Leftrightarrow k \leq \log n$).

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

Datastruktur for Disjoint Sets via træer



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 $O(m \cdot \alpha(n))$, hvor $\alpha(n)$ er en meget langsomt voksende funktion (defineret i afsnit 21.4, som ikke er pensum).

Analyse: i et senere kursus.