Disjoint Sets

Partition

En partition (dansk: 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 $\{a, b, c, d, e, f, g, h\}$

Datastrukturen Disjoint Sets

Disjunkte opdeling som datastruktur?

Datastrukturen Disjoint Sets

Disjunkte opdeling som datastruktur? Følgende samling operationer har vist sig relevante i mange anvendelser (herunder nogle senere i kurset):

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

Opret $\{x\}$ som en ny mængde (x må ikke være element i andre mængder).

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

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.

Datastrukturen Disjoint Sets

Disjunkte opdeling som datastruktur? Følgende samling operationer har vist sig relevante i mange anvendelser (herunder nogle senere i kurset):

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

Opret $\{x\}$ som en ny mængde (x må ikke være element i andre mængder).

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

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

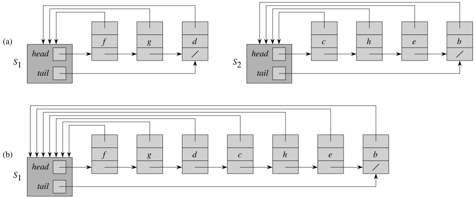
$\textsc{Find-Set}(x)$:

Returner (en ID for) mængden indeholdende x.

NB: Vi har ingen krav til ID for mængder. Den skal blot være ens for alle x i samme mængde, således at vi kan checke om to elementer x og y ligger i samme mængde.

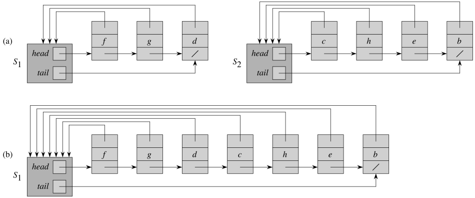
Disjoint Sets implementeret via lænkede lister

Hver mængde er en lænket liste af elementer, ID for mængde er første element i listen:



Disjoint Sets implementeret via lænkede lister

Hver mængde er en lænket liste af elementer, ID for mængde er første element i listen:



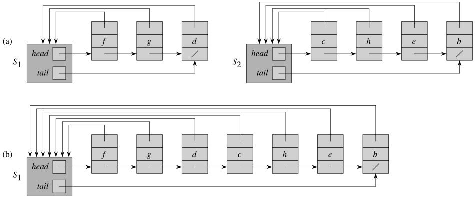
- $\textsc{Find-Set}(x)$: returner (via header-pointer) første element i listen.
- $\textsc{Make-Set}(x)$: opret ny liste.
- $\textsc{Union}(x, y)$: slå lister sammen, behold én header, ændrer alle header-pointere i den anden liste.

Disjoint Sets implementeret via lænkede lister

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

Disjoint Sets implementeret via lænkede lister

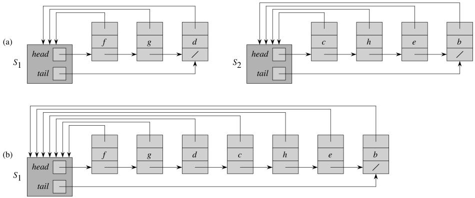
Køretid (n er antal elementer, dvs. antal MAKE-SETs udført)?



- FIND-SET(x): returner (via header-pointer) første element i listen: $O(1)$.
- 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: $O(n)$.

Disjoint Sets implementeret via lænkede lister

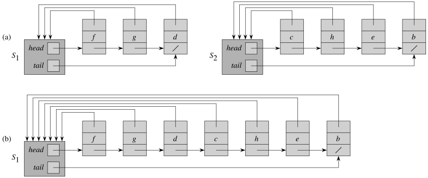
Køretid (n er antal elementer, dvs. antal MAKE-SETs udført)?



- FIND-SET(x): returner (via header-pointer) første element i listen: $O(1)$.
- 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: $O(n)$.

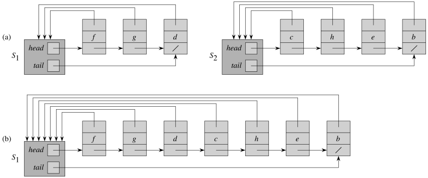
Naiv analyse: n MAKE-SET, op til $n - 1$ UNION, og m FIND-SET koster $O(m + n^2)$.

Disjoint Sets implementeret via lænkede lister, version 2



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

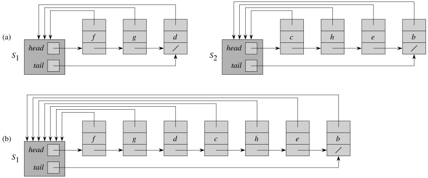
Disjoint Sets implementeret via lænkede lister, version 2



- $\textsc{Find-Set}(x)$: returner (via header-pointer) første element i listen: $O(1)$.
- $\textsc{Make-Set}(x)$: 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 $(1 \cdot 2^k \leq n \Leftrightarrow k \leq \log n)$.

Disjoint Sets implementeret via lænkede lister, version 2



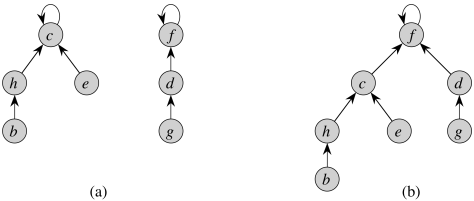
- FIND-SET(x): returner (via header-pointer) første element i listen: $O(1)$.
- MAKE-SET(x): opret ny liste: $O(1)$.
- UNION(x, y): slå lister sammen, behold header af længste liste, ændre alle header-pointere i korteste liste: $O(n)$.

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 ($1 \cdot 2^k \leq n \Leftrightarrow k \leq \log n$).

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

Disjoint Sets implementeret via træer

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



(a) (b)

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

Disjoint Sets implementeret via træer, version 2

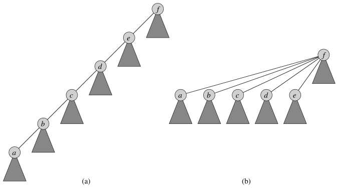
Disjoint Sets implementeret via træer, version 2

Union by rank: Tilføj et heltal (rank) til alle rødder. Sættes til 0 ved nye træer (under $\textsc{Make-Set}(x)$). Kontrollerer forældre-barn beslutningen ved $\textsc{Union}(x, y)$: rod med størst rank får den anden rod som barn. Ved ens rank, lad y få x som barn og øg y's rank med 1.

Disjoint Sets implementeret via træer, version 2

Union by rank: Tilføj et heltal (rank) til alle rødder. Sættes til 0 ved nye træer (under MAKE-SET(x)). Kontrollerer forældre-barn beslutningen ved UNION(x, y): rod med størst rank får den anden rod som barn. Ved ens rank, lad y få x som barn og øg y's rank med 1.

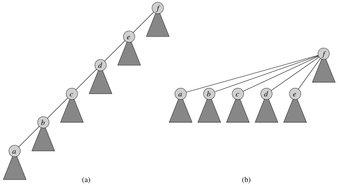
Path compression: Under FIND-SET(x), sæt alle passerede knuder op som børn af roden. For FIND-SET(a):



Disjoint Sets implementeret via træer, version 2

Union by rank: Tilføj et heltal (rank) til alle rødder. Sættes til 0 ved nye træer (under $\textsc{Make-Set}(x)$). Kontrollerer forældre-barn beslutningen ved $\textsc{Union}(x, y)$: rod med størst rank får den anden rod som barn. Ved ens rank, lad y få x som barn og øg y's rank med 1.

Path compression: Under $\textsc{Find-Set}(x)$, sæt alle passerede knuder op som børn af roden. For $\textsc{Find-Set}(a)$:



Union by rank og path compression $\Rightarrow$ meget tæt på $O(m + n)$ tid. Mere præcist $O(m \cdot \alpha(n) + n)$, hvor $\alpha(n)$ er en meget langsomt voksende funktion.

Funktionen $\alpha(n)$ samt beviset for køretiden findes i afsnit 19.4, som ikke er pensum (gennemgås i et senere kursus på datalogistudiet).

Disjoint Sets implementeret via træer, version 2

Pseudokode (med union by rank og path compression) er forbavsende simpel:

```
MAKE-SET(x)
  x.p = x
  x.rank = 0

UNION(x, y)
  LINK(FIND-SET(x), FIND-SET(y))

FIND-SET(x)
  if x ≠ x.p
      x.p = FIND-SET(x.p)
  return x.p

LINK(x, y)
  if x.rank > y.rank
      y.p = x
  else x.p = y
      // If equal ranks, choose y as parent and increment its rank.
      if x.rank == y.rank
          y.rank = y.rank + 1
```