Ugeseddel: Forén og find

Philip Bille

Om denne uge

Litteratur *Introduction to Algorithms*, Cormen, Rivest, Leisersons og Stein (CLRS): kap. 21.1-21.4 og *Algorithms*, 4ed., Sedgewick og Wayne: kap 1.5.

Opgaver

1 Håndkøring af forén og find Kig på følgende sekvens af operationer: INIT(7),UNION(3,4), UNION(5,0), UNION(4,5), UNION(4,3), UNION(0,1), UNION(2,6), UNION(0,4) og UNION(6,0).

1.1 [o] Håndkør sekvensen med hurtig find. Vis indholdet af *id* tabellen efter hvert skridt. Antag at UNION(i,j) operationen altid opdaterer *id* for mængden angivet ved i.

1.2 [o] Håndkør sekvensen med hurtig forening. Vis træerne efter hvert skridt. Antag at UNION(i,j) altid sætter roden af træet angivet ved i til at være barn af roden af træet angivet ved j.

1.3 Håndkør sekvensen med vægtet forening. Vis træerne efter hvert skridt. Antag at UNION(i,j) sætter roden af træet angivet ved i til at være barn af roden af træet angivet ved j når træerne har samme størrelse.

1.4 Vis resultat af stikompression efter en FIND(x) operation, hvor x er hhv. et blad, en intern knude af dybde 1 og en intern knude af højde 1, i et træerne fra øvelserne ovenfor.

1.5 Giv en sekvens af operationer, der fører til et træ af maksimal dybde i hurtig forening.

1.6 Giv en sekvens af operationer, der fører til et træ af maksimal dybde i vægtet forening.

1.7 Skriv pseudokode for en algoritme til stikompression. *Hint:* løb stien igennem to gange.

2 Alternativ hurtig find algoritme En af dine medstuderende foreslår følgende intuitive variant af UNION hurtig find. Virker den?

```
UNION(i, j)
if FIND(i) ≠ FIND(j) then
  for k = 0 to n − 1 do
    if id[k] == id[i] then
      id[k] = id[j]
    end if
  end for
end if
```

3 Dynamiske sammenhængskomponenter og grafsøgning Med grafsøgning (DFS eller BFS) kan vi finde sammenhængskomponenterne i en graf. Giv en simpel løsning til dynamiske sammenhængskomponenter vha. grafsøgning og sammenlign kompleksiteten med løsningerne til forén og find.

4 Implementering af forén og find Vi vil gerne implementere datastrukturer for forén og find der understøtter INIT, UNION og FIND.

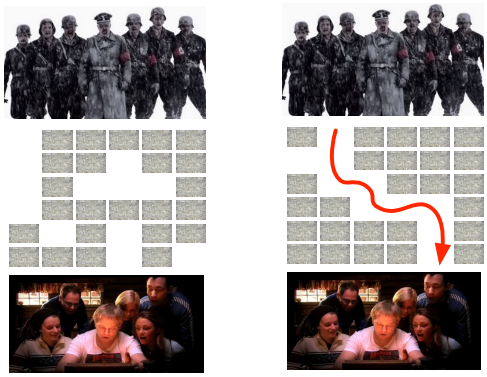
4.1 [$D\dagger$] Implementer hurtig find.

4.2 [$\dagger$] Implementer hurtig forening.

4.3 [$\dagger$] Udvid løsning med vægtet forening.

4.4 [$\dagger$] Udvid løsning med stikompression.

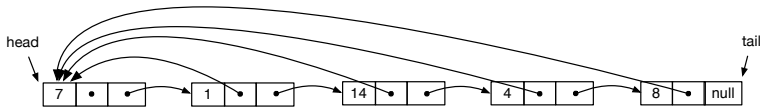
5 [$*$] **Zombieinvasion** I den post-apokalyptiske zombieverden har du og en lille gruppe overlevende forskanset sig i en mindre beboelse. Det eneste der afholder den glubske hær af zombier i at komme ind og spise dig og dine venner er solidt fæstningværk. Fæstningsværket består af et $k \times k$ gitter af mure. Illustreret her med et 6×6 gitter af mure.[1]



I toppen af gitteret venter zombierne på at komme ind og i bunden er beboelsen. Murerne er desværre skrøbelige og falder derfor sammen med jævnlige mellemrum. Hvis en sti af mure mellem toppen og bunden af gitteret et faldet sammen kan zombierne komme ind. For at kunne nå at evakure vil du derfor gerne kunne holde styr på om der er en sti igennem fæstningsværket eller ej. Giv en datastruktur, der effektivt kan holde styr om der er en sti, efterhånden som mure falder sammen.

6 [$*$] **Rekursiv stikompression** Skriv pseudokode for en *rekursiv* algoritme til stikompression. *Hint:* det kan gøres med meget lidt kode.

7 Forén og find med hægtede lister og vægte Vi ønsker at implementere en variant af hurtig find med hægtede lister på følgende måde. Hver mængde er repræsenteret ved en enkelt hægtet liste. Repræsentanten for en mængde er det første element i listen og hver element i listen har en peger til repræsentanten. Derudover vedligeholder vi en peger til halen af listen. F. eks. kan datastrukturen for mængden $\{1, 4, 7, 8, 14\}$ med repræsentant 7 se sådan ud:



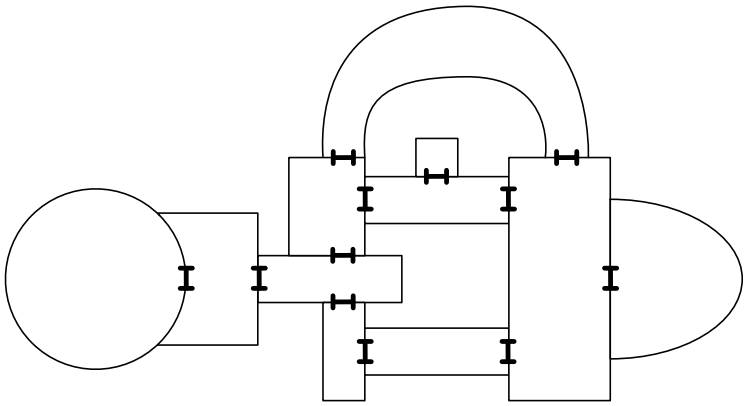
7.1 Vis hvordan man i denne repræsentation kan implementere INIT(n) i $O(n)$ tid, FIND(i) i $O(1)$ tid og UNION(i, j) i $O(|S(i)|)$ tid, hvor $S(i)$ er mængden der indeholder i.

7.2 Vis hvordan man kan udvide løsningen således at INIT og FIND kører i samme tid, men tiden for UNION(i, j) bliver $O(\min(|S(i)|, |S(j)|))$. *Hint:* vedligehold lidt ekstra information.

7.3 [$*$] Vis at ovenstående løsning giver at køretiden for p FIND og m UNION operationer på n elementer er $O(p + m \log n)$.

[1] Billeder fra "Død snø", 2009.

O Obligatorisk afleveringsopgave: Plantegning (opgaver fra eksamen 2015) Et *plantegning* består af en mængde af R rum $r_0, .., r_{R-1}$ og D døre $d_0, .., d_{D-1}$, der hver forbinder præcis to rum. Hvert rum er en geometrisk figur og en dør mellem to rum er angivet ved tre små fede streger mellem rummene. F. eks. består nedenstående plantegning P af 11 rum og 12 døre.



O.1 Beskriv hvordan man kan modellere en plantegning som en graf.

O.2 Tegn grafen svarende til plantegningen P i eksemplet ovenfor.

O.3 Vi er nu interesserede i at undersøge om det er muligt at evakuere rummene i tilfælde af en brand. *Entréen* er et særligt rum på plantegningen. En *branddør* er en dør, der automatisk lukker i tilfælde af brand. Et rum kan *evakueres* til entréen hvis der er en forbindelse fra rummet til entreen, der ikke benytter branddøre. Giv en algoritme, der givet en plantegning, en entré e og en mængde B af k branddøre, afgør om alle rum kan evakueres til e. Analyser køretiden af din algoritme som funktion af R, D, og k.

O.4 Vi er nu interessede i at udstille kunst i form af enten en skulptur eller et maleri i alle rum på en behagelig facon. En *rute* i plantegning er en sekvens af rum $r_0, \ldots, r_{z-1}$ således at rum r_i og r_{i-1} er forbundet af en dør, og $r_0 = r_{z-1}$. Et rum må gerne besøges mange gange på ruten. En rute er *smuk* hvis der i hvert rum på ruten skiftevis er udstillet en skulptur eller et maleri. Kig på eksemplet med plantegningen P fra før. Er muligt at udstille en skulptur eller et maleri i hvert rum således at *alle* ruter der starter og slutter i rummet længst til venstre er smukke?

O.5 Giv en algoritme, der givet en plantegning og en entré e, afgør om alle ruter der starter og slutter i e er smukke. Vi antager at der er en rute fra e til alle rum. Analyser køretiden af din algoritme som funktion af R og D.