Algoritmer og Datastrukturer Eksamen

Eksamensnummer: 137

April 1, 2020

Opgave 1

Del 1.1

En Fibonacci heap, H, er en mængde af rodfæstede træer der overholder *min-heap property*'en - at nøglen i en knude er større eller lig med forældrens nøgle, hvilket resulterer i, at én af knuderne har den mindste nøgle med den mindste værdi i hele heap'en - den såkaldte *minimum node*. Vi tilgår en Fibonnaci heap med pointer'en *H.min*, der pejer til denne *minimum node*. Træernes rødder har pejerene *left* og *right* der henholdsvis pejer til rødderne for træerne til højre og venstre for sig selv - dette skaber den såkaldte *root list*.
Alle knuder har en pejer, *p*, til sin forældre og en pejer, *child*, til ét af sine børn; børnene er sammensat i en *circular, doubly linked list*, hvilket herved gør det muligt for forældren at tilgå enhver af sine børn. Alle børn har pejerne *left* og *right*, der ved hjælp af den *circular, doubly linked list* pejer til barnets højre og venstre søskene. Herudover har alle knuder attributen *degree*, der beskriver antallet af børn som knuden har.
Herudover har alle knuder attributen *mark*, der indikerer om en knude har mistet et barn siden knuden selv var lavet til et barn af en anden knude - dog bliver rodknuder ikke markeret, hvis når den mister et barn. En knude bliver *unmarked* når den bliver lavet til et barn af en anden knude (hvilket kan ske når den mister sit andet barn), samt når den bliver skabt.
Fibonnaci heaps er en meget effektiv datastruktur, idet størstedelen af operationerne i amortiseret køretid kører i konstant tid - de eneste operationer der ikke kører i konstant tid er `EXTRACT-MIN` og `DELETE`, der har køretiderne $O(\lg n)$ og $O(\lg n)$ amortiseret køretid. Herved egner datastruktureren sig godt til algoritmer der anvender datastrukturerens operationer mange gange - og særlig godt, hvis `EXTRACT-MIN` og `DELETE` ikke anvendes særlig mange gange. Dette ses eksempelvis fra pensum i Prims algoritme og i Dijkstras algoritme.
Ved brugen af binære heaps har Prims algoritme en køretid på $O(E \lg V)$. Anvender algoritmen dog Fibonnaci heaps istedet, får algoritmen en køretid på $O(E + V \lg V)$, hvilket er bedre end implementationen med binære heaps, hvis tilfældet er, at antallet af knuder er meget mindre end antallet af kanter.
Dijkstras køretid afhænger af hvordan *min-priorty* køen bliver implementeret - implementeres den ved hjælp af Fibonnaci heaps, bliver køretiden $O(V \lg V + E)$ - meget bedre end andre implementationers køretid, som for eksempel $O(V^2)$. Dette skyldes, at algoritmen generelt kalder `DECREASE-KEY` mange flere gange, end algoritmen kalder `EXTRACT-MIN`.

Del 1.2

Jeg har følgende algoritme for `INCREMENT`:

```
INCREMENT(A):
    i = 0
    while (i < A.length and A[i] == 9):
        A[i] = 0
        i += 1
    if (i < A.length):
        A[i] += 1
```

Algoritmen fungerer ved at iterere igennem array A indtil der findes en indgang der ikke er lig 9 - alle indgange op til dette sættes til 0, idet at lægge 1 til 9 resulterer i 0. Herefter lægges der 1 til den første værdi der ikke er lig 9, idet der herved er tale om et *carry over* fra det forrige ciffer.