Algoritmer, datastrukturer och komplexitet, hösten 2015

Uppgifter till övning 2

Datastrukturer och grafer

På denna övning är det också **inlämning av skriftliga lösningar av teoriuppgifterna till labb 1** och muntlig redovisning av teoriuppgifterna. Teoriuppgifterna redovisas individuellt, till skillnad från labben som görs i par.

Samsortering Beskriv en algoritm som samsorterar k stycken var för sig sorterade listor i tid $O(n \log k)$ där n är det totala antalet element.

Datastruktur med max och min Konstruera en datastruktur som har följande operationer och komplexitet:

- `insert(x)` (insättning av ett element) tar tid $O(\log n)$,
- `deletemin()` (borttagning av det minsta elementet) tar tid $O(\log n)$,
- `deletemax()` (borttagning av det största elementet) tar tid $O(\log n)$,
- `findmin()` (returnerar minsta elementet) tar tid $O(1)$,
- `findmax()` (returnerar största elementet) tar tid $O(1)$.

n är antalet element som för närvarande finns lagrade. Du får anta att man kan jämföra två element och ta reda på vilket som är störst i tid $O(1)$.

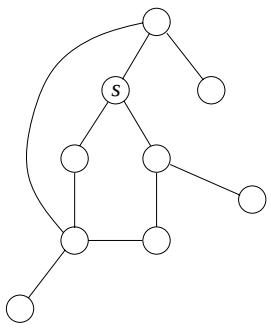
Bloomfilter med borttagning Det finns bara två tillåtna operationer på ett vanligt Bloomfilter: Insert och IsIn. Hur kan man modifiera datastrukturen så att den också tillåter operationen Remove, som inte får ta längre tid än Insert?

Suffixregeluppslagning i Stava Stava har cirka 1000 suffixregler av typen

-orna ⟵ *-a, -an, -or*

När ett ord som *dockorna* ska kontrolleras kommer Stava att i tur och ordning gå igenom alla suffix efter den första stavelsen, det vill säga *-ϵ, -a, -na, -rna, -orna, -korna, -ckorna*. Hur ska suffixreglerna lagras för att det ska gå snabbt att slå upp suffixen?

Grafsökning Gör en DFS- och en BFS-genomgång av följande graf. Starta i hörnet s och numrera hörnen i den ordning dom besöks.



Topologisk sortering av DAG En DAG är en riktad acyklisk graf. En topologisk sortering av en sån graf är en numrering av hörnen så att alla kanter går från ett hörn med lägre nummer till ett hörn med högre nummer.

Modifiera den vanliga djupetförstsökningsalgoritmen så att den konstruerar en topologisk sortering av grafen i linjär tid.

Klick En *klick* (eng. clique) i en oriktad graf är en mängd hörn där varje par av hörn har en kant mellan sig. Ett viktigt problem i datalogin är problemet att hitta den största klicken i en graf.

Ett annat mycket omtalat problem är problemet att hitta den största *oberoende mängden* i en graf. En oberoende mängd (eng. independent set) är en mängd hörn som inte har någon kant alls mellan sig.

Dessa båda problem är så lika varandra att om man har en algoritm som givet en graf hittar den största klicken så kan man utnyttja den algoritmen i en för övrigt mycket enkel algoritm som givet en graf hittar den största oberoende mängden. Visa detta!

Kantmatrisprodukt På föreläsningen beskrevs hur en riktad graf $G = \langle V, E \rangle$ representeras som en kantmatris B (eng. incidence matrix) av storlek $|V| \times |E|$. Beskriv vad elementen i matrisen BB^T representerar (där B^T är B transponerad).

Bipartithet Beskriv och analysera en algoritm som avgör om en graf är bipartit. Tidskomplexiteten för algoritmen ska vara linjär i antalet kanter och antalet hörn i grafen.

Lösningar och ledningar

Ledning till Samsortering
Metod 1: Använd en heap där ett element från varje lista finns med och där det minsta elementet ligger i roten. Varje heapoperation tar $O(\log k)$ och det görs n insättningar i och n borttagningar ur heapen.
Metod 2: Samsortera listorna parvis. Nu finns det bara $k/2$ sorterade listor. Samsortera dessa listor parvis, och så vidare. Efter $\log k$ samsorteringsomgångar finns det bara en lista kvar. Varje samsorteringsomgång tar tid $O(n)$ eftersom det är totalt n element som ska samsorteras i varje omgång.

Lösning till Datastruktur med max och min
Antingen kan man lösa uppgiften med hjälp av två trappor (heapar) med samma element, men där den ena är ordnad med minsta elementet i roten och den andra med största elementet i roten eller också med hjälp av ett balanserat sökträd som utökas med två extra variabler, en för det minsta värdet i trädet och en för det största.

Implementationerna är i båda fallen enkla. I binärträdsfallet är det trivialt att skriva `findmin` och `findmax`. För dom tre övriga operationerna använder man motsvarande vanliga balanserade sökträdsoperationer men efterbehandlar genom att kopiera dom minsta och största värdena i trädet till dom två variablerna. □

Lösning till Bloomfilter med borttagning
Låt Bloomfiltret bestå av tal (till exempel 8- eller 16-bits heltal) istället för bitar. Talet noll motsvarar bitvärdet 0 och varje positivt tal motsvarar bitvärdet 1. Istället för att sätta en bit till 1 så ökar Insert istället motsvarande tal med ett. Remove fungerar precis som Insert men minskar med ett istället för att öka med ett. Nu räknar heltalen hur många element som är hashade till varje position. När ett heltal kommit ner till noll så betyder det att inget element längre hashas till den positionen.

Storleken på heltalen måste avpassas till antalet element som beräknas ligga i Bloomfiltret. Man får nämligen problem om en räknare når heltalstypens tak. En möjlighet är att införa en extra bit per position som håller reda på om taket någon gång nåtts för den positionen. När en räknare med den biten satt kommer ner till noll så måste hela Bloomfiltret beräknas på nytt.

Ett alternativ är att den räknare som slår i taket får sitta fast i taket även om Remove anropas. Det betyder att den räknaren aldrig kommer att nå noll, även om alla element som hashas till den positionen tas bort. Det kommer förmodligen aldrig att inträffa eftersom det var så många element som hashats till den positionen, och om det inträffar så ökar sannolikheten för fel bara för dom ord som hashas dit, och det förmodligen inte så mycket. □

Lösning till Suffixregeluppslagning i Stava
Det man slår upp på är vänsterleden i reglerna. Vi kallar dessa för ingångssuffixen. Alla regler som har tomt ingångssuffix lagrar vi i en lista för sig. Övriga regler lagrar vi i en array sorterade efter ingångssuffixet läst baklänges. Vi använder latmannahashning på en bokstav, det vill säga vi har en bokstavsindexerad array som anger indexet till första ingångssuffixet som slutar på den bokstaven.

Vid sökning kollar man först i listan med regler med tomt ingångssuffix. Därefter slår man upp sista bokstaven (*a* i *dockorna*) med latmannahashning och går igenom dom regler som har den enda bokstaven som ingångssuffix. Sedan binärsöker man efter näst sista bokstaven (*n*) bland dom ingångssuffix som slutar med *a* och går igenom dom regler som har *na* som ingångssuffix. Därefter söker vi efter nästnäst sista bokstaven (*r*) bland dom ingångssuffix som slutar med *na* och går igenom dom regler som har *rna* som ingångssuffix och så vidare.

På detta sätt blir det färre och färre ingångssuffix att binärsöka bland. □

Lösning till Grafsökning – egen övning!

Lösning till Topologisk sortering av DAG
Om man tittar på i vilken ordning djupetförstsökningen passerar hörnen på tillbakavägen i grafen (det vill säga i vilken ordning hörnen färdigbehandlas av sökningsproceduren) så ser man att det är precis omvänd topologisk ordning. Om vi vill sortera hörnen topologiskt behöver vi alltså bara lägga till en sats `Push(u)` sist i proceduren `DFSVISIT(u)`. När djupetförstsökningen är genomförd kan vi poppa hörnen ett i taget från stacken, och då kommer dom topologiskt sorterade. □

Lösning till Klick
Ledning: studera komplementgrafen (som har kanter bara där ursprungsgrafen inte har kanter). □

Lösning till Kantmatrisprodukt
Diagonalelementet (i, i) anger hur många kanter som har sin ändpunkt i hörn i. Ickediagonalelementet (i, j) är det negerade antalet kanter som går mellan hörnen i och j. □

Lösning till Bipartithet
Använd djupetförstsökning men färga hörnen alternerande med rött och grönt. Om en kant mellan två hörn av samma färg upptäcks är grafen inte bipartit. □
