Algoritmer, datastrukturer och komplexitet

Övning 2

Anton Grensjö
grensjo@csc.kth.se

14 september 2015

Kursplanering

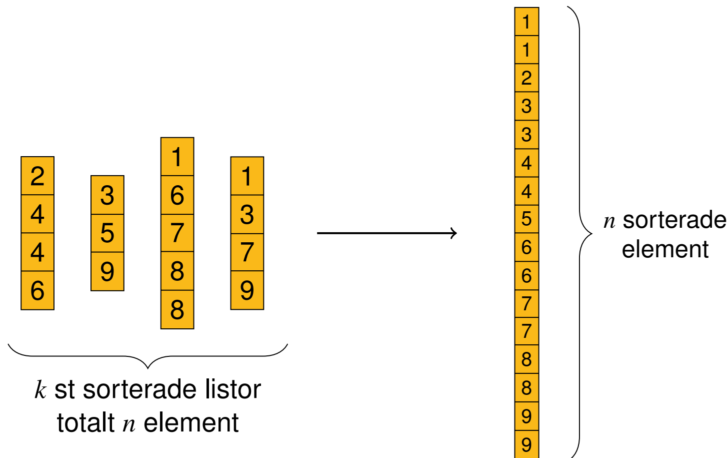
- Ö1: Algoritmanalys
- F4: Datastrukturer
- F5: Grafsökning
- **Ö2: Datastrukturer, grafer**
- F7: Giriga algoritmer, totalsökning
- F8: Dekomposition
- F9: Dynamisk programmering
- Ö3: Dekomposition och DP

Idag

- Datastrukturer
- Grafteori
- Labbteoriredovisning (innan pausen)

Uppgift 1: Samsortering

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



Uppgift 1: Samsortering

Lösningsförslag 1

- Vi löser ett enklare problem först!
- Hur löser vi problemet om $k = 2$?
    - Använd att listorna redan är sorterade.
    - Titta på det första elementet i de båda listorna.
    - Välj det minsta, och ta bort det ur listan.
    - Upprepa tills det är klart.
    - Tidskomplexitet: $\mathcal{O}(n)$
- Hur löser vi problemet för $k > 2$?
    - Slå samman listorna parvis. Nu har vi hälften så många listor!
    - Efter att ha gjort detta steg s gånger har vi $\frac{k}{2^s}$ listor kvar (ungefär).
    - Upprepa tills endast en lista finns kvar.
    - Hur många halveringar behövs? Svar: $\log k$
- Total tidskomplexitet: $\mathcal{O}(n \log k)$

Uppgift 1: Samsortering

Lösningsförslag 2

- Använd en minheap!
- Låt heapen hela tiden innehålla det minsta elementet från varje lista (k st).
- Låt en counter för varje lista säga hur många element som plockats ut.
- Upprepa så länge heapen inte är tom:
  - Plocka bort det minsta elementet ur heapen, lägg till i resultat.
  - Lägg till nästa element från listan det borttagna elementet kom från (om möjligt).
- Tidskomplexitet
  - Heapoperationer: $\mathcal{O}(\log k)$
  - Antal insättningar: n
  - Antal borttagningar: n
  - Svar: $\mathcal{O}(n \log k)$

Uppgift 2: Datastruktur med max och min

Konstruera en datastruktur som har följande operationer och komplexitet:

Operation	Komplexitet
`insert(x)`	$\mathcal{O}(\log n)$
`deletemin()`	$\mathcal{O}(\log n)$
`deletemax()`	$\mathcal{O}(\log n)$
`findmin()`	$\mathcal{O}(1)$
`findmax()`	$\mathcal{O}(1)$

- $n =$ antal lagrade element
- Jämförelser kan göras på konstant tid.

Uppgift 2: Datastruktur med max och min

Lösningsförslag 1

- Använd två heapar! En minheap och en maxheap.
- Sätt in varje element i bägge, med pekare till rätt plats i den andra heapen.
- Hur gör vi...
  - ...insert?
  - ...findmin/findmax?
  - ...deletemin/deletemax?
    - Vanlig delete från den ena, följ pekaren och ta bort från den andra.

Uppgift 2: Datastruktur med max och min

Lösningsförslag 2

- Betrakta ett balanserat sökträd.
- Problem: `findmax` och `findmin` är $\mathcal{O}(\log n)$. Vi vill ha $\mathcal{O}(1)$.
- Utöka med två variabler som alltid lagrar min/max.
- `insert`: uppdatera min/max om det behövs.
- `remove`: om min/max togs bort: hitta ny på $\mathcal{O}(\log n)$ tid och spara.

Bloomfilter

- Repetition: vad är bloomfilter?
  - Bitarray arr av längd m, initialiserad till nollor.
  - Vi har k st hashfunktioner: $f_1, f_2, \ldots, f_k$.
  - Insättning av element x:

    1: **for** $i \leftarrow 1$ **to** k **do**
    2: $\quad arr[f_i(x) \ \ mod \ \ m] \leftarrow 1$

  - För att kolla om x existerar: Kolla om alla platser som borde vara 1 är 1 (obs: risk för false positives).
  - Remove ej möjligt.

Example

Lägg till orden A, B och C i ett bloomfilter av storlek 9, givet följande tre hashfunktioner:

	A	B	C
f_1	5	2	5
f_2	0	5	8
f_3	2	7	0

Uppgift 3: Bloomfilter med borttagning

- Ett bloomfilter har bara två operationer: `insert` och `isin`.
- Uppgift: Modifiera datastrukturen så att den även har `remove` (får ej ta längre tid än `insert`).
- Lösning:
  - Låt varje plats i arrayen bestå av t.ex. 8 bitar istället för 1 och spara antalet förekomster!
  - Gör om exemplet!

	A	B	C
f_1	5	2	5
f_2	0	5	8
f_3	2	7	0

- Hur kan vi hantera overflow?

Uppgift 4: Stava

- Rättstavning. Vi har en ordlista, men den innehåller inte alla möjliga ord.
- Finns istället ett antal regler i stil med:

$$\text{-orna} \leftarrow \text{-a}, \text{-an}, \text{-or}$$

  Detta betyder:

$$X\text{a}, X\text{an}, X\text{or} \quad \text{existerar} \implies X\text{orna} \;\; \text{existerar}$$

  "orna" kallas för ingångssuffix.
- När vi ska kolla om ett ord existerar så måste vi alltså även undersöka om någon av dessa regler är uppfyllda.
- Uppgift: Hur ska vi lagra suffixreglerna för effektiv uppslagning?

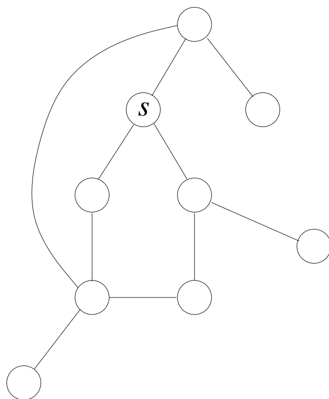
Uppgift 4: Stava

Två vettiga alternativ:

- Lagra i en array, sorterat på ingångssuffixet baklänges.
  - Förbättra med latmanshashning på sista bokstaven.
- Spara ingångssuffixen baklänges i en trie.

Uppgift 5: Grafsökning

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

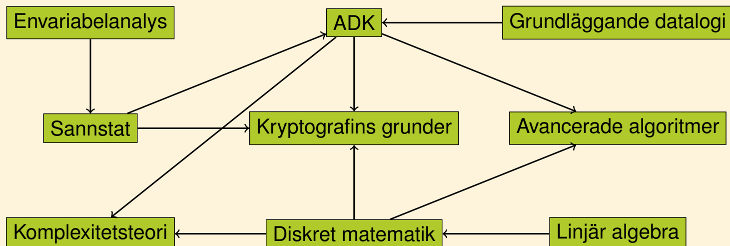


Uppgift 6: Topologisk sortering av DAG

- DAG = Directed Acyclic Graph = riktad acyklisk graf

Exempel

Kurser med förkunskapskrav kan representeras som en DAG:



Uppgift 6: Topologisk sortering av DAG

Definition

En **topologisk sortering** av en DAG är en numrering av hörnen så att alla kanter går från ett hörn med lägre nummer till ett hörn med högre nummer.

Notera!

Det kan finnas flera olika topologiska sorteringar av en och samma graf.

- Uppgift: Modifiera DFS så att den konstruerar en topologisk sortering av grafen i linjär tid.

Uppgift 6: Topologisk sortering av DAG

- Gör en vanlig DFS, fast innan du lämnar en nod, lägg till den i en stack.
- Kan behöva göra flera DFS:er tills hela grafen är besökt.
- Plocka ur noderna ur stacken i tur och ordning och lägg dem i en lista (ordningen blir omvänd).
- Varför fungerar detta?
  - När vi lämnar en kurs har vi redan besökt alla kurser som den är ett behörighetskrav till.
  - Dessa ligger då före i stacken, och kommer hamna efteråt i listan.

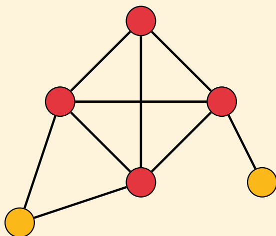
Uppgift 7: Klick

Definition

En klick i en graf är en mängd hörn, där varje hörn har en kant mellan sig.

Example

De 4 rödmarkerade noderna utgör grafens största klick, men det finns även en av storlek 3.

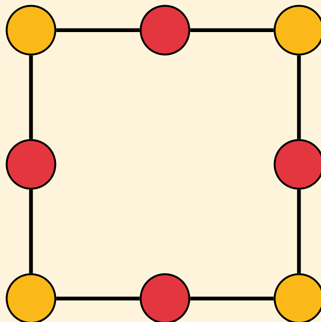


Uppgift 7: Klick

Definition

En oberoende mängd i en graf är en mängd hörn, där inget av de ingående hörnen har en kant mellan sig.

Exempel



Uppgift 7: Klick

- Ett viktigt problem är att hitta största klicken i en graf.
- Ett annat viktigt problem är att hitta största oberoende mängden i en graf.
- Dessa problem är väldigt lika varandra.

- Om man har en algoritm som hittar största klicken i en graf, så kan man utnyttja den för att hitta den största oberoende mängden i en graf. Visa detta!

Uppgift 7: Klick

Definition

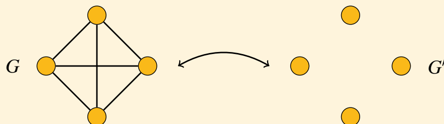
Komplementgrafen G' av en graf G är en graf med samma noder som G, men där kanterna uppfyller:

$$(u, v) \in G' \iff (u, v) \notin G$$

Notera!

Komplementgrafen till en klick är en oberoende mängd och vice versa.

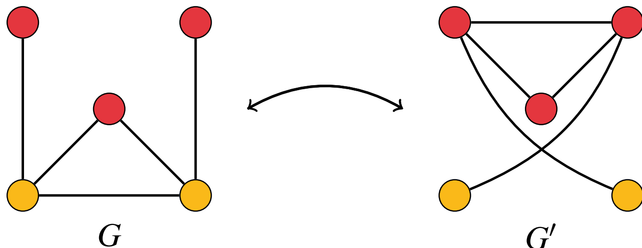
Exempel



Uppgift 7: Klick

- Den största oberoende mängden i en graf är alltså densamma som den största klicken i dess komplementgraf.
- Algoritm för att hitta största oberoende mängden:

1: **function** MAXINDEPSET(G)
2: **return** MAXCLICK(G')



Uppgift 8: Bipartithet

Definition

En bipartit graf är en graf vars noder kan delas in i två mängder A och B så att det inte finns några kanter inom A eller B (dvs alla kanter går mellan de två mängderna).

- Uppgift: Beskriv och analysera en algoritm som avgör om en graf är bipartit. Tidskomplexiteten ska vara linjär i antalet hörn och kanter i grafen.

Uppgift 8: Bipartithet

Definition

En graf är k-färgbar om vi kan färga grafens noder med k färger så att inga närliggande noder har samma färg.

- Observera: En bipartit graf är 2-färgbar. Varför?
  - Vi kan låta noderna i A få den ena färgen och noderna i B få den andra.
- Dessutom: Om en graf är 2-färgbar så är den också bipartit. Varför?
  - Vi kan låta noderna med den ena färgen vara i A och noderna med den andra färgen vara i B.
- Således: 2-färgbarhet $\iff$ bipartithet

Uppgift 8: Bipartithet

- Lösning: vi kan kontrollera om en graf är 2-färgbar genom att med hjälp av DFS/BFS försöka färga den.
- Om en 2-färgning hittas så är grafen bipartit.
- Om en 2-färgning ej hittas så existerar det ingen (varför?) och grafen är ej bipartit.

Nästa vecka

- Torsdag kl 15 i Q15.
- Algoritmkonstruktion
  - Dekomposition och dynamisk programmering.
- För att hitta mina slides, gå in på kurshemsidan och klicka på mitt namn.