Linköpings Tekniska Högskola
Institutionen för Datavetenskap
Tommy Färnqvist

18 oktober 2011

TDDC70/TDDC91 Datastrukturer och algoritmer
Tentamen 2011-10-18, 14–19 (TER1)

Examinator: Tommy Färnqvist
Jour: Tommy Färnqvist (telefon 070 4547668).
Max poäng: 28 poäng (betyg 5 = 24p, 4 = 19p, 3 = 14p)
Hjälpmedel: INGA HJÄLPMEDEL TILLÅTNA!!!

VÄNLIGEN IAKTTAG FÖLJANDE

- Lösningar till olika problem skall placeras enkelsidigt på separata blad. Skriv inte två lösningar på samma papper.
- Sortera lösningarna innan de lämnas in.
- MOTIVERA DINA SVAR ORDENTLIGT: avsaknad av, eller otillräckliga, förklaringar resulterar i poängavdrag. Även felaktiga svar kan ge poäng om de är korrekt motiverade.
- Om ett problem medger flera olika lösningar, t.ex. algoritmer med olika tidskomplexitet, ger endast optimala lösningar maximalt antal poäng.
- SE TILL ATT DINA LÖSNINGAR/SVAR ÄR LÄSBARA.
- Lämna plats för kommentarer.

Lycka till!

1. Vilka av följande påståenden är sanna och vilka är falska? Svar utan motivering ger inga poäng. (2 p)

   (a) En algoritm med tidskomplexiteten $O(\log(n!))$ går i polynomisk tid. (1)

   (b) $\max\{f(n), g(n)\} \in O(f(n)+g(n))$. Antag att $f(n)$ och $g(n)$ är icke-negativa för $n > 0$. (1)

2. Algoritmkonstruktion (4 p)
   Ett forensiskt laboratorium får en leverans av n prover. Proverna ser likadana ut, men faktum är att vissa av dem har olika kemisk sammansättning. Laboratoriet har tillgång till en maskin som kan användas för att ta reda på om två prov har olika sammansättning eller inte. Redan i förväg vet man att de flesta (mer än 50%) av proverna är identiska. Hitta ett av dessa identiska prover genom att använda jämförelsemaskinen som mest n gånger. (*Se upp*: det är fullt möjligt att två prov är identiska utan att tillhöra den stora majoriteten av identiska prover.)

3. Stackar och köer (6 p)

   (a) Beskriv hur en initialt tom stack ser ut efter varje operation i följande sekvens av operationer: (2)
   push(5), *push*(3), *pop*(), *push*(2), *push*(8), *pop*(), *pop*(), *push*(9), *push*(1), *pop*(), *push*(7), *push*(6), *pop*(), *pop*(), *push*(4), *pop*(), *pop*().

   (b) Beskriv hur en initialt tom kö ser ut efter varje operation i följande sekvens av operationer: (2)
   enqueue(5), *enqueue*(3), *dequeue*(), *enqueue*(2), *enqueue*(8), *dequeue*(), *dequeue*(), *enqueue*(9), *enqueue*(1), *dequeue*(), *enqueue*(7), *enqueue*(6), *dequeue*(), *dequeue*(), *enqueue*(4), *dequeue*(), *dequeue*().

   (c) Beskriv med pseudokod en algoritm med linjär tidskomplexitet för att vända på en kö Q. För att komma åt elementen i kön får du bara använda metoderna som finns i ADT Queue. (2)

4. Binära sökträd (4 p)

   (a) Rita ett binärt sökträd T sådant att (1)

   - varje nod i T lagrar en bokstav,
   - en traversering i *preorder* av T ger ordet EXAMFUN och
   - en traversering i *inorder* av T ger ordet MAFXUEN.

   (b) Ge en algoritm med exekveringstid $O(n)$ för att beräkna djupet av varje nod i ett träd T, där n är antalet noder i T. Antag att det finns två metoder setDepth(v,d) och getDepth(v) som använder $O(1)$ tid. (3)

5. Hashtabeller (2 p)
   Vi använder en array med indexen 0 till 4 för att implementera en hashtabell av längd 5. Rita en representation av hashtabellen och dess innehåll efter att vi använt hashfunktionen

   $$h(x) = 2x + 1 \bmod 5$$

   för att sätta in nycklarna 10, 11, 3, 5, 4, 7, 16, 22 i den initialt tomma tabellen. Antag att vi hanterar kollisioner med separat länkning (*separate chaining*).

6. Sortering (5 p)

(a) Antag att vi får en sekvens S innehållande n element, där varje element är färgat med en av k färger. Antag att S representeras med en array och ge en **in-place**-algoritm för att ordna S så att alla element med samma färger ligger i följd i arrayen. Algoritmen ska ha exekveringstid $O(nk)$ i värsta fallet. (3)

(b) Antag att vi får en sekvens S innehållande n element, där varje element är ett heltal från intervallet $[0, n^2-1]$. Beskriv ett enkelt sätt att sortera S i $O(n)$ tid. (*Tips*: fundera på olika sätt att representera elementen.) (2)

7. Grafer (5 p)

(a) Ge ett exempel på en viktad riktad graf G där det finns negativa bågvikter, men inga negativa cykler, sådan att Dijkstras algoritm på ett felaktigt sätt beräknar de kortaste avstånden från någon nod v. Illustrera exekveringen av Dijkstras algoritm för att visa vad som går snett. (2)

(b) Betrakta följande giriga strategi för att hitta kortaste vägen från *start* till *goal* i en given sammanhängande graf. (3)

i. Initialize *path* to *start*.

ii. Initialize *VisitedVertices* to {*start*}

iii. If *start* = *goal*, return *path* and *exit*. Otherwise, continue.

iv. Find the edge (*start*,*v*) of minimum weight such that v is adjacent to *start* and v is not in *VisitedVertices*.

v. Add v to *path*.

vi. Add v to *VisitedVertices*.

vii. Set *start* equal to v and go to step iii.

Hittar den här giriga strategin alltid kortaste vägen från *start* till *goal*? Ge antingen en översiktlig förklaring till varför det fungerar eller visa ett motexempel.