Linköpings Tekniska Högskola
Institutionen för Datavetenskap
Tommy Färnqvist

23 augusti 2016

TDDC91 Datastrukturer och algoritmer
Datortentamen (DAT1)
2016-08-23, 14–18

Examinator: Tommy Färnqvist
Jour: Tommy Färnqvist (telefon 013-282479).
Max poäng: 112 poäng (betyg 5 = 110p, 4 = 106p, 3 = 100p)
Hjälpmedel: Inga hjälpmedel tillåtna, förutom OpenDSA!

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. OpenDSA (100 p)

Efter inloggning i datortentasystemet finns i startmenyn för Linux Mint:

- "OpenDSA" (öppnar URL för tentamensversion av OpenDSA i Chromium) och
- "Inloggningsuppgifter OpenDSA" (öppnar sida med inloggningsuppgifter till tentamensversion av OpenDSA i Chrome).

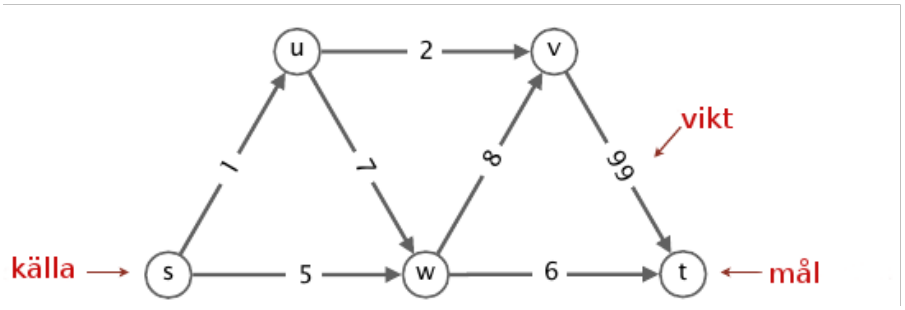
Starta tentamensversionen av OpenDSA, logga in med inloggningsuppgifterna enligt ovan och lös de anvisade uppgifterna. Genom att klicka på ditt inloggningsnamn kan du se i betygsboken hur många av de poänggivande uppgifterna du löst hittills. När du har full poäng i betygsboken är du färdig med den här uppgiften och kan logga ut. Du behöver inte skicka in något via tentamenssystemet. Om du inte löser samtliga poänggivande uppgifter får du noll poäng på den här tentamensuppgiften.

2. Grafer (4 p)

Betrakta de två följande grafproblemen:

- KortasteVäg: Givet en bågviktad riktad graf G med ickenegativa bågvikter, en källnod s och en målnod t, hitta en kortaste väg från s till t.
- KortasteTeleporteringsVäg: Givet en bågviktad riktad graf G med ickenegativa bågvikter, en källnod s och en målnod t, hitta en kortaste väg från s till t där du tillåts *teleportera* längs en båge utan kostnad. Det vill säga, en vägs totala längd är summan av alla bågvikter längs vägen utom den största.

Till exempel är den kortaste vägen från s till t (med vikt 11) $s \to w \to t$ medan den kortaste teleporteringsvägen är $s \to u \to v \to t$ (med vikt 3) i grafen nedan.



(a) Antag att du har tillgång till ett program som löser problemet KortasteTeleporteringsVäg. Om du nu får en graf G med noder s och t som du vill lösa KortasteVäg-problemet för så är ett sätt att göra det att på något sätt omvandla grafen och sedan anropa programmet som löser KortasteTeleporteringsVäg för att få reda på svaret på problemet du egentligen försöker lösa. Hur skulle du omvandla G och hur ska s och t placeras så att denna strategi ger rätt svar? Exemplifiera genom att anta att KortasteVäg-problemet du vill lösa är grafen ovan. (1)

(b) Antag att du har tillgång till ett program som löser problemet KortasteVäg. Om du nu får en graf G med noder s och t som du vill lösa KortasteTeleporteringsVäg-problemet för så är ett sätt att göra det att på något sätt omvandla grafen och sedan anropa programmet som löser KortasteVäg för att få reda på svaret på problemet du egentligen försöker lösa. Hur skulle du omvandla G och hur ska s och t placeras så att denna strategi ger rätt svar? Exemplifiera genom att anta att KortasteTeleporteringsVäg-problemet du vill lösa är grafen ovan. (3)

3. Datastrukturdesign (4 p)

Designa en effektiv datastruktur för att lagra en *trådad mängd strängar* vilken håller reda på en mängd (inga dubletter) av strängar och ordningen strängarna sattes in i enligt följande API:

- `ThreadedSet()` skapa en tom trådad mängd
- `void add(String s)` sätt in strängen `s` i mängden (om den inte redan finns i mängden)
- `boolean contains(String s)` finns strängen `s` i mängden?
- `String previousKey(String s)` returnera strängen som sattes in precis före `s` i mängden (`null` om `s` är första strängen som sattes in; undantag om `s` inte finns i mängden)

Exempel:

```
ThreadedSet set = new ThreadedSet();
set.add("aardvark");            // [ "aardvark" ]
set.add("bear");                // [ "aardvark", "bear" ]
set.add("cat");                 // [ "aardvark", "bear", "cat" ]
set.add("bear");                // [ "aardvark", "bear", "cat" ]
                                // (lägga till en dublett har ingen effekt)
set.contains("bear");           // true
set.contains("tiger");          // false
set.previousKey("cat");         // "bear"
set.previousKey("bear");        // "aardvark"
set.previousKey("aardvark");    // null
```

Beskriv dina instansvariabler och hur du skulle implementera var och en av operationerna. Vilken tidskomplexitet får de olika operationerna?

4. 4-SUM (4 p)

Betrakta 4-SUM-problemet: Givet N heltal, finns det fyra bland dem som summerar till noll?

(a) Vilken är tiskomplexiteten för följande brute forcelösning till problemet? (1)

```
boolean fourSum(int[] a) {
   int N = a.length;
   for (int i = 0; i < N; i++)
      for (int j = i+1; j < N; j++)
         for (int k = j+1; k < N; k++)
            for (int l = k+1; l < N; l++)
               if (a[i] + a[j] + a[k] + a[l] == 0) return true;
   return false;
}
```

(b) Konstruera en algoritm som löser 4-SUM med tidskomplexitet $\mathcal{O}(N^2)$ och minneskomplexitet $\mathcal{O}(N^2)$. Antag att du har tillgång till en hashtabell där `put()` och `get()` exekverar i konstant tid för heltalsnycklar. (3)