MATEMATISKA INSTITUTIONEN
STOCKHOLMS UNIVERSITET
Avd. Beräkningsmatematik
Examinator: Lars Arvestad

Tentamensskrivning i
DA3018 Datalogi för matematiker
7.5 hp
2019-08-23

- Inga hjälpmedel tillåtna.
- **Skriv tydligt.** Svårlästa svar riskerar 0 poäng.
- Skriv bara på **en sida av varje papper**!
- **Motivera** alla svar (om inte annat anges)!
- **Betygsgränser:** E: 25, D: 30, C: 35, B: 40, A: 45

---

1. (a) Nämn två egenskaper som man kan förvänta sig av en bra hash-funktion. (2p)

(b) Varför använder man ordo-notation (dvs $O(f(n))$) för tidskomplexitet? (2p)

(c) Ge en fördel och en nackdel med att matrisrepresentation för grafer. (2p)

(d) Vad är tidskomplexiteten för att avgöra om en lista är sorterad eller ej? (2p)

(e) Beskriv datastrukturen *stack*. (2p)

2. Figur 1 har pseudokod för att bygga en uppslagstabell. Parametrarna `elements` och `data_list` är vanliga enkel-länkade listor av okänd storlek.

(a) Gör en tidskomplexitetsanalys för funktionen `build_lookup_table`. (3p)

(b) Antag att man vill lösa följande problem:

- In: en sträng med text och en lista med nyckelord
- Ut: en uppslagstabell där man kan se hur många positioner varje nyckelord finns på.

Varför är `build_lookup_table` inte lämplig för det problemet? (2p)

3. Figur 2 har ett utkast till kod för urvalssortering (eng: *selection sort*). Tyvärr har koden ett fel.

(a) Rätta felet så att funktionen returnerar en sorterad lista. (2p)

(b) Gör en analys av tidskomplexiteten för den korrekta versionen av urvalssortering. (2p)

(c) Vad måste du tänka på om du ska implementera denna version av urvalssortering och vill se till att sorteringen är stabil? (2p)

4. (a) Skapa och illustrera en max-heap för poängen (ta inte med lagnamnen) i tabell 1. (2p)

(b) Hur tar man bort element från en heap i tid $O(\log n)$? Exemplifiera standardalgoritmen genom att ta bort Rosengårds resultat. (3p)

(c) Kan operationen `find(key)` implementeras i tid $O(\log n)$ för en heap? Om ja, beskriv hur; om nej, argumentera för varför det inte går. (2p)

5. Antag att du vill lagra data för fotbollslag i ett binärt sökträd.

(a) Rita ett *balanserat* binärt sökträd för lagen i tabell 1 och använd lagnamnen som nycklar. Alfabetisk ordning gäller. Argumentera för att ditt träd är balanserat. (3p)

(b) Ge pseudokod för en rekursiv funktion som skriver ut nycklarna i ett binärt sökträd i ordning. För sökträdet i uppgift (a) ska alltså ”Djurgårdens IF” skrivas ut först. (4p)

(c) Ge pseudokod för en funktion `check_if_present(key, bst)` som returnerar `True` om `key` finnst i det binära sökträdet `bst`, och `False` annars. (2p)

(d) Härled den asymptotiska tidskomplexiteten för `check_if_present` under antagandet att `bst` är balanserat. (2p)

6. Optimeringsproblemet *Oberoende mängd* (eng: *Independent set*) går ut på att, given en graf $G = (V, E)$ hitta en så stor delmängd $V_O \subseteq V$ som möjligt så att för varje par $v, w \in V_O$ gäller att $(v, w) \notin E$.

I figur 3 finns en heuristik för att hitta en oberoende mängd.

(a) Ge en kort och kärnfull beskrivning i ord, som till en kollega, för hur heuristiken fungerar. (2p)

(b) Gör en analys av tidskomplexiteten för heuristiken under rimliga antaganden om hur pseudokoden implementeras i ett verkligt programmeringsspråk. (4p)

7. Antag att en kompis vill annonsera på internet och köper den tjänsten från en leverantör som kan associera annonsen med vissa nyckelord under en vecka. Kompisen får en lång lista med nyckelord och vad de kostar att vara associerade med samt en uppskattning på hur många individer som antagligen kommer att vara intresserade av nyckelordet. Hen vill ha så bra räckvidd för sin annons som möjligt, men har bara råd att lägga en mindre summa pengar på annonsen och ber dig om hjälp.

Formulera ett *beräkningsproblem* för uppgiften. (5p)

```
def build_lookup_table(elements, data_list):
    t = new Dictionary()     // Works like a hash table
    for e in elements:
        position = data_list.index_of(e) // Locate e in the list
        if position >= 0:
            t[e] = position
    return t
```

Figur 1: Pseudokod till fråga 2. Syftet med funktionen är att skapa en tabell där man kan slå upp på vilket index i `data_list` ett element finns, om det finns med i `data_list`. Vi antar att `index_of` returnerar ett negativt värde om elementet inte hittas i `data_list`. Parametrarna `elements` och `data_list` är vanliga enkel-länkade listor. Datastrukturen `Dictionary` fungerar som **`dict`** i Python eller `HashMap` i Java.

```
def selection_sort(input_list):
    sorted_list = []
    n = len(input_list)
    for i from 0 to n-1:
        elem = min(input_list)   // Find smallest elem
        sorted_list.append(elem) // Put it last in the output list
    return sorted_list
```

Figur 2: Trasig pseudokod för Urvalssortering.

```
def independent_set(V, E):
   V_chosen = emptyset()
   V_discarded = emptyset()
   L = sort_in_order_of_increasing_degree(V)
   for v in L:
      if v not in V_discarded:
         V_chosen = union(V_chosen, {v})
         for w such that (v,w) in E:
            V_discarded = union(V_discarded, {w})
   return V_chosen
```

Figur 3: Pseudokod för uppgift 6.

Lag	**Poäng**
FC Rosengård	30
Kopparberg FC	26
Vittsjö GIK	25
KIF Örebro	23
Linköpings FC	22
Kristianstads DFF	22
Piteå IF	20
Eskilstuna United	17
Växjö DFF	15
Djurgårdens IF	9
IF Limhamn	4
Kungsbacka DFF	4

Tabell 1: Aktuell ställning i Damallsvenskan.