Lösningsförslag Tenta

Niclas Börlin

2023-05-03

Contents

1 Identifiera algoritmer (10p) 2

2 Gränssnitt för Lista (16p) 3

3 Pseudokod 4
3.1 Nycklar i tabell (15p) . . . 4
3.1.1 Lösningsförslag . . . 5
3.1.2 Bedömning . . . 5
3.2 Komplexitet för Table-key-list (3p) . . . 5

4 Hashtabell (15p) 6

5 Problemlösningsstrategier (10p) 6

6 Tabell 7
6.1 Tabell, relation (6p) . . . 7
6.2 Egenskaper för Tabell (9p) . . . 7

7 Grafer, grafalgoritmer 8
7.1 Minsta uppspännande träd . . . 8
7.2 Kruskals algoritm (12p) . . . 9

1 Identifiera algoritmer (10p)

```
Algorithm foo(a: Array, u: Int)

for i from Low(a) to High(a) do
  if Inspect-value(a, i) = u then
    return i

return Low(a) - 1
```

```
Algorithm bar(a: Array, u: Int)

i ← Low(a)
j ← High(a)
while j - i > 0 do
  k = baz((i + j) / 2)
  if Inspect-value(a,k) = u then
    return k
  else if Inspect-value(a, k) < u then
    i ← k + 1
  else
    j ← k - 1

return Low(a) - 1
```

```
Algorithm baz(i: Int)

if not (i mod 2 = 0) then
  i ← i - 1

return i / 2
```

Betrakta pseudokoden till de tre algoritmerna `foo`, `bar` och `baz` till vänster.

a) Vilken algoritm svarar `foo` mot? **Linjärsökning**

b) Vilken algoritm svarar `bar` mot? **Binärsökning**

c) Vilken förenklad asymptotisk komplexitet har `foo`? $O(n)$

d) Vilken förenklad asymptotisk komplexitet har `bar`? $O(\log n)$

e) För en av algoritmerna måste heltalen i `a` vara ordnade på ett speciellt sätt. Vilken algoritm, och hur måste värdena vara ordnade? **bar, sorterade i stigande ordning**

2 Gränssnitt för Lista (16p)

Nedan finns 10 frågor av typen sant/falskt. De är uttalandena om gränssnittsfunktioner för den abstrakta datatypen List(val).

Rätt svar ger 1.5 p per fråga, felaktigt svar ger -1.5 p och inget svar alls ger 0p. Lägsta totalpoäng på frågorna är 0p. Högsta totalpoäng på frågorna är 16 p.

1. Funktionen `First(l: List(val))` returnerar den aktuella positionen i listan `l`. **Falskt**, det finns ingen aktuell position i en lista.
2. Funktionen `First(l: List(val))` returnerar den första positionen i listan `l`. **Sant**.
3. Funktionen `End(l: List(val))` returnerar positionen som följer efter det sista elementet i listan `l`. **Sant**.
4. Funktionen `End(l: List(val))` returnerar positionen för det sista elementet i listan `l`. **Falskt**, se förra frågan.
5. Positionen som returneras av funktionen `End(l: List(val))` ändras när element sätts in eller tas bort i listan `l`. **Sant**.
6. Positionen som returneras av funktionen `First(l: List(val))` ändras när element sätts in eller tas bort i listan `l`. **Falskt**, `First()` förändras inte.
7. För en tom lista `l` gäller att `First(l) = End (l)`. **Sant**.
8. För en tom lista `l` gäller att `First(l)` och `End (l)` är odefinierade. **Falskt**, `First()` och `End()` är alltid definierade för en lista.
9. Funktionen `Remove(p: pos, l: List(val))` returnerar positionen efter det element som `Remove raderat`. **Sant**, se den informella specifikationen för Lista.
10. Om `p = End(l)` så är resultatet av anropet `Remove(p: pos, l: List(val))` odefinierat. **Sant**, `End(l)` är positionen **efter** det sista elementet, så det finns inget element vid den positionen.

3 Pseudokod

3.1 Nycklar i tabell (15p)

```
abstract datatype Table(arg, val)
  Empty() → Table(arg, val)
  Insert(k: arg, v: val, t: Table(arg, val)) → Table(arg, val)
  Isempty(t: Table(arg, val)) → Bool
  Lookup(k: arg, t: Table(arg, val)) → (Bool, val)
  Remove(k: arg, t: Table(arg, val)) → Table(arg, val)
  Choose-key(t: Table(arg, val)) → arg

abstract datatype DList(val)
auxiliary pos
  Empty() → DList(val)
  Isempty(l: DList(val)) → Bool
  First(l: DList(val)) → pos
  Next(p: pos, l: DList(val)) → pos
  Isend(p: pos, l: DList(val)) → Bool
  Inspect(p: pos, l: DList(val)) → val
  Insert(v: val, p: pos, l: DList(val))
                                → (DList(val), pos)
  Remove(p: pos, l: DList(val)) → (DList(val), pos)
```

Låt den abstrakta datatypen Tabell ha gränsytan till vänster. Funktionen `Table-choose-key(t)` returnerar en godtycklig nyckel som finns i tabellen `t`. Om tabellen `t` är tom är returvärdet odefinierat.

Skriv en algoritm i pseudokod med huvudet `Table-key-list(t: Table)` som tar en Tabell `t` som inparameter och returnerar en Riktad lista (DList) med alla nycklar som lagras i tabellen. Operatorn (←, vänsterpil, tilldelning, kopiering) är definierad för enkla datatyper, men inte för sammansatta. Om tilldelningsoperatorn appliceras på en sammansatt datatyp så kopieras en *referens* till objektet. Du kan utgå från att typerna `arg` och `val` är enkla.

När algoritmen är klar ska `t` ha samma innehåll som när algoritmen startade. Var noggrann med att visa hur du tar hand om alla returvärden som behövs från alla funktioner. Du får bara använda funktioner i gränsytorna till Table respektive DList.

Algoritmen ska ha följande huvud: `Algorithm Table-key-list(t: Table)`

Det som kommer att bedömas är

- att pseudokoden löser uppgiften under de givna förutsättningarna,
- att pseudokoden är fri från språkspecifika konstruktioner (inga `i++` eller `for i in range(...)`, etc.),
- att koden är korrekt indenterad,
- att koden är rimligt kommenterad, och
- om koden har optimal komplexitet ($g(n)$ är viktigast).

3.1.1 Lösningsförslag

```
Algorithm Table-key-list(t: Table)
// Return a directed list of all keys in the Table t.
// The Table t is unchanged after the function.

// Start with an empty key list
kl ← DList-empty()

// This will become the copy of the table
tc ← Table-empty()

while not Isempty(t) do
  // Pick an arbitrary key
  k ← Choose-key(t)
  // Extract the value associated with the key
  v ← Lookup(k, t)
  // Put the key in the key list
  kl ← DList-insert(k, First(kl), kl)
  // Put the key-value pair in the table copy
  tc ← Insert(k, v, tc)
  // Remove the key-value pair from the original table
  t ← Remove(k, t)

// Now the input table is empty
// Restore the input table
t ← tc

// Return the key list
return kl
```

3.1.2 Bedömning

Vanliga saker som gav avdrag:

- Språkspecifika saker, t.ex. for-loopar à la Python eller måsvingar á la C. Håller er till PSEUDO-kod (c:a 1/3 av poängen avdrag).
- Sammanblandning av index/position. Se till att ni har koll på vad man får göra med index till Fält resp. positioner i Listor (typ halva poängen ryker).
- Inte tar hand om returvärden från t.ex. Table-insert, List-remove e.dyl. Var noga med att ta hand om alla returvärden som behövs (2p avdrag per gång vilket kan bli en del).
- Antag inte att Tabell har heltal som nycklar (3-4p avdrag).
- In-tabellen återställdes aldrig (halva poängen).
- Glömmer bort att returnera resultat från en algoritm (1p avdrag).
- Anrop till Table-choose-key trots att tabellen var tom (3-4p avdrag).
- Användande av typnamn vid funktionsanrop (3-4p avdrag).
- Löser ej uppgiften (0p förstås).

3.2 Komplexitet för Table-key-list (3p)

Vilken absolut tidskomplexitet har din algoritm från förra uppgiften? Funktionerna `Table-lookup()` och `Table-remove()` har komplexiteten $O(n)$, där är antalet nyckel-värde-par i tabellen. Övriga Tabell-operationer och alla DList-operationer har komplexiteten $O(1)$.

Svar: $O(n^2)$. Vi gör en `Lookup()` och `Remove()` för varje nyckel i `t`. Totalt n anrop á $O(n)$, dvs. totalt $O(n^2)$.

4 Hashtabell (15p)

I en hashtabell avbildar man en stor mängd nyckelvärden på en mindre mängd tal. Detta leder till att kollisioner är oundvikliga. Kollision kan dock hanteras med **sluten** hashning. En sådan hashtabell kommer att ha ett statiskt antal element. Kollisioner kan också hanteras med **öppen** hashning. En sådan hashtabell kommer att ha ett **dynamiskt** antal element.

5 Problemlösningsstrategier (10p)

Travelling salesman-problemet går ut på att hitta den kortaste rutten som besöker alla städer med vägar mellan dem. Lisa, som gått DoA-kursen, vet förstås att det kan ta väldigt lång tid att lösa problemet om man inte gör nånting smart, så Lisa implementerade följande algoritm:

1. Skapa en graf där städerna är noder och vägarna mellan dem är bågar
2. Sätt alla noder som obesökta
3. Välj ut en godtycklig nod, sätt den som aktuell nod n och markera den som stängd
4. Hitta kortaste bågen som förbinder n och en obesökt nod g
5. Sätt g som aktuell nod n och markera g som stängd
6. Om alla noder har besökts: avsluta. Annars gå till punkt 4.

a) Vilken typ av lösningsteknik använder den här algoritmen (5p)? **Girig algoritm**

b) Vilken problemlösningsstrategi använder man om man löser större och större problem tills man har löst sitt problem av storlek n, och använder information från tidigare lösningar för varje steg? (5p) **Dynamisk programmering**

6 Tabell

6.1 Tabell, relation (6p)

```
abstract datatype Table(arg, val)
  Empty() → Table(arg, val)
  Insert(k: arg, v: val, t: Table(arg, val)) → Table(arg, val)
  Isempty(t: Table(arg, val)) → Bool
  Lookup(k: arg, t: Table(arg, val)) → (Bool, val)
  Remove(k: arg, t: Table(arg, val)) → Table(arg, val)
  Choose-key(t: Table(arg, val)) → arg
```

Gränssnittet till den abstrakta datatypen Tabell beskrivs i panelen till vänster. Låt `arg` vara argumenttypen (nyckeltypen) och `val` vara tabellvärdetypen.

a) Vilken relation `R` måste vara definierad för att `Table-lookup()` ska gå att implementera? **Likhet**

b) För vilken typ ska relationen `R` vara definierad? Likhetsrelationen måste vara definierad för nyckeltypen `arg`.

6.2 Egenskaper för Tabell (9p)

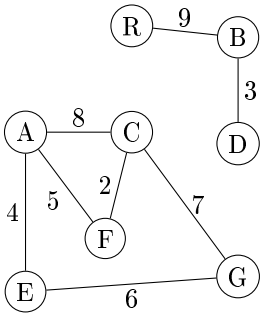
Vilken/vilka av nedanstående egenskaper har den abstrakta datatypen Tabell?

	Sant	Falskt
Statisk		X
Homogen	X	
Ordnad		X
Sorterad		X
Rekursiv		X

Not: Tabellen är homogen då den innehåller **par** (nyckel, värde), där alla nycklar är av samma typ och alla värden är av samma typ. Däremot kan nyckeltypen och värdetypen vara olika.

7 Grafer, grafalgoritmer

7.1 Minsta uppspännande träd



Denna uppgift handlar om algoritmer för att beräkna minsta uppspännande träd i den viktade grafen till vänster. Om någon algoritm behöver en startnod ska noden **B** användas.

Antag att vi applicerar **Prims** algoritm på grafen.

- Hur många noder ingår i resultatet? **3**
- Hur många bågar ingår i resultatet? **2**

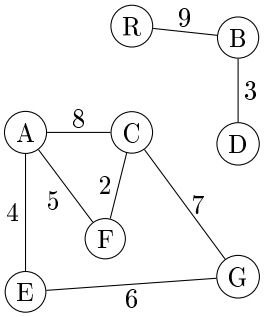
Not: Prims algoritm konstruerar ett uppspännande träd för den sammansatta komponent som innehåller startnoden (**B**). I detta fall kommer trädet att ha 3 noder och 2 bågar.

Antag att vi applicerar **Kruskals** algoritm på grafen.

- Hur många noder ingår i resultatet? **8**
- Hur många bågar ingår i resultatet? **6**

Not: Kruskals algoritm konstruerar ett uppspännande träd för varje sammansatta komponent i grafen. Därför kommer alla 8 noder att ingå i den uppspännande ”skogen”, med totalt 6 bågar.

7.2 Kruskals algoritm (12p)



Vi ska nu studera Kruskals algoritm steg för steg, applicerad på grafen till vänster. När algoritmen ska välja en färg för omfärgning, ska omfärgningen göras med den färg som tillhör trädet med den bokstav som kommer först i alfabetet.

1. I steg 1 av huvudloopen, vilken/vilka noder kommer att få förändrad färg? **C och F** Bågen C-F har lägst vikt (2) och kommer därför att väljas först. Bägge noderna är ofärgade och kommer därför att få en ny färg, t.ex. röd.

2. I steg 2 av huvudloopen, vilken/vilka noder kommer att få förändrad färg? **B och D** Bågen B-D har näst lägst vikt (3). Bägge noderna är ofärgade och kommer därför att få en ny färg, t.ex. blå.

3. I steg 3 av huvudloopen, vilken/vilka noder kommer att få förändrad färg? **A och E** Bågen A-E kommer härnäst med vikten 4. Bägge noderna är ofärgade och kommer därför att få en ny färg, t.ex. grön.

4. I steg 4 av huvudloopen, vilken/vilka noder kommer att få förändrad färg? **C och F** Bågen A-F kommer härnäst med vikten 5. Då bägge noderna är färgade med olika färg ska vi att färga om ett träd. Enligt först-i-alfabet-regeln så kommer A före C och vi ska därför färga om C-F-trädet till grönt.

5. I steg 5 av huvudloopen, vilken/vilka noder kommer att få förändrad färg? **G** Bågen E-G kommer härnäst med vikten 6. Då noden E är färgad men inte G så kommer vi att färga G grönt.

6. I steg 6 av huvudloopen, vilken/vilka noder kommer att få förändrad färg? **Ingen** Bågen C-G kommer härnäst med vikten 7. Då bägge noderna är färgade med samma färg så ignorerar vi bara bågen.