Lösningsförslag och rättningsmall för DoA-tentan

Niclas Börlin

2024-05-13

1. Grundbegrepp (12)

1.1 Egenskaper för Fält (3)

Syftet med denna uppgift är att examinera kunskap kring grundläggande egenskaper för datatypen Fält samt skillnaden mellan element och elementvärden.

Betrakta ett Fält med lägsta index a och högsta index b, där $b \geq a$.

a) Hur många giltiga index n har Fältet? Skriv ett uttryck i a och b.

På nedanstående frågor kan du använda n från uppgift a) i svaren.

b) Vilket är det minsta antalet **element** som Fältet kan innehålla?

c) Vilket är det största antalet **element** som Fältet kan innehålla?

d) Vilket är det minsta antalet **elementvärden** Fältet kan innehålla?

e) Vilket är det största antalet **elementvärden** Fältet kan innehålla?

Rätt svar

a) $n = b - a + 1$. Exempelvis om $a = 10$ och $b = 14$ så är $10, 11, 12, 13, 14$ giltiga index och $n = 5$.

b) och c) Ett Fält innehåller alltid n *element*, men antalet *elementvärden* kan variera.

d) Det minsta antalet elementvärden är 1 eller 0 beroende på om vi räknar det odefinierade värdet som ett värde eller inte. Jag har gett rätt för bägge. Exempelvis har Fältet med värdena [3,3,3,3] 4 element men 1 elementvärde.

e) Det största antalet elementvärden Fältet kan innehålla är n. Då innehåller varje element ett unikt värde, t.ex. Fältet [2,3,5,8].

Bedömning

- Om man angett fel svar på a), t.ex. $n = a - b$ och sedan använt $a - b$ på rätt ställen i uppgifterna b)-e) så har jag gett fel på a) men rätt på de övriga.
- Några har svarat $n = b + 1$, vilket är korrekt om $a = 0$, vilket är fallet i C men inte generellt.
- Uppgiften hade 0.5p för varje rätt svar och 3p om alla rätt.
- Notera att autorättningen inte godkänt $(b - a) + 1$ m.fl. då det inte ingick bland de korrekta strängar jag gissade att ni kunde välja. Jag tror jag rättat upp alla korrekta svar manuellt.

1.2 Egenskaper för Lista (3)

Syftet med denna uppgift är att examinera kunskap kring grundläggande egenskaper för datatypen Lista samt skillnaden mellan element och elementvärden.

Betrakta en Lista med n element:

a) Hur många giltiga positioner har Listan?

b) Vilket är det minsta antalet **elementvärden** som Listan kan innehålla?

c) Vilket är det största antalet **elementvärden** som Listan kan innehålla?

Rätt svar

a) En Lista med n element har alltid $n + 1$ giltiga positioner, där den ”extra” positionen är *efter* det sista elementet.

b) Det minsta antalet elementvärden är 1 om alla elementen innehåller samma värde.

c) Det största antalet elementvärden är n. Då innehåller varje element ett unikt värde.

Bedömning

- Hade missat att slå ”ignorera vita fält” (whitespace), så autorättningen underkänner `n + 1`. Jag ska ha rättat upp alla såna manuellt.
- Uppgiften ska ha 1p per svar. Auto-rättningen var inställd på 0.5p per rätt svar (3 för alla rätt). Jag ska ha rättat upp alla manuellt.

1.3 Egenskaper för datatyp (4)

Syftet med denna uppgift är att examinera kunskap kring grundläggande egenskaper hos datatyper.

Antag att du vet att den datatypen **Prioritetskö** är konstruerad som en **Hög**. Baserat på denna information, vilka av nedanstående egenskaper har **Prioritetskön**?

Denna uppgift har minuspoäng för fel svar. Du kan behöva ladda om sidan för att nollställa dina svar.

Rätt svar

	Sant	Falskt	Partiellt	Hierarkiskt	Okänt
Sammansatt	X				
Ordnad	X				
Sorterad	X				
Implementerad					X

Rättning

- Hur datatypen är **konstruerad** spelar ingen roll, dvs. Prioritetskö ska uppföra sig som en Prioritetskö oberoende om den är konstruerad som Hög, Lista, eller något annat.
- Prioritetskön är **sammansatt** — innehåller element.
- Prioritetskön är **sorterad** enligt prioriteringsordningen. (En Hög är partiellt sorterad, men frågan gäller Prioritetskön.)
- Prioritetskön är automatiskt **ordnad** eftersom den är sorterad. (En Hög är heirarkiskt ordnad, men frågan gäller Prioritetskön.)
- Det är **okänt** om Prioritetskön är *implementerad*. Att den är *konstruerad* betyder bara att vi bestämt hur datat ska lagras internt. Jag har också godkänt svaret Falskt, även om vi faktiskt inte *vet* att den inte är implementerad.

1.4 Egenskaper för problem (4)

Vad gäller för ett problem vars snabbaste lösningsalgoritm har komplexitet $O(2^n)$?

Denna uppgift har minuspoäng för fel svar. Du kan behöva ladda om sidan för att nollställa dina svar.

Rätt svar

	Sant	Falskt
Problemet är beräkningsbart	X	
Problemet är hanterligt		X

- Det existerar en algoritm som löser problemet, alltså är problemet beräkningsbart.
- Komplexiteten $O(2^n)$ är super-polynomiellt, alltså ohanterligt.

2. Binära sökträd (15)

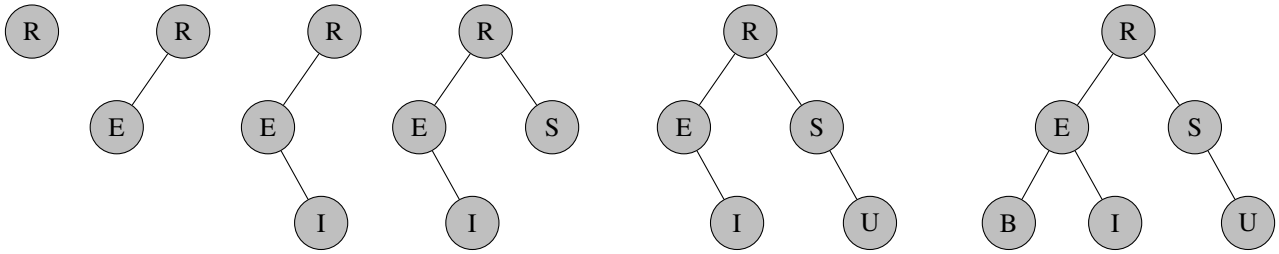
Syftet med denna uppgift är att examinera förståelsen av egenskaper för Relationer och Binära sökträd samt förmågan att skapa ett Binärt sökträd.

Stoppa in bokstäverna R, E, I, S, U, B i ett binärt sökträd, i den ordningen, enligt den binära relationen R. Relationen R ("i bokstavsordning") är definierad för bokstäver och är sann för alla par av bokstäver x och y sådana att x kommer före y i det svenska alfabetet. Gör endast insättning, dvs. gör inga försök att balansera trädet.

Betrakta det binära sökträdet efter insättning av bokstäverna ovan.

Lösningsförslag

Så här ser sökträden ut efter varje steg:



Frågor och rätt svar

a) **Hur många noder** har den längsta grenen i det resulterande trädet?

- **Svar:** 3

b) Följ den **längsta grenen** från roten till lövet. Skriv in etiketterna, separerade med bindestreck. Om det finns flera grenar av samma längd, välj en:

- **Svar:** Någon av R-E-B, R-E-I, R-S-U

c) Ange etiketterna på **löven**, från vänster till höger. Separera dem med bindestreck.

- **Svar**: B-I-U

d) Vilken är den optimala höjden för ett binärt träd med 6 noder?

- **Svar**: 2 (godkänner även 3 då jag skrivit fel på en slide)

e) Vilken traverseringsordning ska du använda för att skriva ut noderna i det binära sökträdet i ordningen som bestäms av R?

- **Svar**: Inorder eller Djupet-först, inorder.

3. Listalgoritmer (15+6)

Syftet med denna uppgift är att examinera förmågan att skriva algoritmer i pseudokod utifrån en given gränsyta samt att examinera förståelsen för listalgoritmer, inklusive deras komplexitet.

```
abstract datatype DList(val)
auxiliary pos
  Empty() → DList(val)
  Isempty(l: DList(val)) → Bool

  Copy(l: DList(val)) → DList(val)

  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)

  Kill(l: DList(val)) → ()
```

Gränsytan till Riktad lista (*Directed list*, DList) anges i panelen till vänster. Notera att gränssnittet inkluderar en funktion `Copy()` som returnerar en kopia av inlistan. Funktionen `Copy()` har komplexiteten $O(n)$, där n är antalet element i listan. Övriga funktioner i gränsytan har komplexitet $O(1)$. Dessutom finns funktionen `Value-isequal(a, b: Value)` definierad som returnerar `True` om `a` och `b` anses lika. Funktionen `Value-isequal()` har komplexiteten $O(1)$.

a) Skriv en algoritm (funktion) `Find-value(v: Value, l: Directed list)` som returnerar `True` om värdet `v` finns i listan `l`, annars `False`.

b) Skriv en funktion `Set-union-no-duplicates(s, t: DList)` som tar två Riktade listor som representerar två mängder och returnerar *unionen* av mängderna, dvs. mängden av alla element som finns i minst en av mängderna `s` och `t`. Ingen av in- eller ut-listorna får innehålla dubletter.

c) Skriv en funktion `Set-union-with-duplicates(s, t: DList)` som har exakt samma funktion som `Set-union-no-duplicates()`. Skillnaden är att dubletter är **tillåtna** i in- och ut-listorna.

Dina lösningar ska ha optimal komplexitet (på $g(n)$-nivå). I uppgift b) och c), använd algorimen `Find-value` från uppgift a) om så behövs.

Fundera på vilket fall som är enklast — med eller utan dubletter? I fallet utan dubletter kan du anta att in-listorna saknar dubletter men måste se till att ut-listan saknar dubletter. I fallet med dubletter måste du hantera att in-listorna kan innehålla dubletter, men måste inte tillse att ut-listan är dublettfri.

Var noggrann med att visa hur du tar hand om returvärden från funktionerna. Förutom `Value-isequal()` får du bara använda funktioner i gränsytan till DList.

Det som kommer att bedömas är:

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

d) Antag att listorna i genomsnitt har samma antal element n. Vilken komplexitet har dina funktioner som funktion av n?

- `Find-value()`
  - **Svar:** $O(n)$ då listan traverseras en gång.
- `Set-union-no-duplicates()`
  - **Svar:** $O(n^2)$ då `Find-value()` anropas för varje element i en av listorna.
- `Set-union-with-duplicates()`
  - **Svar:** $O(n)$ då lösningen traverserar vardera listan en gång

Lösningsförslag

```
Algorithm Find-value(v: Value, l: Directed list)
// Return True if v is found in s, otherwise False

p ← First(l)

while not Isend(p, l) do
  if Value-isequal(v, Inspect(p, l)) then
    return True
  p ← Next(p, l)

return False

Algorithm Set-Union-no-duplicates(s, t: Directed list)

// First, copy s to the output
u ← Copy(s)

// Then, copy all new element values from t to u
q ← First(t)
while not Isend(q, t) do
  v ← Inspect(q, t)
  // Only insert the value if it is not already in u
  if not Find-value(v, u) then
     u ← Insert(v, First(u), u)
  q ← Next(q, t)

// Return the merged list
return u

Algorithm Set-Union-with-duplicates(s, t: Directed list)

// First, copy s to the output
u ← Copy(s)

// Now, copy all new element values from t to u
q ← First(t)
while not Isend(q, t) do
  v ← Inspect(q, t)
  // Insert value without checking since duplicates are allowed
  u ← Insert(v, First(u), u)
  q ← Next(q, t)

// Return the merged list
return u
```

Bedömning

- 5p per funktion
- En del inlämnade lösningar har beräknat **snittet** av två mängder, dvs. mängden av alla elementvärden som ingår **både** i s och t. Uppgiften var att konstruera **unionen** av två mängder, dvs. mängden av alla elementvärden som ingår i minst en av mängderna. Detta har vanligtvis gett noll poäng då koden inte löser uppgiften. I de fall någon varit nära en gräns har jag tittat igenom koden igen för att se om jag kunde hitta tillräckligt mycket kunskap för att samla ihop poäng till nästa betyg.
- En del skrev en egen kopieringsloop trots att `Copy()` fanns i gränsytan. Inget avdrag då jag inte krävt att ni skulle använda `Copy()`.
- Däremot avdrag om man skrivit en egen sökalgoritm på b) eller c), då uppgiften specificerade att ni skulle använda `Find-value()` om den behövdes.
- Allvarliga fel:
  - Använda heltal som position, dvs. blanda ihop index och positioner. **Ger automatisk noll poäng på hela uppgiften.**
  - Använder en position som tillhör en lista i en annan. **Ger också noll poäng.**
  - Sammanblandning av typer alt. typer och variabler, t.ex. returnera en lista när man ska returnera ett heltal. Ger stora avdrag.
  - C-specika lösningar, t.ex. ++. Ger stora avdrag.
  - Fel loopkonstruktioner, if där det borde vara while, etc.
  - Iterationsuppdatering next() utanför while-loopen i stället för inuti.
  - Flera lösningar där man jämfört värdet i en position med nästa, vilket alltid fick problem i slutet på listan (next(end()) är odefinierat).
  - Avancerar inte alltid i listan.
  - **Total avsaknad av kommentarer och indentering => noll poäng.**
- Mindre allvarliga
  - Funktionerna ska klara tomma listor som input.
  - Felanvändning av gränsytans funktioner, t.ex. Isend(l) utan positionsparameter
  - Tar ej hand om returvärden från t.ex. Insert()/Remove().
  - Använder positioner som ej är definierade efter Insert()/Remove().
  - Tilldelningsoperationen använd för flera olika saker
- Har man inte lämnat in någon lösning eller en gravt felaktig sådan så blir det i allmänhet inga poäng på komplexitetsuppgiften.

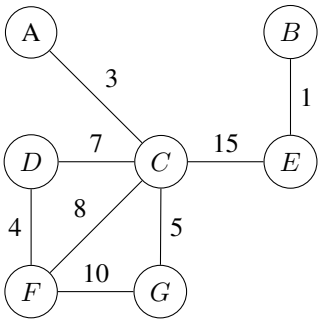
4. Sortering (10)

	Bubble	Merge	Quick
Är en rekursiv algoritm		X	X
Använder ett pivåelement			X
Finns i stabil version som kräver endast $O(1)$ minne	X		
Har medelkomplexitet $O(n \log n)$		X	X
Har värstafallskomplexitet $O(n \log n)$		X	
Har bästafallskomplexitet som är snabbare än $O(n \log n)$	X		X

- Bubblesort har värstafallskomplexitet $O(n^2)$ på omvänt sorterat data.
- Quicksort har värstafallskomplexitet $O(n^2)$ på redan sorterat respektive omvänt sorterat data, om pivåelementet är första eller sista elementet. För andra val av pivåelement så ger andra indata $O(n^2)$. Bästafallskomplexiteten är $O(n)$ för data som består av (nästan) samma värde.
- Bubblesort har bästafallskomplexitet $O(n)$ för redan sorterat data.
- Quicksort har bästafallskomplexitet $O(n)$ om alla element har samma värde.

5. Grafer, grafalgoritmer (7+5)

5.1 Kruskals algoritm



Syftet med denna uppgift är att examinera förmågan att applicera en grafalgoritm - Kruskals - på en given graf.

Applicera Kruskals algoritm på grafen till vänster. I varje varv av huvudloopen i Kruskals algoritm väljs en båge ut. Ett beslut tas sedan om att eventuellt färga om noder i grafen. I frågorna nedan, ange vilken/vilka noder, om någon, som kommer att färgas om i de olika varven av huvudloopen. Vid omfärgning, välj den färg som tillhör trädet som innehåller en nod som ligger så tidigt i alfabetet som möjligt (dvs. färgen i trädet med noden A vinner alltid).

Frågan ger avdrag för fel svar.

Svar:

I varje varv så anger jag vilka noder som färgas om. Jag använder olika bokstäver för att representera olika färger.

Varv	Vald båge (vikt och nodpar)	A	B	C	D	E	F	G	Ingen	Kommentar
1	1 B-E		R			R				Ingen färg — ny färg
2	3 A-C	G		G						Ingen färg — ny färg
3	4 D-F				B		B			Ingen färg — ny färg
4	5 C-G							G		C färgad — färga G
5	7 C-D				G		G			Färgade med olika färger, färga om med A-trädets färg
6	8 C-F								X	Färgade med samma färg, gör ingenting
7	10 F-G								X	Färgade med samma färg, gör ingenting
8	15 C-E		G			G				Färgade med olika färger, färga om med A-trädets färg

Bedömning:

- Jag har i huvudsak låtit autorättningens poäng stå kvar och gjort någon bedömning av rimligheten av poängen baserat på det fel jag gissar att ni gjort.

5.2 Prim och Kruskal

Syftet med denna fråga är att examinera förståelse kring två grafalgoritmer - Prims och Kruskals - som löser samma problem.

Låt en graf ha n noder och m bågar. Grafen har sammanhängande komponenter med vardera minst två noder. Den sammanhängade komponent som innehåller noden med etikett A innehåller 4 noder.

Frågor och svar

a) Vilket är det maximala antalet varv som huvudloopen i Kruskals algoritm kommer att köras?

- **Svar:** Kruskals algoritm måste köras ett varv för varje båge i grafen, dvs. totalt m varv.

b) Antag att vi modifierar Kruskals algoritm till att hålla reda på hur många noder i grafen som är ofärgade. Kan vi använda den informationen till att avbryta algoritmen i förtid, dvs. vi behöver inte köra maximalt antal varv i huvudloopen?

- **Svar:** Falskt. Alla noder kan ha fått färg tidigt i algoritmen, men en omfärgning kan komma så sent som i sista varvet av huvudloopen om bågen binder ihop två existerande träd (jfr förra frågan).

c) Antag att Kruskals algoritm körs på grafen. Hur många noder kommer att totalt att finnas i det/de träd som genereras av algoritmen?

- **Svar:** Kruskals algoritm kommer att bygga träd som inkluderar alla noder som ingår i alla bågar, alltså n noder totalt.

d) Antag att Prims algoritm körs på grafen med start i noden A. Hur många noder kommer att totalt att finnas i det/de träd som genereras av algoritmen?

- **Svar:** Prims algoritm bygger ett träd som innehåller alla noder i den sammanhängande komponent som inkluderar startnoden, alltså totalt 4 noder.

e) Antag en båge tas bort så att en sammanhängande komponent har bara en nod. Vad händer med då med svaren till fråga c) och d)?

- **Svar:** Om en båge tas bort från en två-nodskomponent så betyder det att komponenten delas i två komponenter som består av vardera en nod och ingen båge. Prim är opåverkad eftersom A-komponenten innehöll 4 noder. Kruskal påverkas dock eftersom vi nu har två noder UTAN bågar, så Kruskal kommer att inkludera $n - 2$ noder i sina resulterande träd.

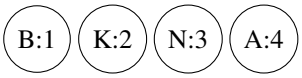
6. Huffman (15)

Syftet med denna uppgift är att examinera förståelsen för Huffman-algoritmen samt förmågan att konstruera ett Huffman-träd givet ett meddelande.

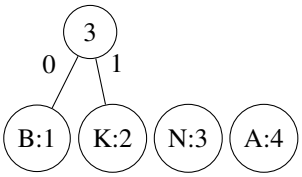
Fråga 1

Generera ett Huffman-träd för meddelandet `BANANKAKAN`. Organisera de inledande träden i viktordning, störst vikt till höger. Om det finns flera val när ni ska kombinera träd, välj ut de träd som innehåller den bokstav som kommer först i alfabetet. När ni kombinerar träden, låt det vänstra benet få etiketten 0 och det högra etiketten 1.

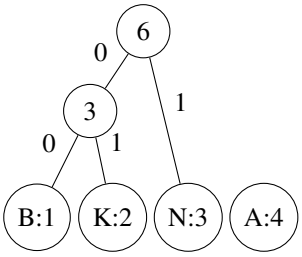
Start:



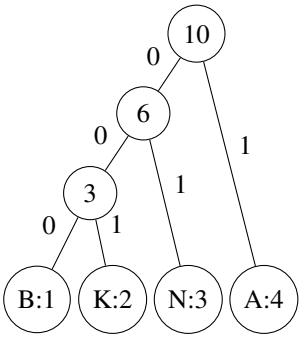
Efter första varvet:



Efter andra varvet:



Efter tredje varvet — klar!



Svar 1

Kodsträngar för respektive bokstav:

A	1
N	01
K	001
B	000

Rättning:

- Har gett rätt om kodlängden varit rätt men man inte följt instruktionerna kring vänster/höger.
- Har man fått rätt svar på A men det är uppenbart att man inte följt Huffman-algoritmen har jag tagit bort autorättningspoängen.

Fråga 2:

Huffman-algoritmen kan ibland ha flera lösningar som genererar optimal längd för det kodade meddelandet. Antag att vi vill att den längsta kodsträngen ska vara så kort som möjligt. Vilken egenskap hos Huffman-trädet vill vi då minimiera?

Svar 2:

- Höjden hos trädet. Ett exempel som ger olika maxlängder är frekvenstabellen [1,1,1,2,3,8,8] som ger maxlängd 4 vid minimal höjd och maxlängd 6 vid maximal höjd.

7. Komplexitet (8)

Studera funktionen $T(n) = 3n^2 + 2n + 2$. Givet $T(n)$, vad blir c och lägsta n_0 för $g(n) = n^2$ enligt ordo-definitionen? Om något värde behöver avrundas, avrunda uppåt till närmast högre heltal.

- **Svar:** $c = 4$ ger $n_0 = 3$ enligt nedanstående tabell.

Vad blir c och lägsta n_0 för $g(n) = n^3$ enligt ordo-definitionen? Om något värde behöver avrundas, avrunda uppåt till närmast högre heltal.

- **Svar:** $c = 1$ ger $n_0 = 4$ enligt nedanstående tabell.

n	$T(n)$	$4n^2$	n^3	Kommentar
1	7	4	1	
2	18	16	8	
3	35	36	27	$T(n) \leq 4n^2$ från $n_0 = 3$
4	58	64	64	$T(n) \leq n^3$ från $n_0 = 4$
5	87	100	125	

Rättning

- Om c är fel => inga poäng för n_0.
- Har gett halv poäng för n_0 som är korrekt men inte lägsta
- Jag *önskade* lägsta möjliga heltals-c, men då jag inte specificerat det i frågan har jag godkänt korrekta c och gett poängen för n_0 utgående från det c-värdet och resonemanget ovan.

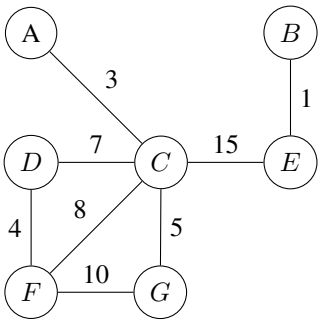
8. Operationskategorier (5)

Studera gränsytan till Riktad lista som är angiven i pseudokodsuppgiften. Vilka operationskategorier tillhör respektive funktion? Om flera val är tänkbara, välj det till höger i tabellen.

	Konstruktor	Inspektor	Modifikator	Navigator	Komparator	Destruktor
Empty	X					
Isempty		X				
Copy	X					
First				X		
Next				X		
Isend				X		
Inspect		X				
Insert			X			
Remove			X			
Kill						X

Illustrationer

```
abstract datatype DList(val)
auxiliary pos
  Empty() → DList(val)
  Isempty(l: DList(val)) → Bool

  Copy(l: DList(val)) → DList(val)

  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)

  Kill(l: DList(val)) → ()
```


A
B
3
1
D
7
C
15
E
8
4
5
F
10
G