KTH Information and
Communication Technology

ID1019
Johan Montelius

Programmering II (ID1019)
2014-01-16 09:00-12:00

Förnamn: _____

Efternamn: _____

Instruktioner

- Du får inte ha något materiel med dig förutom skrivmateriel. Mobiler etc, skall lämnas till tentamensvakten.
- Svaren skall lämnas på dessa sidor, använd det utrymme som finns under varje uppgift för att skriva ner ditt svar.
- Svar skall skrivas på svenska.
- Du skall lämna in hela denna tentamen.
- Inga ytterligare sidor skall lämnas in.

Betyg

Tentamen har ett antal uppgifter där några är lite svårare än andra. De svårare uppgifterna är markerade med en stjärna, *poäng**, och ger poäng för de högre betygen. Vi delar alltså upp tentamen in grundpoäng och högre poäng. Se först och främst till att klara de normala poängen innan du ger dig i kast med de högre poängen.
Notera att det av de 24 grundpoängen räknas bara som högst 22 och, att högre poäng inte kompenserar för avsaknad av grundpoäng. Gränserna för betyg är enligt nedan.

- E: 14 grundpoäng
- D: 18 grundpoäng
- C: 22 grundpoäng
- B: 22 grundpoäng och 8 högre poäng
- A: 22 grundpoäng och 12 högre poäng

Kursens slutbetyg kan blir högre om man ligger nära en gräns och har skrivit mycket bra rapporter.

Erhållna poäng

Skriv inte här, detta är för rättningen.

Uppgift	1	2	3	4	5	Σ
Max G/H	4/-	6/4	4/4	4/4	6/4	24/16
G/H						

Totalt antal poäng:

Betyg:

Namn:_____Personnummer:_____

1 Datastrukturer och möstermatchning

1.1 representera ett träd [2 poäng]

Antag att vi vill representera en, möjligtvis tom, mängd element där varje element är associerat till en nyckel. Beskriv en representation av ett binärt träd där varje nod innehåller en nyckel och varje löv innehåller en nyckel och ett element.

1.2 vad är Y [2 poäng]

Vad är bindningen för `Y` i följande mönstermatchningar (var för sig), i de fall där matchningen lyckas:

- `[X|Y] = [1,2,3]`
- `[X,Y] = [1,2,3]`
- `[X,Z|Y] = [1,2,3]`
- `X = {f, 42}, {G, Y} = X`
- `H = head, T = tail, Y = [H|T]`

Namn:_____Personnummer:_____

2 Funktionell programmering

2.1 XXVIII [2 poäng]

Antag att romerska tal är representerade som en sträng, till exempel "XXVIII". Vi förutsätter att alla tal är korrekt skrivna och endast använder sig av talen "X", "V" och "I". En sträng i Erlang representeras naturligtvis av en lista av ascii-värden där värden för "X", "V" och "I" kan skrivas `$X`, `$V` och `$I`; strängen "XVII" kan alltså skrivas `[$X,$V,$I, $I]`.
Skriv en funktion som tar ett tal skrivet med romerska siffror och returnerar motsvarande heltal. Vi antar att talen är skrivna i sin enkla form, talet 14 kommer alltså inte skrivas på formen "XIV" utan på formen "XIIII".

2.2 XXIV [2 poäng]

Antag nu att vi tillåter den mer kompakta formen; 14 skrivs "XIV" och 19 "XIX" (vi struntar i "VX" eftersom man då skriver "V"). Skriv en funktion som tar sådana tal och returnerar motsvarande heltal. Antag att alla romerska tal är välskrivna, de kommer alltså inte att se ut som "XIIV" eller "XVIX".

Namn:_____Personnummer:_____

2.3 operationer på mängder [2 poäng]

Din uppgift är att representera mängder av heltal och definierar operationer på dessa. Följande operationer skall definieras:

- union/2 : $\{1,3\} \cup \{3,5\} = \{1,3,5\}$
- elem/2 : $3 \in \{3,5\} = true$
- isec/2 : $\{1,3\} \cap \{3,5\} = \{3\}$

Du får inte använda dig av några inbyggda funktioner (som till exempel `++`) eller funktioner från bibliotek (som till exempel `lists:delete/2`. Beskriv först hur mängder är representerade; var noga i din beskrivning.

Namn:_____Personnummer:_____

2.4 mängddifferens [4 poäng*]

Givet representationen i uppgiften ovan, definierar mängddifferens. Du får inte använda några inbyggda funktioner eller biblioteksfunktioner. Tänk på att detta skall gälla:

$$1 \in ((\{1,2,3\} \cup \{1,5\}) \backslash \{1\}) = \text{false}$$

- diff/2 : $\{1,3,5\} \backslash \{3,5\} = \{1\}$

Namn:_____Personnummer:_____

3 Högre ordningens funktioner

3.1 funktionen map/2 [2 poäng]

Definiera funktionen `map/2` som tar en funktion och en lista och returnerar en lista bestående av resultaten av att applicera funktionen på vart och ett av elementen i listan. Om vi anropar funktionen `map(F, [a,b,c])` skall resultatet vara listan `[F(a), F(b), F(c)]`.

3.2 map, fold och filter [2 poäng]

Antag att vi har följande funktioner definierade:

- `map(F, L)`: returnerar listan av $F(E)$, där E är element i listan `L`
- `foldr(F, A, L)`: returnerar A_k där k är längden på listan `L`, A_0 = `A`, $A_i = F(E_i, A_{(i-1)})$ och E_i det *i:te* elementet i listan
- `filter(F, L)`: returnerar listan av element E_i, där E_i är element i listan `L`, för vilka $F(E_i)$ returnerar *true*

Definiera en funktion `tutti_paletti/1`, som givet en lista returnerar summan av kvadraten av de tal som är större än noll. Funktionen tar en lista av tal; uttrycket `tutti_palettiti([1,-2,3])` evalueras till exempel till 10 eftersom 1 och 3 är större än noll och summan av dess kvadrater, $(1*1+3*3)$, är 10.

Namn:_____Personnummer:_____

3.3 höger eller vänster [4 poäng*]

I Erlang, liksom i de flesta funktionella språk, finns de två inbyggda funktionerna *foldl* och *foldr* som gör "fold" på elementen i en lista antingen från vänster (*foldl* från början) eller från höger (*foldr* från slutet). Det finns för- och nackdelar med dessa två strategier och det är inte alltid klart vilken som blir mest effektiv.
I nedanstående exempel vill vi lägga ihop alla elementen i en lista av listor. Funktionen *flatten([[1,2],[3,4,5],[6,7]])* skall returnera *[1,2,3,4,5,6,7]*. Är det mest effektivt att använda *foldl* eller *foldr* i definitionen nedan (vi får vända på argumenten till `append/2` för att ordningen skall bli densamma). Argumentera varför den ena skulle vara bättre än den andra, ange även den asymptotiska tidskomplexiteten i de två fallen.

```
flatten(Lists) ->
    foldl(fun(X,Acc) -> append(Acc,X) end, [], Lists).
```

eller

```
flatten(Lists) ->
    foldr(fun(X,Acc) -> append(X,Acc) end, [], Lists).
```

Namn:_____Personnummer:_____

4 Komplexitet

4.1 sökning i lista [2 poäng]

Vad är den asymptotiska tidskomplexiteten för funktionen `member/2` nedan. Förutsätt att vi söker efter en atom i en lista av atomer.

```
member(_,[]) -> false;
member(X,[X|_]) -> true;
member(X,[_|T]) -> member(X,T).
```

4.2 lista av listor [2 poäng*]

Vad är den asymptotiska tidskomplexiteten för funktionen `member/2` om vi förutsätter att vi söker efter en given lista i en lista av listor.

Namn:_____Personnummer:_____

4.3 sökning efter värde [2 poäng]

Beskriv en datastruktur som representerar en mängd *nyckel/värde-par* och en sökfunktion `lookup/2`, som givet en nyckel hittar ett värde med asymptotisk tidskomplexitet $O(\lg(n))$.

4.4 nyckel som nyckel [2 poäng*]

Vad krävs för att den föreslagna datastrukturen skall kunna används, finns det några begränsningar på de nycklar som vi kan använda? Skulle man kunna använda vilka nycklar som helst så länge de är olika?

5 Concurrency

5.1 ett konto [2 poäng]

Definiera en process i Erlang som innehåller ett saldo och som kan ta emot följande meddelanden:

- `{deposit, Amount}`: addera *Amount* till saldot.

- `{withdraw, Amount, From}`: dekrementera saldot och skicka *ok* till processen *From*
- `{check, From}`: skicka *{saldo, Saldo}* till processen *From*.

Vi tillåter att kontot blir övertrasserat dvs, vi kan ha ett negativt saldo.

5.2 undvik att övertrassera [2 poäng]

Antag att vi vill undvika att övertrassera kontot och därför implementerar följande funktion:

```
safe_withdrawal(Acount, Amount) ->
    Account ! {check, self()},
    receive
      {saldo, Saldo} ->
           if
               Saldo >= Amount ->
                     Account ! {withdraw, Amount, self()},
                     receive
                         ok -> ok
                     end;
                true ->
                     no
```

```
            end
    end.
```

Antag att alla processer som använder konto måste använda sig av denna funktionen när de gör uttag. Hur säkra är vi på att vi inte övertrasserar kontot, vad kan hända?

5.3 foo-bar-zot [2 poäng]

Givet nedanstående definition av procedurerna `go/1`, `foo/2`, `bar/2` och `zot/2`, kommer funktionsanropet `go(5)` att returnera någonting och i så fall vad?

```
go(X) ->
   Self = self(),
   Pid = spawn(fun() -> foo(X, Self) end),
   Pid ! {sub, 3},
   Pid ! {mul, 2},
   Pid ! {add, 4},
   receive
      {done, V} ->
          V
   end.

foo(N, Go) ->
     receive
         {add, X} -> bar(N+X, Go);
         {mul, X} -> bar(N*X, Go)
     end.

bar(N, Go) ->
     receive
         {mul, X} -> zot(N*X, Go);
         {add, X} -> zot(N+X, Go);
         {sub, X} -> zot(N-X, Go)
```

```
     end.

zot(N, Go) ->
     Value = receive
               {mul, X} -> N*X*2;
               {sub, X} -> N-(X*2);
               {add, X} -> N+(X*2)
             end,
    Go ! {done, Value}.
```

5.4 fördel och nackdel [4 poäng*]

Antag att vi vill implementera en tjänst som håller två värden. Gränssnittet är en uppsättning funktion som kan läsa eller skriva dessa värden. En lösning skulle kunna se ut som följer:

```
new() ->
 A = spawn(fun() -> init(0) end),
 B = spawn(fun() -> init(0) end),
 {A, B}.

add_a({A,_}, N) ->
 A ! {add, N}.
```

```
add_b({A,B}, N) ->
 B ! {add, N}.
```

Där själva processerna är implementerade som förväntat.
Ett alternativt sättatt implementera tjänsten är enligt följande:

```
new() ->
 spawn(fun() -> init(0,0) end),

add_a(Pid, N) ->
 Pid ! {add_a, N}.
add_b(Pid, N) ->
 Pid ! {add_b, N}.
```

I detta fall är det alltså en process som tar emot meddelanden om att ändra värdet för antingen det ena eller andra värdet.
Diskutera för- och nackdelar med de två implementationerna ovan. Ge exempel på utvidgningar som är enkla att göra i det ena fallet men svårare i det andra? Finns det prestandavinster att göra?