TDP007 - Tenta

2023-03-23

Regler

- All kod efterrättas på denna tenta
- Vid toalettbesök ska vakt informeras.
- All form av kontakt mellan studenter under tentamens gång är strängt förbjuden.
- Böcker och anteckningssidor kan komma att granskas av assistent, vakt eller examinator under tentamens gång.
- Frågor om specifika uppgifter eller om tentamen i stort ska ställas via tentasystemets kommunikationsklient.
- Systemfrågor kan ställas till jourhavande genom att räcka upp handen
- Endast uppgifter inskickade före tentamenstidens slut rättas.

Hjälpmedel	En Rubybok (exempelvis the pickaxe) Ett A4-ark med egna anteckningar Tillgång till webresurser: ruby-docs, rubular och tidig version av kursboken

Information

Betygssättning vid tentamen

Tentamen består av ett antal uppgifter på varierande nivå. Uppgifter som uppfyller specifikationen samt följer god sed och konventioner krävs för maxpoäng på en uppgift. Avvikelser från ovanstående ger avdrag.

Tentan består av uppgifter (några indelade i deluppgifter) som totalt kan ge 50 poäng.

Ruby-docs

På tentan har du tillgång till referenssidorna på `https://ruby-doc.org/` (både core och std-lib finns tillgänglig) via webbläsaren.

Rubular

På tentan har du tillgång till sidan `https://rubular.com/`.

Givna filer

Eventuella givna filer finns länkade från kurssidan.

Avslutning

När du skickat in alla uppgifter och känner dig färdig kan du stänga tetaklienten och logga ut.

Uppgift 1 - Teori (13p)

Två av dessa uppgifter relaterar till andra uppgifter på tentan. Vi rekommenderar att du först löser den praktiska uppgiften och sedan svarar på motsvarande teorifråga.

Uppgift 1a - (Inte relaterat till någon praktiskt uppgift) (5p)

```
                          A       B  C       D
Available memory: ..............................
                  ..           ..  . .....      .
                  ..............................

Length of data to store: 6 3 4 2 3
```

I en dators lagringsutrymme kan det finnas sammanhängande platser av olika längd, ett problem som man kan behöva ta hänsyn till är hur man bäst ska skriva data till dessa platser så man inte slösar med utrymmet. Ovan ser du ett exempel på 4 delar av en dators lagringsutrymme (A, B, C och D) och längden av ett antal data som ska sparas. A har i exemplet 10 tillgängliga platser. I del A skulle man alltså kunna lagra en data av längden 6 och en data av längen 4 innan utrymmet var fullt.

I den givna filen `amb_test.rb` hittar du den problemlösare vi arbetat med under laborationsserien. Förklara hur man med hjälp av denna problemlösaren kunde representera tillgängligt lagringsutrymme, data som ska lagras och sedan få ut alla möjliga kombinationer av lagring. Din beskrivning bör vara skriven på ett sätt att en annan programmerare (som är någorlunda bekant med problemlösaren och ruby) kan implementera denna lösning.

Uppgift 1b - (relaterat till Uppgift 2 om Sax-parsning) (4p)

Förklara hur du kunde få ut motsvarande information med ett regexp. Du behöver inte (och bör inte med hänsyn till tid) tillhandahålla en implementation av detta. Men förklara på ett tydligt sätt vilka steg som behöver tas och vilka reguljära uttryck som behövs etc. Jag ska i princip kunna utgå från denna beskrivning för att relativt friktionsfritt kunna lösa uppgiften med regexp istället för en xml-parser.

Uppgift 1c - (relaterat till Uppgift 4 om rdparse) (4p)

Förklara hur du skulle kunna lägga till tilldelning och uppslagning av variabler i detta språk. Övertyga mig om du kunde implementera detta på ett robust och fungerande sätt. Det är ok att utöver beskrivningen tillhandahålla kodexempel men inte nödvändigt. Vi kan anta att allt kommer finnas i ett globalt scope.

Uppgift 2 - Sax-parsning av XML (9p)

I den givna filen `bgg.xml` finns xml-element som representerar en spelares samling av brädspel. Pontus letar nu efter ett riktigt bra spel att spela med sina sex kompisar, problemet är att han har så många spel i sin samling att det är svårt att välja. Ditt jobb är att med SAX-parsning plocka ut namnet, betyget och maximala spelarantalet på alla spel som uppfyller följande kriterier:

- Minst 7 spelare som maximalt spelarantal
- Minst 9 i betyget som Pontus satt på spelet

Spelen du filtrerar ut ska sparas i form av en `Array` av `Hash`ar, där varje `Hash` innehåller namn, max spelare och betyg för ett spel. Du hittar ett exempel på hur man kan arbeta med en sax_parser i `sax_example.rb`.

Körexempel:

```
[{:name=>"One Night Ultimate Werewolf", :maxplayers=>10, "rating"=>9},
{:name=>"Race Royale", "rating"=>10, :maxplayers=>10},
{:name=>"Team Kill", :maxplayers=>36, "rating"=>10},
{:name=>"Tempel des Schreckens", :maxplayers=>10, "rating"=>9}]
```

Krav: Kriterierna för spelarantal och betyg ska gå att sätta genom konstruktorn för implementationen av din strömparser.
Krav: Ett betyg som inte är satt (N/A) räknas som betyg 0.
Krav: Du kan anta att de intressanta elementen existerar men inte ordningen de förekommer i.

Tips: Max spelarantal hittar du i elementet `stats`
Tips: Betyget Pontus satt på ett spel hittar du i elementet `rating`

Uppgift 3 - Dom-parsning av XML(8p)

I den givna filen `bgg.xml` finns ett utdrag av alla spel som finns i en användares samling från boardgamegeek. Titta igenom den filen och sätt dig in i strukturen. Filen är hämtad genom deras xml-api.

Den här användaren har ett problem som du måste lösa. Användaren pratade nyligen med en kompis om ett spel som de spelat tillsammans men de kan inte komma ihåg vilket spel det var. Användaren vet följande saker om spelet:

- Spelet släpptes mellan 1997 och 1999 (inklusive på båda årtalen)
- Användaren har tidigare ägt spelet

Ditt jobb är att skriva en funktion `previously_owned_in_span` som tar 3 parametrar, ett xmldocument, ett minimum år och ett maximum år. Funktionen ska sedan returnera en `Array` som innehåller `Hash`, varje `Hash` representerar ett spel. Där varje `Hash` innehåller namnet på ett spel och året detta släpptes. Den `Array` som returneras ska innehålla alla spel som uppfyller kriterierna. Lägg till ett anrop till funktionen och en utskrift av returvärdet.

Körexempel:

```
{:name=>"For Sale", :year=>1997}
{:name=>"Metro", :year=>1997}
{:name=>"Mysteriet På Greveholm", :year=>1999}
{:name=>"Paths of Glory", :year=>1999}
{:name=>"ThinkBlot", :year=>1997}
```

Tips: Publiceringsåret finns i elementet `yearpublished`
Tips: Om spelet tidigare ägts finns i elementet `status`

Krav: Problemet ska lösas med Dom-parsning

Uppgift 4 - Parser(11p)

I filen `rdparse.rb` finns den parser som vi arbetat med i kursen, ditt jobb i denna uppgift är att utöka/ändra denna parser så att det givna språket nedan fungerar enligt beskrivningen. Detta språk är ett mycket enkelt programspråk där användaren kan mata in ett godtyckligt antal heltal, operatorer, tecknet `p` eller tecknet `d`, dessa separeras med blanksteg(mellanslag). Dessa saker tolkas sedan en och en från vänster till höger. Listan nedan beskriver vad som ska hända när varje symbol hanteras. I princip så bygger vi här en stackmaskin. Denna stackmaskin består av en behållare som representerar stacken. I stacken läggs ett antal heltal på hög och sedan finns det matematiska operationer som verkar på talen i stacken. Du vill lämpligen ha en `Array` som representerar stacken av heltal.

- **HELTAL** Pusha heltalet till stacken
- **p** Poppa 1 heltal från stacken, skriv ut talet med `puts`, returnera sedan heltalet. Returen spelar ingen roll, men rdparse hanterar returvärden av typen `nil` på ett speciellt sätt, så vi behöver returnera något annat för att komma runt det vanliga returnvärdet av `puts`.
- **d** Poppa 1 heltal från stacken. Pusha det heltalet 2 gånger till stacken.
- **+** Poppa 2 heltal från stacken. Pusha ett nytt heltal som är summan av de två talen till stacken.
- **-** Poppa 2 heltal från stacken. Pusha ett nytt heltal som är differensen av de två talen till stacken. Tänk på ordningen eller absolutbelopp (se körexemplet).
- ***** Poppa 2 heltal från stacken. Pusha ett nytt heltal som är produkten av de två talen till stacken.
- **/** Poppa 2 heltal från stacken. Pusha ett nytt heltal som är kvoten av de två talen till stacken. Tänk på ordningen (se körexemplet), talet som ligger överst på stacken ska vara täljaren och det andra talet ska vara nämnaren.

Följande strängar bör ge följande output vid körning

```
parser.parse_string("1 d + p") => 2
parser.parse_string("1 2 + p") => 3
parser.parse_string("2 1 - p") => 1
parser.parse_string("1 2 + 3 * p") => 9
parser.parse_string("2 3 4 + * p") => 14
parser.parse_string("1 2 + 3 * 3 / p") => 3
```

Tips: Tänk på att parsern i detta fall blir väldigt lättviktig eftersom varje token kan tolkas helt separat
Tips: Försök inte att sätta ihop regler i stil med `e := e + t` för att hantera addition
Tips: Det är i uppgiften odefinierat vad som händer om det exempelvis inte skulle finnas tillräckligt med tal på stacken för att utföra en operation. Det är helt ok.

Uppgift 5 - DSL (9p)

I den givna filen `dsl.rb` finns en fil som specificerar ett contraintnätverk genom ett dsl. Ditt jobb är att implementera detta domänspecifika språk med hjälp av `method_missing` i ruby. I den givna filen `dsl_reader.rb` finns lite given kod som du kan utgå ifrån. I filen `dsl.rb` finns det två olika typer av rader som din implementation ska klara av.

Den första och fjärde raden sätter upp nya binära constraints. De första 2 bokstäverna är connectors som är indatan och den tredje bokstaven är connectorn som är utdatan, sist kommer en sträng med operatorn. När en ny constraint sätts upp ska denna lagras i en hash där ut-connectorn associeras med in-connectors och operatorn (se körexemplet).

De andra raderna sätter en connector till ett heltal. När en sådan rad läses in ska connectorn associeras med heltalet.

Båda dessa `Hash`ar ska gå att plocka ut efter inläsningen är klar, se körexempelet.

Körexempel:

```
constraint_network = ConstraintParser.parse("dsl.rb")
puts constraint_network.connectors()
puts constraint_network.constraints()

{:x=>5, :y=>4, :r=>6}
{
  :z=>{:in_a=>:x, :in_b=>:y, :operator=>:+},
  :s=>{:in_a=>:z, :in_b=>:r, :operator=>:*}
}
```

- Formateringen av utskrift spelar ingen roll, men datatyperna och hur de associeras med varandra spelar roll
- Du behöver inte hantera annat än binära connectors i denna uppgift
- Du kan konvertera en sträng till en symbol med `#String.to_sym`
- Du kan kontrollera datatypen på ett object med `#Object.is_a? Integer` (för att kontrollera om ett object är av typen Integer)
- Din implementation ska fungera för godtyckliga operatorer (alla strängar anses vara operatorer i denna uppgift), godtyckliga namn på connectors (text som följer reglerna för ett funktionsnamn anses vara en connector i denna uppgift).