TDP007 - Tenta

2024-03-21

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 och Regex (10p)

Uppgift 1a (5p)

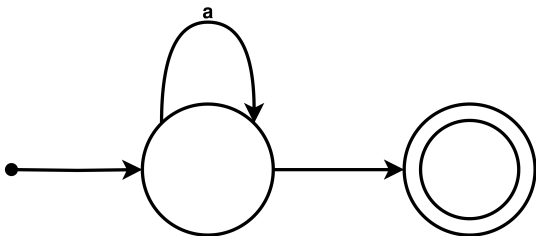
I figurspelet Warhammer bygger varje spelare sin egen samling av figurer och sedan när man spelar så använder varje spelare en delmängd av sin samling för att utkämpa strider. För att saker och ting ska vara rättvist kostar olika figurer olika många poäng och tillhör tre olika kategorier (hjältar, grund och special). Man kommer överens med sin motståndare om hur många poäng totalt som man ska använda under striden och det finns också en begränsning av hur stor andel av poängen som får komma från olika kategorier. Exempelvis kan jag och en motståndare kommit överens om följande:

- Max 2000 poäng
- Max 50 procent får vara av typen hjältar
- Minst 25 procent måste vara av typen grund
- Max 50 procent får vara av typen special

Förklara hur du med hjälp av denna problemlösare `amb_test.rb` kan ge förslag på godkända delmängder av en spelares samling. Du behöver förklara vad för databehållare du behöver sätta upp, hur kombinationer som ska testas genereras och hur kontrollen av en godkänd kombination utförs. Du ska inte skapa en implementation i denna. Förklara med pseudokod och vanlig text som enkelt går att följa för att göra implementationen.

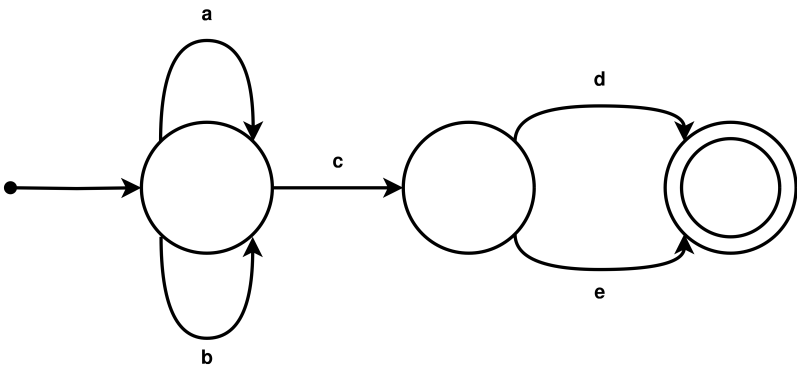
Uppgift 1b (5p)

Implementera ett regex som motsvarar följande automat:



Uppgift 1c (5p)

Implementera ett regex som motsvarar följande automat:



Uppgift 2 - SAX-Parsning av XML (XXXXXXXXXXXXp)

I den givna filen `dnd.xml` finns xml-element som representerar monster och föremål man kan använda när man spelar Dungeons and Dragons. Pontus som DM får många frågor ifrån sina spelare som undrar över föremåls vikter men problemet är att det existerar en stor mängd föremål. För att underlätta detta problemet ska du nu skriva ett program som med hjälp av en SAX-parser hämtar ut alla föremål väger mindre än `0.03` ifrån den givna filen `dnd.xml`.

De föremål du filtrerar ut ska sparas i form av en Array av Hashar där varje Hash innehåller föremålets namn, typ och vikt. Du hittar ett exempel på hur man kan arbeta med en sax_parser i `sax_example.rb`.

Körexempel:

```
[{:name=>"Copper (cp)", :type=>"$", :weight=>"0.02"},
  {:name=>"Electrum (ep)", :type=>"$", :weight=>"0.02"},
  {:name=>"Gold (gp)", :type=>"$", :weight=>"0.02"},
  {:name=>"Platinum (pp)", :type=>"$", :weight=>"0.02"},
  {:name=>"Silver (sp)", :type=>"$", :weight=>"0.02"},
  {:name=>"Blowgun Needle of Slaying", :type=>"A", :weight=>"0.02"},
  {:name=>"Blowgun Needles", :type=>"A", :weight=>"0.02"},
  {:name=>"Blowgun Needles +1", :type=>"A", :weight=>"0.02"},
  {:name=>"Blowgun Needles +2", :type=>"A", :weight=>"0.02"},
  {:name=>"Blowgun Needles +3", :type=>"A", :weight=>"0.02"},
  {:name=>"Walloping Blowgun Needle", :type=>"A", :weight=>"0.02"},
  {:name=>"Ball Bearings", :type=>"G", :weight=>"0.002"},
  {:name=>"Sling +1", :type=>"R", :weight=>"0"},
  {:name=>"Sling +2", :type=>"R", :weight=>"0"},
  {:name=>"Sling +3", :type=>"R", :weight=>"0"},
  {:name=>"Sling of Warning", :type=>"R", :weight=>"0"},
  {:name=>"Vicious Sling", :type=>"R", :weight=>"0"}]
```

Tips: XML elementet weight finns även bland monstren i filen så man måste se till att man är på rätt ställe och hämtar ut informationen.

Krav: Kriteriet för vikten ska kunna sättas genom konstruktorn i parserklassen.

Krav: Programmet ska skriva ut resultatet, det behöver inte vara exakt samma representation men informationen ska stämma överens med körexemplet.

Uppgift 3 - DOM-Parsning av XML(XXXXXXXXXXXp)

I den givna filen `dnd.xml` finns xml-element som representerar monster och föremål man kan använda när man spelar Dungeons and Dragons. Filen har de officiella föremålen i spelet och har även uttökats med några hemmaknåpade föremål. De har frånskiljts med ett attribut i filen. Din uppgift är nu att skriva ett program som med hjälp av en DOM-parser hämtar ut alla föremål som uppfyller följande kriterier. Föremålet ska ha attributet `tier` satt till `homebrew` och sällsyntheten ska ha texten `common` eller `rare`.

Körexempel:

```
[{:name=>"Arrows",
  :rarity=>"common",
  :text=>
    "Ammunition: You can use a weapon that has the ammunition property to
    make a ranged attack only if you have ammunition to fire from the
    weapon. Each time you attack with the weapon, you expend one piece of
    ammunition. Drawing the ammunition from a quiver, case, or other
    container is part of the attack. At the end of the battle, you can
    recover half your expended ammunition by taking a minute to search the
    battlefield. Source: Player's Handbook p. 150"},
 {:name=>"Arrows +2",
  :rarity=>"rare",
  :text=>
   "You have a +2 bonus to attack and damage rolls made with this piece of
   magic ammunition. Once it hits a target, the ammunition is no longer
   magical. Source: Dungeon Master's Guide p. 150"}]
```

Tips: Sällsyntheten på ett förmål anges som en sträng i xml elementet detail i det givna `dnd.xml` dokumentet.

Tips: Det kan förekomma flera text element i XML filen för ett föremål. Din lösning får endast spara en sträng per föremål.

Krav: De föremål du filtrerar ut i funktionen ska returneras i form av en Array av Hashar där varje Hash innehåller föremålets namn, sällsynthet och text.

Krav: Programmet ska skriva ut resultatet, det behöver inte vara exakt samma representationen men informationen ska stämma.

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. Det här är ett domänspecifikt språk för att skapa listor av enheter i figurspelet Warhammer the Old World, men det behöver vi inte veta mycket om. Det viktiga att veta är att det finns olika enheter och att man sätter ihop en lista av dessa enheter innan man spelar spelet. I den givna filen `warhammerlist.txt` finns ett program skrivet i språket. Det finns 3 typer av rader i denna fil som parsern på ett rimligt generellt sätt ska kunna hantera:

- De första 6 raderna kod är tilldelning av enhetsbeskrivningar till en variabel. Formatet av denna typ av rad är: variabelnamn, likhetstecken, enhetens namn inom citattecken, antalet modeller i enheten, antalet poäng enheten kostar (uttryckt som helttal följt av bokstäverna `pts`) och tillsist enhetens kategori. Resultatet av att köra denna rad i parsern är att en variabel som innehåller denna information ska representeras med lämplig databehållare.
- De följande 10 raderna med kod är insättning av den data som en variabel innehåller i spelarens lista. Listan innehåller alltså 10 enheter.
- Den sista raden i programmet skriver ut listan som skapats. För full poäng ska listan av enheter skrivas ut en gång, formateringen behöver inte se ut precis som körexemplet, men informationen där ska framgå.

Tips: Grammatiken för detta program kan bli mycket enkel om delar av arbetet görs i lexern.
Tips: Tänk på att vi i lexern kan skapa instanser av egna klasser och använda dessa klassers typ för att styra grammatiken.

Uppgift 5 - dsl (10p)

I den givna filen `dsl.rb` finns ett givet program skrivet i ett domänspecifikt språk, programmet är gjort för att lättare skapa armélistor i figurspelet Warhammer. I den givna filen `dsl_reader.rb` finns en påbörjad dsl-läsare med method missing. Ditt jobb är att implementera en läsare för det givna programmet.

Det givna programmet innehåller i 2 typer av rader. En typ rad associerar data med ett namn, detta är de första 4 raderna i programmet i `dsl.rb`. Först skrivs namnet på variabeln, efter det följer 3 komma-separerade data: beskrivning, enhetstyp och poängkostnad. Denna del av problemet ska lösas med method missing, det skall alltså gå att ha godtyckliga variabelnamn och godtyckliga enhetstyper (dessa följer dock Rubys regler för vad ett variabelnamn är). Vid deklaration skall de tre datapunkterna sparas undan och associeras med variabelnamnet.

Den andra typen av rad är sådana som inleds med `add`. Dessa rader lägger till den datan som finns associerad med det som står efter add på raden i en lämplig databehållare. `add` kan läggas som en separat medlemsfunktion i läsaren, men namnet på datan som ska läggas till behöver fortfarande hanteras av method missing på lämpligt sätt.

Lägg till en lämplig print-funktion i din dsl-läsare och tillhörande anrop i huvudprogrammet så programmet skriver ut resultatet med lämpligt format (formateringen behöver inte följa körexemplet).

Körexempel:

```
[["1 High Elf prince on mount", "lord", 515],
 ["20 High Elf Lothern Seaguard", "core", 200],
 ["20 High Elf Lothern Seaguard", "core", 200],
 ["20 High Elf Lothern Seaguard", "core", 200],
 ["20 High Elf Phoenix Guard", "special", 300],
 ["20 High Elf Phoenix Guard", "special", 300],
 ["1 Frost Phoenix", "rare", 250]]
```

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 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).