Laboration-X i Java
TDDE22 och 725G97
Datastrukturer och algoritmer

Magnus Nielsen

21 september 2018

Förord

Denna laboration skiljer sig något från de andra laborationerna. Dels ska den inte köras igenom *Kattis*, och dels är inlämningen något annorlunda. Det räcker med att skicka in MinPQ.java, samt ett reflektionsdokument på valfritt (rimligt) filformat (lämpliga exempel: .txt, .md, .odt). Detta skickas sedan manuellt till er laborationsassistent för rättning.

Programkod

Givna filer kommer ni åt genom att köra (copy paste till terminalen):

```
cd ~/TDDC91 && mkdir LabX && cd LabX && mkdir src && cp ~TDDE22/labs/labx/* ./src
```

Därefter kan ni i Ecliple köra new -> java project och döp projektet till LabX. Alla filerna bör därefter dyka upp Eclipse om ni inte föredrar att arbeta i någon annan editor i denna laboration.

Observera: Det är inte meningen att man ska skriva om kodskeletten eller skriva helt egen kod; utan man ska utgå från den existerande koden och komplettera den med de delar som saknas, och som beskrivs i lydelsen. Assistenterna har inte möjlighet att lägga ned den tid som krävs för att sätta sig in i och rätta helt egna lösningar på uppgifterna.

Redovisning

Som beskrivs ovan behöver denna laboration ej godkännas genom *Kattis* utan vid slutfört arbete ska den visas upp för assistent. Assistenterna kommer i vanlig ordning att ha frågor som ska besvaras. Efter godkänd redovisning ska MinPQ.java samt ert reflektionsdokument manuellt skickas via email till er respektive labbassistent (den ni valt i WebReg). Eventuella kompletteringar sköts sedan via email.

Deadline

Deadline för alla laborationsuppgifterna är **första tentamenstillfället** i kursen.

Lycka till!
Magnus Nielsen

Regler för examinering av datorlaborationer vid IDA

Datorlaborationer görs i grupp eller individuellt, enligt de instruktioner som ges för en kurs. Examinationen är dock alltid individuell.
Det är inte tillåtet att lämna in lösningar som har kopierats från andra studenter, eller från annat håll, även om modifieringar har gjorts. Om otillåten kopiering eller annan form av fusk misstänks, är läraren skyldig att göra en anmälan till universitetets disciplinnämnd.
Du ska kunna redogöra för detaljer i koden för ett program. Det kan också tänkas att du får förklara varför du har valt en viss lösning. Detta gäller alla i en grupp.
Om du förutser att du inte hinner redovisa i tid, ska du kontakta din lärare. Då kan du få stöd och hjälp och eventuellt kan tidpunkten för redovisningen senareläggas. Det är alltid bättre att diskutera problem än att, t.ex., fuska.
Om du inte följer universitetets och en kurs examinationsregler, utan försöker fuska, t.ex. plagiera eller använda otillåtna hjälpmedel, kan detta resultera i en anmälan till universitetets disciplinnämnd. Konsekvenserna av ett beslut om fusk kan bli varning eller avstängning från studierna.

Policy för redovisning av datorlaborationer vid IDA

För alla IDA-kurser som har datorlaborationer gäller generellt att det finns en bestämd sista tidpunkt, deadline, för inlämning av laborationer. Denna deadline kan vara under kursens gång eller vid dess slut. Om redovisning inte sker i tid måste, den eventuellt nya, laborationsserien göras om nästa gång kursen ges.
Om en kurs avviker från denna policy, ska information om detta ges på kursens webbsidor.

Lab X: Simulering

Mål: *När du gjort den här laborationen ska du känna till hur man kan simulera rörelserna hos N partiklar som kolliderar elastiskt med hjälp av händelsedriven simulering. Specifikt ska du behärska användandet av prioritetsköer vid sådan simulering. Den här typen av simulering är mycket vanligt förekommande när man försöker förstå och förutsäga egenskaper hos fysikaliska system på molekyl- och partikelnivå. Detta inkluderar rörelserna hos gasmolekyler, dynamiken i kemiska reaktioner, diffusion på atomnivå, stabiliteten hos ringarna runt Saturnus, och fasövergångarna i olika grundämnen. Samma tekniker används i andra tillämpningsområden, som datorgrafik, datorspel och robotik, för att simulera olika partikelsystem.*

Den här laborationen är delvis baserad på en C++-laboration vi har använt i andra DALG-kurser, och den är i sin tur en anpassad version av en mycket mer omfattande labbserie i Java som ges vid Princeton University av Robert Sedgewick.

Förberedelser: *Läs på om prioritetsköer i OpenDSA.*

Uppgift: Simulering av partikelkollisioner i Java

Vår implementation innefattar följande klasser: `MinPQ`, `Particle`, `Event`, `StdDraw` och `CollisionSystem`. `Event`-klassen representerar en kollisionshändelse. Det finns fyra sorters händelser: kollision med vertikal vägg, kollision med horisontell vägg, kollision mellan två partiklar och en händelse som betyder att det är dags att rita om bilden av partikelsystemet. Vi har valt följande enkla (men inte särskilt eleganta) lösning: För att kunna avgöra om en händelse är giltig sparar `Event` antalet kollisioner relevanta partiklar varit inblandade i när händelsen skapades. Varje Event har två huvudpartiklar (de två som förutses kollidera). När händelsen bearbetas svarar den mot en fysisk kollision om antal kollisioner för vardera partikel är samma som när händelsen skapades, dvs de inblandade partiklarna har inte varit inblandade i några andra kollisioner från det att händelsen förutsågs tills det den bearbetas.

Katalogen `/home/TDDE22/labs/labx` innehåller, förutom klasserna ovan, filen `Simulation.java` som i sin tur innehåller ett program som driver simuleringen. Utöver källkodsfilerna innehåller katalogen även några indatafiler: `billiards10.txt` vilket är en väldigt liten simulering, `diffusion.txt` som är lite större och mer intressant, och slutligen `brownian.txt` vilket är den stora simuleringen. Frånsett testprogrammet (TestMinPQ.java) är det enklast att manuellt kompilera och köra programmet i terminal. Om du navigerar till src mappen kan du kompilera genom att köra:

```
>javac Simulator.java
```

Du kan nu ge kommandot:

```
>java Simulation N
```

för att simulera N partiklar med slumpmässiga egenskaper i 10000 sekunder. Kommandot

```
>java Simulation < file
```

läser in partikeldata från filen *`file`* och kör simuleringen. Det är nu dags att du experimenterar lite med programmet och sätter dig in i källkoden för att få en översiktlig bild av hur simuleringen går till. Kör definitivt minst en gång med `brownian.txt` som indatafil enligt ovan innan ni börjar hacka kod.

Uppgift: *I filen `MinPQ.java` finns den prioritetskö simuleringsprogrammet använder sig av. Din uppgift är att implementera* heap-insättning *i prioritetskön. I nuläget görs lat insättning, vilket innebär att vi bara stoppar in event i kön på första bästa lediga plats, vilket i sin tur innebär att vi måste testa och återställa heap-egenskapen för hela heapen varje gång vi tar ut min-elementet. För denna insättning används metoden toss i `MinPQ`.*

Funktionen `insert` *kastar bara ett undantag, men ska istället göra insättning enligt principen för heapar. Metoder för att återställa heap-egenskapen finns givna, det är dock* ***inte*** *ok att anropa fixHeap() i insert då detta är extremt ineffektivt (och inte speciellt utmanande). Om du anser dig behöva göra andra ändringar i* `MinPQ.java` *för att implementationen ska bli korrekt bör du se till att det går att ändra tillbaka och köra med pq.toss i CollisionSystem (förklaras nedan). I labbkatalogen finns också filen* `TestMinPQ.java` *som innehåller ett program som testar ifall prioritetskön fungerar som den ska. Anropet* `javac TestMinPQ.java` *kompilerar testprogrammet och du kan sedan köra det med* `java TestMinPQ` *alternativt köra programmet i Eclipse.*

När du fått prioritetskön att fungera med heap-insättning är det dags att ersätta alla anrop till `pq.toss` *med* `pq.insert` *i* `CollisionSystem.java` *och kontrollera att simuleringsprogrammet fortfarande fungerar som det ska. Observera att* `toss` *och övriga metoder i* `MinPQ` *fortfarande måste fungera korrekt trots att heap-insättning kan ha gjorts.*

Datafilerna för partikeldata har följande format: Första raden innehåller antalet partiklar N. Resterande N rader innehåller vardera 6 reella tal (position, hastighet, massa och radie) följt av tre heltal (rött, grönt och blått färgvärde). Alla positionskoordinater ska ligga mellan 0 och 1 och färgvärdena mellan 0 och 255. Ingen partikel får heller ha något överlapp med någon annan partikel eller väggarna.

```
N
rx ry vx vy mass radius r g b
rx ry vx vy mass radius r g b
rx ry vx vy mass radius r g b
rx ry vx vy mass radius r g b
```

Uppgift: *Skriv en kort reflektion över de förändringar du har gjort. Vilken version är bättre? Varför? Finns det fall då den "sämre" implementationen är bättre än den du valde ut?*
Frivilligt: Skapa minst en ny partikeldatafil som simulerar någon intressant situation. Om fantasin tryter ger vi några förslag nedan.

Förslag på situationer att simulera:

- En partikel i rörelse.
- Två partiklar i rörelse rakt emot varandra.
- Två partiklar, en i vila, som kolliderar i vinkel.
- N partiklar i ett rutnät med slumpmässiga initiala riktningar (men samma fart), så att den totala kinetiska energin motsvarar en fix temperatur T och totalt rörelsemängdsmoment = 0.
- Diffusion: N väldigt små partiklar av samma storlek nära mitten av behållaren med slumpmässiga hastigheter.
- Diffusion av partiklar från en fjärdedel av behållaren (jämför `diffusion.txt`).
- N stora partiklar, så att det inte finns så mycket rörelseutrymme.
- 10 partiklar i en rad som inte kolliderar med 9 partiklar i en kolumn.
- 9 partiklar i rad som kolliderar med 9 partiklar i en kolumn.
- Andra biljardsituationer än den i `billiards.txt`.
- En liten partikel trängs mellan två stora partiklar.
- Newtons pendel.

Redovisning: *Följ instruktionen för inlämning av labbar på sid. 2 i detta kompendium. Du behöver bara redogöra för dina ändringar av* `MinPQ.java`*, inte de ändringar du gjort i* `CollisionSystem.java`*. Skicka också med ditt reflektionsdokument till din assistent.*

Teoretisk / vetenskaplig bakgrund

Frivillig läsning. Det är inget krav att man förstår den vetenskapliga bakgrunden eller formlerna som används för att göra beräkningarna. Detta avsnitt kan vara lämpligt att skumma igenom för kontext, eller studera för den nyfikne, men det är inget krav att man har läst det för att klara uppgiften.

Stela sfär-modellen

Stela sfär-modellen är en idealiserad modell av rörelsen hos atomer och molekyler i en behållare. Vi kommer att fokusera på den tvådimensionella versionen, kallad *stela skiv-modellen*. De väsentliga egenskaperna hos denna modell listas nedan.

- N partiklar i rörelse, inneslutna i en låda med sidor av enhetslängd.
- Partikel i har känd position (rx_i, ry_i), hastighet (vx_i, vy_i), massa m_i och radie σ_i.
- Partiklar interagerar genom elastiska kollisioner med varandra och den reflekterande gränsytan.
- Inga andra krafter förekommer. Detta betyder att partiklar rör sig i räta linjer med konstant fart mellan kollisioner.

Den här enkla modellen har en central roll i *statistisk mekanik*, ett vetenskapligt område där man försöker koppla samman makroskopiska fenomen (som temperatur, tryck eller diffusionskonstant) med rörelser och dynamik på atom- och molekylnivå. Maxwell och Boltzmann använde modellen för att härleda ett samband mellan temperatur och fördelningen av olika farter hos interagerande molekyler; Einsten använde den för att förklara den Brownska rörelsen hos pollenkorn i vatten.

Simulering

Det finns två naturliga sätt att försöka simulera partikelsystem.

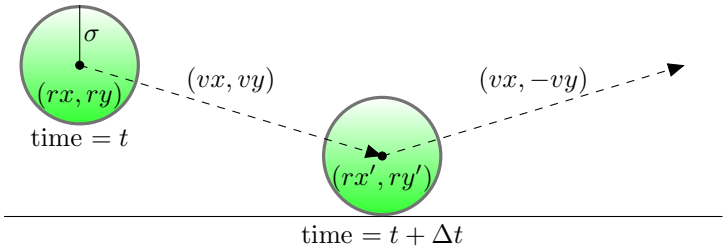
- *Tidsdriven simulering.* Diskretisera tiden i kvanta av storlek dt. Uppdatera varje partikels position efter dt tidssteg och kolla efter överlapp. Om det finns två överlappande partiklar, backa klockan till tidpunkten för kollisionen och förtsätt simuleringen. Det här sättet att simulera innebär att vi måste göra N^2 överlappskontroller per tidskvantum och att, om dt är för stort, riskerar vi att missa kollisioner eftersom kolliderande partiklar inte överlappar när vi tittar efter. För att få tillräcklig noggrannhet måste vi alltså välja dt väldigt litet, vilket saktar ner simuleringen.
- *Händelsedriven simulering.* I den här formen av simulering koncentrerar vi oss bara på de tidpunkter vid vilka någon intressant händelse äger rum. I stela skiv-modellen rör sig alla partiklar i räta linjer, med konstant fart, mellan kollisioner. Vår utmaning blir alltså att bestämma en ordnad sekvens av partikelkollisioner. Detta gör vi genom att upprätthålla en *prioritetskö* med framtida händelser, ordnade efter tid. Vid varje given tidpunkt innehåller prioritetskön alla framtida kollisioner som skulle äga rum, givet att varje partikel fortsätter längs en rät linje för all framtid. Allteftersom partiklar kolliderar och byter riktning blir vissa framtida händelser i prioritetskön ”ogiltiga” och svarar inte längre mot fysiska kollisioner. Vi kommer att använda en lat strategi där vi lämnar sådana händelser kvar i prioritetskön för att identifiera dem och bortse från dem när de sedan plockas ut ur kön. Huvudloopen i simuleringen fungerar enligt följande:
  - Ta bort den närmast förestående händelsen, dvs den med lägst prioritet t.
  - Om händelsen motsvarar en ogiltig kollision, kasta bort den. Händelsen är ogiltig om en av partiklarna har deltagit i en kollision sedan händelsen sattes in i prioritetskön.

- Om händelsen svarar mot en fysisk kollision mellan partikel i och partikel j:
  - Flytta fram alla partiklar till tid t längs rätlinjiga rörelsebanor.
  - Uppdatera hastigheterna för de två kolliderande partiklarna i och j enligt lagarna för elastiska kollisioner.
  - Bestäm alla framtida händelser som skulle äga rum där antingen i eller j är inblandade, under antagandet att alla partiklar rör sig längs räta linjer från tid t och framåt. Sätt in dessa händelser i prioritetskön.
- Om händelsen svarar mot en fysisk kollision mellan partikel i och en vägg, gör motsvarande saker som ovan för partikel i.

Det här händelsedrivna sättet att simulera blir mer robust, exakt och effektivt än det tidsdrivna.

Att förutsäga kollisioner

Hur kan vi förutsäga framtida kollisioner? Partikelhastigheterna innehåller faktiskt all information vi behöver. Antag att en partikel har position (rx, ry), hastighet (vx, vy) och radie σ vid tid t och att vi vill avgöra om och när partikeln kolliderar med en vertikal eller horisontell vägg.



Eftersom alla koordinater är mellan 0 och 1 kommer en partikel i kontakt med en horisontell vägg vid tid $t + \Delta t$ om $ry + \Delta t \cdot vy$ är lika med antingen σ eller $(1 - \sigma)$. Om vi löser ut Δt får vi:

$$\Delta t = \begin{cases} (1 - \sigma - ry)/vy & vy > 0, \\ (\sigma - ry)/vy & vy < 0, \\ \infty & vy = 0. \end{cases}$$

Alltså kan vi sätta in en händelse i prioritetskön med prioritet $t + \Delta t$ (och information som beskriver kollisionen mellan partikel och vägg). En liknande ekvation förutsäger tidpunkten för kollision med en vertikal vägg. Beräkningarna för två partiklar som kolliderar är också snarlika, men mer komplicerade. Notera att allt som oftast leder beräkningen till att en kollision *inte* kommer att ske. I så fall behöver vi inte sätt in något i prioritetskön. Det kan också hända att den förutsedda kollisionen ligger så långt fram i tiden att den inte är intressant. För att slippa lagra sådan information i kön håller vi reda på en parameter `limit` som ger en bortre gräns för vilka händelser vi är intresserade av.

Lösning av kollisioner

När en kollision väl äger rum behöver vi lösa upp den genom att applicera de formler från fysik som bestämmer beteendet hos en partikel efter en elastisk kollision med en reflekterande gränsyta eller med en annan partikel. I vårt exempel ovan, då partikeln träffar en horisontell vägg, ändras hastigheten från (vx, vy) till $(vx, -vy)$ vid tidpunkten för kollisionen. Sambanden för kollision mot vertikal vägg och kollision partikel mot partikel är snarlika.