Laboration 3

TDA416, lp3 2016

Uppgiften går ut på att implementera en mängd med hjälp av två olika sorterade samlingar; sorterad länkad lista och splay-träd. Ni ska också skriva ett program som automatiskt testar era implementeringar. Slutligen ska ni utvärdera komplexiteten hos era implementeringar. För detta tillhandahålls ett litet testprogram. På kursens websida finns en länk till givna filer.

Sorterad länkad lista

Skriv en implementering för interfacet `SimpleSet.java` (given fil) med en sorterad länkad lista med minst element först. Detta interface innehåller en delmängd av javas `Set` och definierar de metoder som ska implementeras. Dessa metoders semantik är densamma som för `Set` så ni kan kolla java API:s dokumentation angående detta. Notera skillnaden jämfört med `Collection`; att två lika objekt aldrig ska finnas samtidigt i mängden. Ni ska implementera en konstruktor som tar noll argument och skapar en tom mängd. Er implementering ska heta `SortedLinkedListSet.java`.

Splay-träd

Implementera `SimpleSet` på nytt, denna gång med ett splay-träd. Liksom i föregående uppgift ska interfacets metoder plus en konstruktor utan argument implementeras. Implementeringen ska heta `SplayTreeSet.java`.

Automatisk testning

Skriv ett körbart program med namnet `TestSetCorrectness.java` som tar fyra argument som alla är heltal. Det första talet, n_1, är 1 om `SortedLinkedListSet` ska testas, 2 om `SplayTreeSet` ska testas. Det andra talet, n_2, bestämmer hur många omstarter som ska göras (se nedan). Det tredje talet, n_3, bestämmer hur många slumpoperationer som ska utföras mellan varje omstart. Det fjärde talet, n_4, bestämmer antalet olika heltal som slumpvis används vid operationerna.

Programmet ska jämföra den valda (alltså beroende på n_1) implementeringen med en implementering av `Set` i javas api. Elementtypen ska vara heltal. För varje omstart ska nya instanser av den testade implementeringen och referensimplementeringen skapas. Sedan ska n_3 operationer utföras, samma operationer för både testade och referensimplementeringen. Vilken operation som utförs ska varje gång väljas slumpvis med 25% chans vardera för de fyra metoderna i `SimpleSet`. För de metoder som tar ett element som indata (alla utom `size`) ska detta väljas slumpvis bland n_4 olika tal, t.ex. mellan 0 och $n_4 - 1$. För varje

operation som utförts ska resultatet från den testade och referensimplementeringen jämföras. Om resultaten inte stämmer överens ska programmet meddela att en bugg hittats och avbryta testningen. Det finns inga speciella krav för hur buggen ska redovisas, men om era implementeringar har buggar kan det vara en bra idé att skriva ut sekvensen av operationer som utförts sedan omstarten. Detta för att enklare förstå hur buggen uppstår.

Se till att båda era implementeringar kan testas av ert testprogram utan att fel hittas. När ni kör test kan ni t.ex. sätta $n_2 = 10000, n_3 = 100, n_4 = 20$.

Tidskomplexitet

Använd givna programmet `TestSetSpeed.java`. Det tar två heltal som argument. Det första väljer implementering på samma sätt som `TestSetCorrectness` ska göra. Det andra talet bestämmer storleken på mängden, alltså variabeln n i tidskomplexiteten. Programmet utför ett antal `contains`, `add` och `remove` och räknar ut genomsnittslig tidsåtgång i nanosekunder för en operation (alltså ett genomsnitt för alla tre operationerna). Notera att i tiden ingår också anrop till `rand.nextInt()`, men detta beror ju inte av n utan är konstant så vi bortser från det.

För `SortedLinkedListSet`, plotta ett antal (minst 10) uppmätta värden för n mellan 1 och 1000 i ett diagram. Beräkna och motivera den asymptotiska komplexiteten genom att analysera de tre metoderna. Det bör bli samma komplexitet för de tre metoderna. Anpassa en kurva som motsvarar den teoretiska komplexiteten till de uppmätta punkterna i diagrammet. Alltså, om er komplexitet är $O(f(n))$ så välj k och m så att funktionen $k \cdot f(n) + m$ följer punkterna så bra som möjligt (enligt ögonmått räcker). Plotta denna kurva i samma diagram. Hur god är överrensstämmelsen?

Upprepa samma sak för `SplayTreeSet` i ett separat diagram, med värden för n mellan 1 och 10000.

Vad ska vi skicka in till Fire?

- `SortedLinkedListSet.java`, `SplayTreeSet.java`, `TestSetCorrectness.java` enligt beskrivning ovan.
- `lab3.pdf` där ni anger och motiverar (i vanligt språk räcker,d.v.s. handviftning) tidskomplexiteten för `contains`, `add` och `remove` för båda implementeringarna (alltså totalt 6 metoder). Dokumentet ska också innehålla de två komplexitetsdiagrammen samt era kommentarer om överensstämmelsen i dessa.

Lösa filer, inga `.zip`, `.tar` eller dyligt.