En kort guide till korrekthetsbevis med loopinvarianter och induktion för DD1338

Gabriel Skolgund

Inroduktion

Bevis av algoritmers korrekthet är något som många studenter upplever som svårt, och denna text är tänkt som ett komplement till övrigt kursmaterial för att visa vad som bör ingå i ett bevis och hur man kan resonera. Man kan börja med att fråga sig varför vi vill göra ett formellt bevis över huvud taget. Som ni säkert märkt är det svårt att skriva kod, och det är lätt att göra misstag som gör att ett program inte beter sig som man tänkt. Genom att skriva bra testfall kan många av dessa problem hittas, men hur kan man veta att man skrivit tester som verkligen täcker alla eventualiteter? Med hjälp av ett formellt bevis kan vi visa att en bit kod alltid kommer ge den utdata vi väntar oss, så länge den får korrekt indata. Naturligtvis kan detta vara en komplicerad och tidskrävande process, och i verkligheten så är det inte mycket av den kod som skrivs som också blir formellt bevisad korrekt. För applikationer där ett fel kan ha allvarliga konsekvenser är det dock mycket värdefullt att ha ett bevis, och det är en väldigt bra färdighet att lära sig. I den här texten ska vi titta på två närbesläktade metoder för att visa korrekthet, *loopinvarianter* och *induktion*.

Bevis med loopinvarianter

En *invariant* är en egenskap som inte ändras under tiden vår kod körs. En invariant kan exempelvis vara *värdet på variabeln x är allid 5*, men invarianter kan även vara definerade i termer av andra variabler, som *värdet på variabeln x är alltid mindre än värdet på variabeln y*, eller *array[0] är alltid det minsta värdet i arrayen*. Då vi visar korrekthet stöter vi ofta på loopar, och här är invarianter användbara för att bevisa att loopen modifierar variabler på ett sätt som leder till ett korrekt resultat. Betrakta följande funktion för att hitta det största talet i en osorterad array:

```java
/**
* Find the largest integer in an array
* @param array the array in which to search.
*              The array must be non-empty
* @return the largest integer value in the array
*/
public int max(int[] array) {
    int max = array[0];
    for (int i = 1; i < array.length; i++) {
        // Invariant: max is the largest value in the array
        // between the indices 0 and i - 1 (inclusive)
        if (max < array[i]) {
            max = array[i];
        }
    }
    return max;
}
```

Vi vill visa att påståendet i JavaDoc-kommentaren faktiskt är korrekt, det vill säga att funktionen faktiskt returnerar det högsta värdet i arrayen så länge denna inte är tom (vad kommer hända om den är det?). I funktionen är en invariant redan utskriven för variabeln `max`. Vi kan använda denna för att visa korrekthet för hela funktionen, men då måste vi visa tre separata egenskaper:

1. Invarianten är sann innan loopen körs (*initialization*)
2. Om invarianten är sann i början på en iteration, så kommer den vara det även i början på nästa iteration (*maintainance*)
3. När loopen körts klart har invarianten en egenskap som hjälper oss att visa korrektheten för funktionen (*termination*)

Vi visar dessa som följer:

1. Innan loopen börjar köras så sätts `max` till `array[0]` och `i` sätts till 1. Invarianten säger att `max` ska vara det största talet mellan `array[0]` och `array[i-1]` = `array[0]`, vilket är uppenbart sant, då det bara finns ett värde i detta intervall, och `max` har detta värde.
2. Antag att invarianten är sann i början på den iteration då `i` $= n$, d.v.s. att `max` är det största värdet mellan `array[0]` och `array[n-1]`. Det finns nu två möjligheter: `max` $<$ `array[n]` eller `max` $\geq$ `array[n]`. I det

första fallet så kommer `max` ändras till värdet `array[n]`, och i det andra fallet så förblir variabeln oförändrad, i båda fallen är `max` garanterat det största talet mellan arrayindex 0 och n. Det sista som sker i loopen är att `i` ökar med ett till $n + 1$. När vi sedan kommer in i nästa iteration så ser vi därmed att invarianten är sann även här.

3. Då loopen avbryts är `i` = `array.length`, och invarianten säger att `max` är det största talet mellan `array[0]` och `array[array.length - 1]`, men detta är ju hela arrayen, så `max` är därmed det största talet arrayen och funktionen kommer alltså returnera ett korrekt värde.

För att vårt bevis ska vara fullständigt så är det alltså nödvändigt att vi genomgående visar dessa tre egenskaper, samt att vi tydligt skriver ut vilken loopinvariant vi använder oss av. När man ställer upp en invariant är det en bra idé att börja i slutet, d.v.s. vad vill vi visa och hur måste *termination*-egenskapen se ut i så fall?

Induktionsbevis

Induktionsbevis har ni förhoppningsvis redan stött på i matematiken, och de bevis vi gör inom datalogin följer exakt samma principer. För att ett bevis ska vara fullständigt så måste vi visa en *induktionsbas*, det vill säga att det påstående vi gör om algoritmen är sant för något initialt värde. Vi måste ställa upp vårt *induktionsantagande*, det vill säga att påståendet är sant för något heltal n, och sedan visa att påståendet även är sant för $n+1$ (*induktionssteget*). I induktionssteget måste vi använda vårt induktionsantagande, och om vi lyckas visa att påståendet gäller för $n + 1$ givet att det gäller för n, och att vi visat att det existerar ett sådant n (basfallet), så följer det att påståendet gäller för alla naturliga tal större än eller lika med basfallet. För algoritmer är såklart påståendet nästan alltid att algoritmen ger ett korrekt resultat, och metoden lämpar sig särskilt väl för rekursivt definierade algoritmer.

Låt oss använda denna metod för att visa korrektheten för en annan implementatation av `max`-funktionen. Denna version hittar rekursivt det största värdet i den övre och nedre halvan av ett arrayintervall, och returnerar det största av dessa. Observera att metoden tar två extra parametrar som anger vart intervallet börjar och slutar, så för att hitta det maximala värdet i en hel array bör vi anropa metoden som `max(array, 0, array.length - 1)`. Anledningen till att funktionen är skriven på detta vis är att det är mycket mer effektivt än att göra kopior av de båda arrayhalvorna med exempelvis `Arrays.copyOf`.

```java
/**
* Find the largest integer in an array section
* @param array the array in which to search.
*              The array must be non-empty
* @param lower the lower bound of the array section (inclusive)
* @param upper the upper bound of the array section (inclusive)
* @return the largest integer value in the section
*/
public int max(int array[], int lower, int upper) {
    if (lower == upper) {
        return array[lower];
    }
    int mid = (upper + lower) / 2;
    int lowerMax = max(array, lower, mid);
    int upperMax = max(array, mid + 1, upper);
    if (lowerMax > upperMax) {
        return lowerMax;
    } else {
        return upperMax;
    }
}
```

Vi börjar först med att identifiera exakt vad vi gör induktion över. Kom ihåg att vår induktionsvariabel måste vara ett naturligt tal, och här väljer vi arrayintervallets längd, definierat som $upper - lower + 1$. I just detta fall använder vi även stark induktion, det vill säga att vi gör antagandet att algoritmen är korrekt för *alla* tal mindre än eller lika med något tal n. Detta är på grund av att vi delar arrayen i två delar, vilka kan vara av olika längd. Huruvida man vill använda stark eller vanlig induktion beror på hur algoritmen är utformad, men rent praktiskt så sker beviset på samma sätt. Vi går helt enekelt igenom och visar de tre stegen:

Induktionsbas: `max` ger ett korrekt resultat på alla intervall av längd 1. Detta är uppenbart sant då alla sådana intervall enbart innehåller ett tal (som därmed är det största), vilket returneras av `max` .
Induktionsantagande: Antag att `max` ger ett korrekt resultat på alla intervall av längd $1 \leq \ell \leq n$ för något heltal n.
Induktionssteg: Vi ska nu visa att `max` ger ett korrekt resultat på intervall av längd $n + 1$. Ett anrop till `max(array, k, k + n)` kommer returnera det största värdet av `max(array, k, k + n/2)` och `max(array, k + n/2 + 1, k + n)`. Dessa har längd $\lfloor \frac{n}{2} \rfloor + 1$ respektive $n - \lfloor \frac{n}{2} \rfloor$, vilka båda är mellan 1

och n.[1] Enligt induktionsantagandet kommer dessa båda anrop ge det största talet i sina respekive intervall, och eftersom dessa intervall tillsammans täcker alla arrayindex i det ursprungliga intervallet av längd $n + 1$ så ger `max` ett korrekt resultat även på intervall av denna längd.

Vi har därmed visat att `max` ger ett korrekt resultat på alla arrayintervall av längd minst 1. Att skapa sådana här bevis kan kännas lite knepigt, men se till att ta ett steg i taget och fundera noga på exakt vad ni vill visa så ska det nog gå bra när ni gör det själva.

[1] Notationen $\lfloor x \rfloor$ innebär att x rundas nedåt till närmsta heltal, d.v.s. $\lfloor \frac{5}{2} \rfloor = \lfloor 2.5 \rfloor = 2$