Opgave 1: IBAN-Calculator

Concurrency 2015/2016

November 2015

Achtergrond

Sinds begin 2014 zijn in Nederland de ouderwetse bankrekeningnummers afgeschaft en wordt alleen nog maar gebruik gemaakt van zogenaamde IBAN-nummers. Bij een niet-nader-te-noemen organisatie die een bepaald bereik van bankrekeningnummers beheert zijn echter helaas nog wat problemen met de invoering. Jouw taak is om deze organisatie te helpen door *te bepalen hoeveel geldige bankrekeningnummers (oud formaat) het gegeven bereik bevat.*

Opdracht

Het is bij deze opdracht de bedoeling dat je een programma schrijft dat het aantal bankrekeningnummers dat de organisatie beheert telt. Omdat het aantal bankrekeningnummers heel groot kan zijn, moet je programma met meerdere threads te gelijk kunnen werken. Naast de *telmodus* moet je programma nog twee additionele taken kunnen uitvoeren: de *lijstmodus* en de *zoekmodus*. Niet ieder bankrekeningnummer is een geldig nummer: de nummers bevatten een checksum-systeem. Of een rekeningnummer geldig is kan je testen door middel van een aangepaste elfproef, die we hierna m-proef noemen.

De m-Proef

Bij de elfproef worden alle cijfers gewogen bij elkaar opgeteld, afhankelijk van hun positie, om vervolgens te controleren of deze som een elfvoud is. Het laatste cijfer wordt met 1 vermenigvuldigd, het voorlaatste met 2, etc. Een voorbeeld met bankrekeningnummer 27.48.56.190:

$$2 \times 9 + 7 \times 8 + 4 \times 7 + 8 \times 6 + 5 \times 5 + 6 \times 4 + 1 \times 3 + 9 \times 2 + 0 \times 1 = 220 = 20 \times 11$$

In de m-proef moet de som een m-voud moet zijn (dus niet altijd 11). Een bankrekeningnummer heeft niet noodzakelijk 9 cijfers, het kunnen er ook meer of minder zijn. Het k^{e} cijfer van rechts wordt altijd met k vermenigvuldigd.

Invoer

Je moet bij deze opdracht een C#-programma schrijven dat in- en uitvoer via de console doet. De invoer bestaat uit één regel met daarop 6 getallen en 1 string, gescheiden door spaties. Dit zijn:

1. Een integer l, het **lock-type** dat gebruikt moet worden: 0 staat voor een zelfgebouwd TaS- of TTAS-lock, 1 het ingebouwde `lock`-statement van C#.
2. Een integer b met $0 \leq b < 2^{31} - 1$, de **inclusieve ondergrens** van het zoekbereik.
3. Een integer e met $b < e \leq 2^{31} - 1$, de **exclusieve bovengrens** van het zoekbereik.
4. Een integer m met $1 \leq m \leq 256$, de **modulus** die gebruikt moet worden in de m-proef.

5. Een integer p met $1 \leq p \leq 256$, het **aantal threads** dat je programma moet gebruiken om te rekenen.

6. Een integer u, de **programmamodus** die opgestart moet worden: 0 staat voor telmodus, 1 voor lijstmodus en 2 voor zoekmodus.

7. Een spatieloze string h die alleen gebruikt wordt in de zoekmodus. Dit is de **hash** waar naar gezocht wordt. Dit wordt alleen in de zoekmodus (modus 2) gebruikt, als het programma niet in zoekmodus wordt gebruikt moet deze string genegeerd worden.

Locking

Het programma moet met meerdere threads tegelijk kunnen zoeken. *De gehele range moet eerlijk verdeeld worden over de verschillende threads, zonder gaten of overlap.* Als het bereik niet deelbaar is door het aantal threads mag het verschil tussen het aantal rekeningnummers dat iedere thread bekijkt maximaal 1 zijn (zo eerlijk mogelijk verdelen).

Je moet bij deze opdracht verschillende soorten *locks* implementeren. Als je een critical section wil maken in je programma dan dien je deze af te schermen met, afhankelijk van de waarde van l in de invoer,

$l = 0$ - een zelf te implementeren TaS- of TTaS-lock

$l = 1$ - het ingebouwde `lock`-statement van C#

Om je eigen lock te implementeren heb een methode nodig uit `System.Threading.Interlocked`. Lees goed de documentatie hiervan na om een geïnformeerde keuze te maken over welke methode je waar gebruikt. Het is niet toegestaan om buiten de implementatie van de locks gebruik te maken van de methoden uit `System.Threading.Interlocked`.

Probeer het wisselen tussen de twee manieren van locking op een nette (objectgeorienteerde) manier te doen.

Hashing

Er is een bevel van de Europese Unie binnen gekomen om de bankrekening van een vertrouweling van de Russische president Putin te blokkeren. Bij dit soort bevelen wordt nooit direct het rekeningnummer opgegeven, om spionage te verhinderen wordt alleen de SHA1 hash van het rekeningnummer en het bereik waarin het nummer te vinden is gestuurd. Deze hash is hexadecimaal weergegeven zoals ook te zien op http://caligatio.github.io/jsSHA/. Schrijf een programma dat zo snel mogelijk uitrekent welk bankrekeningnummer geblokkeerd moet worden, want bij sancties is haast geboden.

Zorg dat het programma van elk gevonden bankrekeningnummer de SHA-1 hash vergelijkt met de gegeven hash, en termineer zo snel mogelijk alle threads zodra het juiste nummer gevonden is. Er is altijd hoogstens één nummer dat overeenkomt met de gegeven hash.

.NET heeft de mogelijkheid om strings met SHA1 te hashen in `System.Security.Cryptography`. Let er op dat je programma nog steeds bij elke juiste invoer moet kunnen termineren.

Programmamodi

Afhankelijk van het getal u dat op de invoer staat, moet je programma verschillende taken kunnen uitvoeren. De verschillende modi zijn:

u = 0 - Telmodus

In deze modus moet je het aantal gehele getallen x met $b \leq x < e$ tellen dat aan de m-proef voldoet. De threads moeten een *gedeelde teller* bijhouden waarin het antwoord wordt opgeslagen. Uiteindelijk is de uitvoer één getal: het aantal geldige rekeningnummers in het opgegeven bereik.

u = 1 - Lijstmodus

De lijstmodus lijkt op de telmodus, maar in plaats van 1 getal is de uitvoer een lijst van alle geldige bankrekeningnummers in het bereik. Per geldig rekeningnummer wordt 1 regel op de uitvoer geschreven, bestaande uit twee getallen: het volgnummer, gevolgd door een spatie, gevolgd door het gevonden rekeningnummer (zie voorbeelden voor meer details). Als een geldig rekeningnummer wordt gevonden moet het direct weggeschreven worden, je mag niet alles tot het eind opsparen.

u = 2 - Zoekmodus

In de zoekmodus is het de bedoeling dat je een geldig rekeningnumer (komt uit het gegeven bereik en voldoet aan de m-proef) vindt waarvan de SHA-1 hash gelijk is aan de gegeven hash h. Zodra het rekeningnummer met de corresponderende hash gevonden is, moet dit naar de console worden geschreven en *moet het programma zo snel mogelijk afsluiten* (de andere threads mogen niet (veel) verder zoeken).

Het is ook mogelijk dat geen rekeningnummer gevonden wordt, in dit geval moet op de uitvoer alleen het getal -1 komen.

Beoordeling

Om je werk te beoordelen kun je het programma TomJudge1.exe gebruiken. Wij gebruiken dit programma ook bij de beoordeling, dus het is heel verstandig om zelf je programma door TomJudge te laten checken. Qua werking lijkt het een beetje op DomJudge, het programma draait een aantal unit-tests om te kijken hoe goed het programma werkt. DomJudge is echter alleen gebouwd voor single-threaded programma's en kan niet goed overweg met multithreading. We hebben geprobeerd om TomJudge gedetailleerdere feedback te laten geven dan DomJudge. Bij iedere testcase staat wat we er mee testen en we geven ook suggesties wat je kunt doen om problemen op te lossen. De volledige in- en uitvoer is zichtbaar.

Zet het bestand TomJudge1.exe in dezelfde map als de *.cs-bestanden van je project. TomJudge zoekt vervolgens automatisch in de `./bin/debug`-map naar de debug-versie van je executable. Het is belangrijk dat je voordat je test met TomJudge eerst zorgt dat de debug-versie van je project up-to-date is (gebruik F5 of F6 in Visual Studio). Je kunt eventueel ook de locatie van je executable doorgeven als argument, voer dan in de opdrachtprompt in: `TomJudge1 "C:/Concurrency/Programma.exe"`. Als je vervolgens op de knop om te testen drukt worden de tests gedraaid, dit kan enkele minuten duren. Als het goed is wordt alles groen, als het (nog) niet goed is worden sommige dingen rood. Klik eventueel op de testcase voor meer details en hints en probeer het te verbeteren. **Een programma dat alle testcases correct heeft, scoort waarschijnlijk een voldoende**. Een aantal mogelijke uitzonderingen hierop is:

- Je programma mag geen code bevatten die specifiek is aan de gebruikte testcases of expres bepaalde delen van het programma vertraagt.
- Er moet aan de implementatie-eisen in de opdracht (zoals het implementeren van de 2 verschillende manieren van locking en het gebruik van (niet) toegestane methoden) voldaan zijn.

TomJudge kan enkele criteria van het programma niet beoordelen. Naast correcte werking in TomJudge kijken we ook nog naar de volgende factoren (die het deel van het cijfer boven de voldoende bepalen):

- Implementatie van je eigen lock.
- Goede verdeling van het zoekbereik.
- Threads op elegante wijze joinen aan het eind (geen busy wait).
- Programmeerstijl

Ook als je programma niet alle testcases correct heeft kun je nog steeds een voldoende halen. Het testprogramma dient alleen ter indicatie.

TomJudge werkt alleen op Windows. Het daarom is niet verplicht om TomJudge te gebruiken maar voor het nakijken is het handig als je het bestand `report.csv` dat TomJudge aanmaakt ook inlevert. Let er ook op dat TomJudge op verschillende computers andere beoordelingen kan geven, het is voor ons immers niet te doen om het op al jullie computers uit te proberen. TomJudge geeft alleen een indicatie en is bedoeld om te helpen, maar we gaan bij het nakijken er van uit hoe je programma op onze computer wordt beoordeeld.

Voorbeelden

- Tel 11-geldige nummers in klein bereik:
  `0 274856170 274856190 11 4 0 0`

  Dan is de uitvoer:
  `2`
- De lijstmodus noemt de nummers ook op in de volgorde waarin ze gevonden worden:
  `0 274856170 274856190 11 4 1 0`

  De volgorde van rekeningnummers is dus willekeurig, maar de regelnummers zijn oplopend:
  `1 274856182`
  `2 274856174`
- Het moet ook werken met een andere m dan 11:
  `0 374856170 374856250 12 4 1 0`

  Deze nummers voldoen aan de 12-proef:
  `1 374856173`
  `2 374856238`
  `3 374856181`
  `4 374856246`
  `5 374856202`
  `6 374856210`
- Test zeker deze zoekinvoer om te weten dat je de SHA1 functie goed toepast:
  `0 274856170 274856190 11 4 2 c736ca9048d0967a27ec3833832f7ffb571ebd2f`

  Met uitvoer:
  `274856182`
- Je programma moet niet alleen termineren als de hash gevonden wordt, maar ook als er geen oplossing bestaat:
  `0 274856170 274856190 11 4 2 77ba9cd915c8e359d9733edcfe9c61e5aca92afb`

  Met uitvoer:
  `-1`