Proefstuderen Informatica

Raphael kena Poss

Introductie

1. Open een terminal, dan vor het volgende commando uit:

   ```
   mkdir opdracht-JOUWNAAM
   ```

   Vervang `JOUWNAAM` door je naam en/of die van je werkpartner.

   > **Tip**
   >
   > - vraag een assistent voor hulp om een terminal te vinden/herkennen.
   > - om een commando opnieuw te roepen, gebruik de pijltoets naar boven.

2. Verplaats je shell naar de nieuwe map:

   ```
   cd opdracht-JOUWNAAM
   ```

   Dan download de bestanden direct in je map:

   ```
   wget  https://science.raphael.poss.name/proefstuderen.zip
   unzip proefstuderen.zip
   ```

3. Voer het volgende commando uit:

   ```
   python bench.py
   ```

   Wat zie je? Het programma `bench.py` informeert je dat het verdere input nodig heeft.

4. Door middel van een teksteditor, tik het volgende in een nieuw bestand `mijnzoek.py` in dezelfde map als eerder:

   ```
   def zoek(n, tabel, waarde):
       for i in xrange(n):
          if tabel[i] == waarde:
             return True
       return False
   ```

   Let goed op de hoeveelheid spaties aan het begin van ieder regel.

5. Voer het volgende commando nu uit:

```
python bench.py mijnzoek.py 100000 10
```

Wat zie je? Het programma `bench.py` executeert je functie `zoek` eerder ingetikt meerdere keer, iedere keer met een nieuwe waarde voor `n`, dan meet zijn executietijd. De eerste kolom geeft de waarde van `n` aan, de 2de kolom geeft de executietijd aan, gemeten in seconden.

De metingen worden uitgevoerd voor verschillende waardes van `n` tussen 100 en het eerste getal in het commando (bvb 100000 hierboven). Probeer verschillende getallen en zie het resultaat voor jezelf.

Er zijn 10 metingen zichtbaar. Het aantal metingen bepaal je door het laatste argument. Probeer verschillende aantallen (bvb. 5, 10, 20) en zie het resultaat voor jezelf.

Wat betekent het hallemaal?

De functie `zoek` hierboven is een voorbeeld belangrijk algoritme in de informatica: het zoeken van een bijzondere waarde in een tabel van `n` verschillende elementen.

Tijdens deze proefstuderenmiddag zal je leren omgaan met het meten van executiesnelheid en algoritmen vergelijken op basis van wetenschappelijke metingen van hun prestaties. Het gegeven programma `bench` is een hulpmiddel om de prestaties van een willekeurige zoekfunctie te meten.

Een diagram maken en resultaten visualiseren

6. Draai het commando opnieuw zodat `n` tot en met 2.000.000 loopt en zodat je 30 metingen krijgt.

7. In de terminal, selecteer met je muiscursor alle regels met metingen.

8. Start een spreadsheetprogramma.

9. Plak de metingen in de spreadsheet.

10. Maak een (X,Y) grafiek met de 2 kolommen metingen.

    Welke informatie wordt hier weergegeven? Geef goede labels aan de X en Y assen. Noem het hele grafiek Executietijd Python 1.

    Wat is de algemene vorm van deze grafiek? Wat weet je daarvan in de wiskunde? Noteer even op een stuk papier of een digitale kladblok wat je daarvan vindt.

    (Je opmerkingen hier heten een wetenschappelijke hypothese.)

11. De kale resultaten laten de totale executietijd zien. Een interessantere metriek om de prestaties van programma's te bestuderen is het aantal operaties per seconde. Dit heet executiesnelheid.

    We kunnen hier dus bekijken hoeveel elementen worden bezocht per seconde.

    Door middel van een spreadsheetberekening, defineer een 3de kolom met het gemiddelde aantal elementen die worden gemeten per seconde (= waarde van `n` gedeeld door de executietijd).

12. Maak een tweede (X,Y) grafiek met de waardes van `n` as X-as en de executiesnelheid als Y-as. Geef weer goede labels aan de X en Y assen. Noem de hele grafiek Snelheid Python 1

    Wat is de algemene vorm van deze grafiek? Komt het goed overeen met je voorspellingen op stap #10? (Je opmerkingen hier heten de evaluatie van een wetenschappelijke hypothese.)

    Zelfs als je niet precies weet wat je hier moet zeggen, ga gewoon door.

Een andere implementatie vergelijken

Het programma `bench` en de `zoek` functie tot nu toe waren geschreven in de programmeertaal Python. In deze sectie, zullen we de metingen vergelijken met hetzelfde algoritme geschreven in de programeertaal C.

13. Door middel van een teksteditor, tik het volgende in een nieuw bestand `mijnzoek.c` in dezelfde map als eerder:

```
int zoek(int n, int tabel[], int waarde)
{
   int i;
   for (i = 0; i < n; i++)
      if (tabel[i] == waarde)
         return 1;
   return 0;
}
```

    Pas goed op alle punctuatietekens.

14. Voer het volgende commando nu uit:

```
cc bench.c mijnzoek.c -O3 -lm -lrt && ./a.out 100000 10
```

    Wat zie je? Zoals eerder, het programma `bench`, nu in C geschreven, executeert je functie `zoek` (ook nu in C geschreven) meerdere keer, weer iedere keer met een nieuwe waarde voor `n`, en meet zijn executietijd in seconden. De betekenis van de 2 waardes rechtaan in het commando is dus gelijk aan eerder.

15. Voer de stappen #6-#12 van eerder weer uit met het nieuwe programma; zorg ervoor dat de gegevens en grafieken voor C onder de eerdere gegevens en grafieken voor Python komen te staan in de spreadsheet. Noem de 2 nieuwe grafieken Executietijd C 1 en Snelheid C 1.

16. Plaats de grafiek Snelheid Python 1 naast Snelheid C 1.

    Welk programma is het snelste?

    Probeer een formule in de spreadsheet te vinden met een berekening van hoeveel sneller de ene is ten opzichte van de andere.

    Het verschil tussen de twee snelheden heet een implementatieverschil of implementatieverbetering.

    Praat met je medestudenten om te peilen of hun resultaten verschillen.

Pause en overleg

Hier zal de docent een snellere zoekfunctie uitleggen, die binaire zoek heet.

(Het principe is hetzelfde als het zoeken van een woord in een woordenboek: wetend dat alle waardes al gesoorteerd zijn, is het niet nodig om de gezochte waarde te vergelijken met alle waardes vanaf het begin! Het kan slimmer door naar het midden eerst te kijken, dan óf links óf rechts doorgaan met zoeken op basis van waar het woord valt in het woordenboek.)

Als je al een beetje weet programmeren, probeer je eigen versie van dit algoritme uit te vogelen. Anders, wacht even op de oplossing van de docent.

Resultaten met de verbeterde versie

17. Plaats de code voor het snellere algoritme in het bestand `zoek_snel.py`.
18. Door middel van `bench.py`, zoals eerder, meet de prestatie van het nieuwe algoritme zodat `n` tot en met 2.000.000 loopt en zodat je 30 metingen terugkrijgt.
19. Maak weer 2 grafieken aan voor de executietijd en de snelheid. Noem de 2 grafieken Executietijd Python 2 en Snelheid Python 2.
20. Plaats je grafieken Snelheid Python 1 en Snelheid Python 2 naast elkaar.

    Welk programma is het snelste? Hoeveel sneller ongeveer?

    Dit verschil heet trouwens een algoritmisch verschil of algorithmische verbetering.
21. Plaats je grafieken Snelheid C 1 en Snelheid Python 2 naast elkaar.

    Welk programma is het snelste? Hoeveel sneller ongeveer?

    Kijk terug naar je eerdere resultaten. Denk dan aan welke is beter: een algorithmische verbetering of een implementatieverbetering?
22. Plaats de code voor het snelle algoritme in de taal C in het bestand `zoek_snel.c`, dan voer weer de meting uit en produceer de 2 nieuwe grafieken Executietijd C 2 en Snelheid C 2.
23. Plaats je grafiek Snelheid Python 2 naast Snelheid C 2.

    Welk programma is het snelste? Hoeveel sneller ongeveer?

Gevorderde activiteit: voorspellingen

24. Kijk naar de vorm van je grafieken Executietijd Python 2 en Executietijd C 2.

    Welke vorm is dit? Heb je eerder in de wiskunde een functie gestudeerd die dezelfde grafische vorm heeft?
25. Maak een nieuwe spreadsheet aan, dan plak de metingen voor zoek_snel.py (Python dus) in de nieuwe spreadsheet **vanaf regel 3**. Hiermee heb je kolommen A en B gevuld, vanaf regel 3.

    Schrijf dan in de cel C1 de formule: `=(B20-B10)/(LOG(A20)-LOG(A10))`

    Dan in C2 schrijf je: `=B20-C1*LOG(A20)`

26. De twee waardes in C1 en C2 heten samen een **logaritmisch model** van de executietijd. Op basis van *alleen* het model is het mogelijk om een *voorspelling* te maken van de executietijd voor een willekeurige tabelgrootte.

    Bijvoorbeeld, schrijf de volgende formule in C26: `=C1*LOG(A26)+C2`

    Zoals je ziet maakt dit formule geen gebruik van de waarde in B26.

    Vergelijk dan het resultaat in C26 met de waarde in B26. Hoeveel schilt het? Kan je dit als procentage (%) berekenen?

    Probeer het weer op een andere regel. Hoeveel schilt het deze keer?

27. Je kunt ook een grafiek maken van je model.

    Hiervoor, schrijf de volgende formule in C3: `=C$1*LOG(A3)+C$2`

    Dan kopieer en plak de formule in C3 op alle volgende rijen in kolom C.

    Dan maak een diagram met de inhoud van kolom A vanaf rij 3 als X-as, en de waardes in kolommen B en C vanaf rijd 4 as Y-waardes. Komen de twee curves goed overeen?

28. Maak een nieuwe rij in kolom A onderaan, met de waarde 100.000.000. Op basis van de formule in kolom C, maak een voorspelling van hoeveel uren en minuten nodig zouden zijn voor het zoeken van een waarde tussen 100.000.000 door middel van hetzelfde algoritme.

    De waarde die je vindt kan je dan gebruiken door te zeggen door het model van de prestaties van dit algoritme schat ik dat de executietijd voor 100.000.000 elementen zo is (je resultaat op stap #28), met een procentage nauwkeurigheid ongeveer (je resultaat op stap #26 hierboven).

    Dit is wetenschap!

Wat heb je vandaag geleerd?

Wat je naar huis kunt meenemen:

- vandaag heb je een hypothese leren formuleren dan haar testen door middel van metingen. Deze wetenschappelijke kant wordt toegevoerd door een universitaire studie aan de technische kant van de informatica.

- vandaag heb je het implementatieverschil gemeten tussen de prestaties van hetzelfde algoritme in de talen Python en C. Hiermee heb je wetenschappelijke kennis opgebouwd over het verschil tussen de twee talen, goed onderbouwd door feiten (reele metingen). Deze kennis zal je in de toekomst eventueel kunnen helpen om programmeertalen te kiezen voor nieuwe projecten.

  Het bouwen van wetenschappelijke kennis, en hoe deze kennis wordt gebruikt in de praktijk, is de kernzaak van alle wetenschappers.

- vandaag heb je het algoritmische verschil gemeten tussen de prestaties van 2 algoritmen die dezelfde resultaat teruggeven maar met twee verschillende methodes. Hiermee heb je drie stukken wetenschappelijke kennis opgebouwd, onderbouwd door feiten:

  - een zeer sneller algoritme in een trage programmeertaal kan sneller worden dan een traag algoritme in een zeer snellere taal.
  - enorme prestatieverschillen kunnen worden behaald door middel van maar een paar extra regels programmacode.
  - het combineren van algoritmische verbeteringen en implementatieverbeteringen levert de beste resultaten op.

- als je de gevorderde activiteit hebt afgemaakt, heb je ook geleerd om voorspellingen te maken voor onbekende situaties op basis van bekende metingen.

  Dit is ook een fundamentale zaak in de wetenschap, die je vaak zal terugvinden tijdens je studie aan de universiteit.

  Bovendien, door het theoretisch bestuderen van algoritmes, zal je krachtige, algemene technieken leren om voorspellingen te maken zelfs zonder metingen te hoeven doen. Dit is hoe hoogopgeleide informatici beter strategisch kunnen kiezen hoe een programma moet worden gemaakt zodat het ook snel en efficient draait.

Als je tijd over hebt

kan je dan de werkomgeving voor alle informatica vakken beter leren kennen door de volgende tutorial: https://science.raphael.poss.name/intro-unix.html

Dankwoord

Deze onderwijsactiviteit is mede mogelijk gemaakt dankzij hulp van Roy Bakker, Merijn Verstraaten, Simon Polstra, Taddeüs Kroes, Tessa Klunder en Nicky Kessels.

Copyright and licensing

SC fingerprint: `fp:rNXcI4sI4TdFzQlXF4roUaKSXd6mq6es6Ra5avpU6pWKOw`