UNIVERSITEIT GENT

Algoritmen en Datastructuren 2

De Beste Binaire $_{\text{Zoek}}$Boom

Deadlines: 31 oktober en 5 december 2021

1 Inleiding

In dit project willen we voor eens en voor altijd een antwoord op de vraag formuleren *"Welke zoekboom is de beste?"*, of toch een poging wagen.

Om deze vraag goed te beantwoorden, moeten we natuurlijk beslissen wat een zoekboom beter maakt dan een andere. Als we een verzameling van sleutels hebben en op voorhand weten hoe vaak die sleutels gaan opgezocht worden, dan kan je met dynamisch programmeren een zoekboom maken die al deze opzoekingen kan uitvoeren door zo weinig mogelijk toppen te bezoeken. In **oefening 76** van de lesnota's wordt er gevraagd om dit algoritme te vinden.

Helaas kennen we in het echt meestal niet het aantal opzoekingen op voorhand. Het is dus vaak onmogelijk om een zoekboom te maken die écht optimaal is voor de volgende opzoekingen. Je kan proberen een schatting te doen van de distributie, maar deze distributie kan ook veranderen tijdens het uitvoeren van een programma waardoor na verloop van tijd de optimale zoekboom niet meer optimaal is.

Dat probleem wordt opgelost door zelf-organiserende zoekbomen: bomen die zich dynamisch herschikken om zo goed mogelijk proberen te zijn. De **semi-splay**bomen die we zien in de cursus is een voorbeeld van een zelf-organiserende zoekbomen.

2 Opgave

De bedoeling van dit project is om een aantal zelf-organiserende zoekbomen te implementeren en hun performantie met elkaar te vergelijken in een aantal situaties. Je schrijft hiervan een verslag waarin je de resultaten en jouw optimalisaties bespreekt.

2.1 Zoekbomen

We verwachten dat je drie binaire zoekbomen implementeert:

- Een **semi-splay**boom zoals beschreven in de cursus.
- Een binaire zoekboom die op aanvraag optimaal gebalanceerd kan worden.
- Een binaire zoekboom die zichzelf organiseert en kiest wanneer die zichzelf herschikt volgens een heuristiek die je zelf mag kiezen.

Het is de bedoeling dat je deze implementaties zo efficiënt mogelijk probeert te maken aan de hand van benchmarks (volgende sectie). Beschrijf zeker in jouw verslag de verschillende ideeën en optimalisaties die je hebt uitgeprobeerd, want dit zal een belangrijk deel uitmaken van je score van dit project.

Met de derde zoekboom geven we je de vrijheid om te experimenteren en interessante ideeën uit te proberen. Je kan hier experimenteren met twee zaken: hoe je de boom herschikt en wanneer je de boom herschikt. Je kan bijvoorbeeld een boom maken die zichzelf zelf kan detecteren wanneer die niet meer efficiënt is en zichzelf dan opnieuw opbouwt met het algoritme voor de optimale zoekboom.

Bij je eigen boom gaan we je voornamelijk beoordelen op de verschillende ideeën die je hebt uitgeprobeerd en de analyses die je hebt gemaakt. Als je een goed idee had maar dat idee in de praktijk niet goed lijkt te werken, dan is dat niet erg. Zeker als je kan vertellen *waarom* da niet goed werkt.

Tip: ga zeker eens op zoek in de literatuur naar veelbelovende heuristieken. Het algoritme waarover wordt gesproken bij de theoretische vragen is misschien een interessant om eens te bekijken.

2.2 Benchmarks

We verwachten dat je een aantal benchmarks maakt waarmee je de efficiëntie van je implementaties in kaart brengt en verbetert. Test daarbij verschillende situaties uit: verschillende distributies, verschillende hoeveelheid opzoekingen t.o.v. toevoegingen, enzovoort. Kijk ook niet enkel naar de uitvoeringstijd, maar ook naar het **aantal bezochte toppen**.

Een aantal tips bij het schrijven van benchmarks:

- Testen zijn meestal geen goede plaats voor benchmarks, je maakt beter een aparte `main`-methode.
- De uitvoeringstijd van snelle methoden (minder dan 100ms) kunnen soms sterk verschillen. Je kan zo'n methode dan meerdere keren na elkaar uitvoeren om deze schommelingen uit te middelen.
- Bij het schrijven van benchmarks is het belangrijk om enkel tijdsmetingen te doen van wat je wil analyseren. Het genereren van testdata of het uitschrijven van resultaten hou je dus buiten je meting.
- Er zijn een aantal zaken die de uitvoeringstijd sterk kunnen beïnvloeden: een zwaar programma die ook aan het uitvoeren is op een andere rekenkern, het performantieprofiel van je laptop, of je laptop aangesloten is op netstroom, ... Probeer dit zo gelijk mogelijk te houden als je verschillende uitvoeringen met elkaar vergelijkt.
- Gebruik onze `Sampler` klassen om data te genereren volgens een kansverdeling. Bekijk de documentatie van deze klassen voor meer info over het gebruik ervan.

In je verslag verwachten we dat je door middel van grafieken aantoont welke binaire boom het meest performant is in welke situatie.

3 Theoretische vragen

Naast de praktische performantie zijn we ook geïnteresseerd in de theoretische complexiteit. Schrijf in je verslag een antwoord op de volgende theoretische vragen:

Stel dat we een zoekboom hebben die zichzelf niet in balans probeert te houden na elke bewerking. Dat wil zeggen dat we nieuwe items altijd toevoegen als blad en bij het verwijderen van een top die geen blad is die top vervangen door het kleinste linker- of rechterkind. Bij een zoekactie wordt de boom niet gewijzigd.

We willen de boom wel performant houden en voorzien daarvoor een manier om de boom helemaal opnieuw op te bouwen. We kunnen kiezen tussen twee methoden om dit te doen:

- We bouwen de boom op als een zo goed mogelijk gebalanceerde binaire boom.
- We gebruiken een algoritme dat een benadering van de optimale zoekboom kan opbouwen. Het algoritme is simpel: kies de wortel van de boom zodanig dat het verschil in gewicht van de linker deelboom en de rechter deelboom zo klein mogelijk is en doe dit recursief opnieuw op elke deelboom. Indien goed geïmplementeerd, heeft dit algoritme een tijdscomplexiteit van $\Theta(n)$.

Bepaal voor elk van deze twee methoden om de boom opnieuw op te bouwen het volgende:

- Stel dat de boom net opnieuw werd opgebouwd volgens de gegeven methode. Wat is de tijdscomplexiteit per bewerking voor een reeks van n bewerkingen op deze boom, gevolgd door het opnieuw opbouwen van die boom?
- Als we weten dat er heel weinig toevoegingen gaan zijn, krijgen we misschien een ander resultaat. We beginnen opnieuw met een boom die net werd opgebouwd. Wat is de tijdscomplexiteit per bewerking voor een reeks van n bewerkingen op deze boom als er maximaal $\log n$ van deze bewerkingen toevoegingen zijn? Na de $n$de bewerking wordt de boom opnieuw opgebouwd.

Tip: de n volgende operaties kunnen een volledig andere distributie volgen dan waarvoor de boom gemaakt is, bestudeer dus zeker goed het slechtste geval.

4 Implementatie

Je krijgt van ons een repository met enkele klassen en interfaces. Je mag deze klassen **niet** aanpassen. Je mag deze klassen wel naar hartenlust uitbreiden met je eigen subklassen.

- Een klasse `Node` die de toppen in je binaire boom voorstelt.
- Een interface `SearchTree` die een simpele binaire zoekboom voorstelt.
- Een interface `OptimizableTree` die `SearchTree` uitbreidt, die een binaire zoekboom voorstelt die geoptimaliseerd kan worden.
- De klassen `Sampler` en `ZipfSampler` die steekproeven simuleren volgens een kansverdeling, je kan deze klassen gebruiken om data te genereren voor jouw benchmarks.

We verwachten dat je drie klassen implementeert:

1. Een klasse `SemiSplayTree` die de `SearchTree` interface implementeert en die gebalanceerd wordt volgens het semi-splay algoritme beschreven in de cursus. Je zorgt ervoor dat bij het verwijderen de verwijderde top vervangen wordt door de **grootste** sleutel uit de **linkerdeelboom**.

2. Een klasse `OptimalTree` die de `OptimizableTree` interface implementeert en die op aanvraag optimaal georganiseerd kan worden. Deze zoekboom hoeft zichzelf niet in balans te houden.

3. Een klasse `MyTree` die de `SearchTree` interface implementeert en die zichzelf organiseert volgens je zelfgekozen methode.

4.1 Testen en optimalisatie

Implementeer JUnit 5 tests in het mapje `test` om je implementatie op correctheid te testen en zorg dat je alle onderdelen van je implementatie goed test. Schrijf je testen zo vroeg mogelijk tijdens het project zodat je geen tijd verliest met het optimaliseren van een foute implementatie. Houd er rekening mee dat je veel punten kan verliezen als er nog fouten in je implementatie zitten, dus test uitvoerig.

Als je programma correct is, kan je beginnen met optimaliseren. Voer benchmarks uit om kritieke punten te vinden en probeer die dan efficiënter te implementeren. Controleer ook of je optimalisaties effect hebben.

5 Verslag

Schrijf een verslag van dit project. Beantwoord eerst de theoretische vragen en beschrijf dan je implementaties, waaronder mogelijke optimalisaties die je hebt doorgevoerd. Geef een duidelijke beschrijving met pseudocode van je algoritmes of verwijs naar algoritmes in de cursus. Implementatiedetails en Java-specifieke dingen horen hier niet thuis. Beschrijf zeker in detail het eigen algoritme dat je hebt geïmplementeerd.

6 Beoordeling

De beoordeling van het ingeleverde werk zal gebaseerd zijn op de volgende onderdelen:

- Werden de theoretische vragen goed beantwoord?
- Is de implementatie volledig? Is ze correct?
- Hoe is er getest op correctheid?
- Is er zelf nagedacht? Is alles eigen werk?
- Hoe goed presteren de implementaties van de zoekbomen op vlak van uitvoeringstijd en geheugengebruik?
- Zijn de benchmarks goed opgezet?

- Worden de resultaten van de benchmarks goed weergegeven en besproken?
- Is het verslag toegankelijk, duidelijk, helder, verzorgd, ter zake en objectief?
- Is het taalgebruik in het verslag correct?

7 Deadlines

Er zijn twee indienmomenten:

1. Tegen zondag 31 oktober 2021 om 22u00 moet je jouw project als eens ingediend hebben op SubGIT door succesvol te pushen naar de **master** branch waarbij de automatische checks slagen. Het belangrijkste is dat je al een simpele, niet noodzakelijk gebalanceerde, binaire boom hebt geïmplementeerd.

2. We verwachten een volledig project tegen zondag 5 december 2021 om 22u00.

Je code en verslag worden elektronisch ingediend. Opgelet: er hoeft geen papieren versie van het verslag meer ingediend te worden.

Nadien zal elk project individueel besproken worden met een van de assistenten. Het tijdstip hiervoor wordt later bekendgemaakt.

8 Indienen

Om in te dienen werken we vanaf dit jaar met **SubGIT**. Je kan je project beginnen door de volgende stappen te ondernemen:

1. Zorg dat je een account op SubGit hebt en je ssh-sleutel is toegevoegd. Je kan hiervoor instructies vinden op https://subgit.ugent.be.

2. Bij de output van het volgende commando zou `2021-2022/AD2/project` moeten staan, contacteer de assistenten als dat niet het geval is.
   ```
   ssh git@subgit.ugent.be project list
   ```

3. Clone jouw persoonlijke repository naar de gewenste map (hier `project`).
   ```
   git clone git@subgit.ugent.be:2021-2022/AD2/project/STUDENTENNUMMER.git project/
   ```

4. Controleer dat je op de **master** branch zit met `git status`
   Als de eerste lijn niet zegt `On branch master`, voer dan de volgende commando's uit:
   ```
   git branch -m master
   git branch --unset-upstream
   ```

5. Voeg de opgaverepo toe als alternatieve remote en merge die in je repository.
   ```
   cd project/
   git remote add opgave git@subgit.UGent.be:2021-2022/AD2/project-assignment
   git pull opgave master
   ```

Je kan nu aan je project werken in je persoonlijke repository. We raden je sterk aan om regelmatig commits te maken en die te pushen (naar SubGIT of een andere remote).

Om in te dienen zorg je dat je de relevante code **commit** en ook **pusht** naar de **master branch** op SubGIT. Na de deadline zal SubGIT eventjes geen pushes meer accepteren en nemen wij een snapshot van je code. De code en het verslag in die snapshot zullen wij verbeteren.

We verwachten de volgende bestanden in je repository:

- verslag.pdf is de elektronische versie van je verslag in PDF-formaat.
- Een map src met het ongewijzigde package opgave en een nieuwe package oplossing met daarin minstens de drie klassen die je moet implementeren voor dit project: SemiSplayTree.java, OptimalTree.java, en MyTree.java Alle code behalve de testen en benchmarks bevindt zich in deze package.
- De map test bevat alle JUnit-tests.
- De map benchmark bevat al je benchmarks.

Wanneer je code pusht naar SubGIT wordt je code automatisch gecontroleerd of die voldoet aan de gevraagde structuur en worden er enkele simpele testen uitgevoerd. Als je code niet slaagt voor deze automatisch checks dan zal een push naar de **master** branch geweigerd worden.

Wacht dus niet tot het laatste moment om te pushen naar SubGIT want dan loop je het risico niet te kunnen indienen met een nul voor het project als resultaat.

9 Algemene richtlijnen

- Dit is een individueel project en dient dus door jou persoonlijk gemaakt te worden. Het is uiteraard toegestaan om andere studenten te helpen of om ideeën uit te wisselen, maar **het is ten strengste verboden code uit te wisselen**, op welke manier dan ook. Het overnemen van code beschouwen we als fraude (van **beide** betrokken partijen) en zal in overeenstemming met het examenreglement behandeld worden. Op het internet zullen ongetwijfeld ook (delen van) implementaties te vinden zijn. Het overnemen of aanpassen van dergelijke code is echter **niet toegelaten** en wordt gezien als fraude. (Als je je code op GitHub/Bitbucket/sourcehut/Gitlab/... plaatst moet je die privaat houden tot de bekendmaking van examenresultaten van de eerste zit van dit vak).
- Zorg ervoor dat alle code compileert met minstens Java 11. Het is toegelaten om functionaliteit te gebruiken uit latere Java-versies. De automatisch testen zullen uitgevoerd worden in Java 17.
- Extra externe libraries zijn niet toegestaan. De JDK en **JUnit 5** mogen wel.
- Niet-compileerbare code en incorrect verpakte projecten **worden niet beoordeeld** en resulteren dus in **nul punten voor het project**.
- Het project wordt gequoteerd op **4** van de 20 te behalen punten voor dit vak en deze punten worden ongewijzigd overgenomen naar de tweede examenperiode.

- Als een van de twee deadlines gemist wordt, zal het project niet meer verbeterd worden: dit betekent het **verlies van alle te behalen punten voor het project**.
- Essentiële vragen worden niet meer beantwoord tijdens de laatste week voor de deadline. (Dus wacht niet tot vlak voor de deadline om aan het project te beginnen).