Inlämningsuppgift B – Python edition:
Binära sökträd och Röd-svarta sökträd

Uppgiften ska utföras individuellt och består av två delar som examineras stegvis:

- En förberedande teorigenomgång som examineras med ett individuellt kunskapsquiz (se mer information på Canvas om quizzet).

- Implementation och analys av röd-svarta träd (Red-Black trees) examineras baserat på inlämnad programkod. I samband med bedömningen av inlämnad programkod, kan vi i vissa fall komma att be om en kompletterande, muntlig redovisning av inlämnat material.

1. Förberedande teorigenomgång

I den förberedande teorigenomgången ska kapitel 12.1-12.3, kapitel 13 samt Appendix B5 i kursboken[1] läsas igenom noggrant. Du ska även ha tagit del av motsvarande kursmaterial som presenterats på tillhörande föreläsningar och seminarier.

Efter den teoretiska genomgången ska du självständigt minst kunna följande:

- Identifiera ett binärt sökträd (BST) och veta vilka kriterier som ska vara uppfyllda för sökträd som är av typen BST.
- Namnge 3-4 alternativa typer av balanserade binära sökträd (inkl. röda-svarta sökträd) och förklara (mycket översiktligt) hur de strukturellt skiljer sig åt.
- Identifiera ett röd-svart sökträd (RB-träd) och veta vilka kriterier som ska vara uppfyllda för sökträd som är av typen RB-träd.
- Förstå tidskomplexiteten och minneskomplexiteten för olika operationer i ett BST resp. RB-träd.
- Förstå och förklara vilka rotationer som är nödvändiga vid a) insättning av nya element och b) borttagning av specifika element i ett RB-träd.
- Förstå syftet med BST resp. RB-träd och hur de kan användas.

2. Implementation av en abstrakt datatyp för röd-svart träd (RB-träd)

Eftersom RB-träd är ett balanserat binärt sökträd (BST), så rekommenderar vi att du först implementerar nödvändiga operationer för att konstruera, förändra och använda ett BST för heltal. När du fått det att fungera, så går du vidare och utvecklar den funktionalitet som krävs för RB-träd enligt nedan.

Du ska implementera en abstrakt datatyp (ADT) som utgör ett rödsvart träd i Python. Följande krav gäller:

- Implementationen ska vara en klass som heter **RedBlackTree** och ligger i en fil (modul) som heter **red_black_tree.py.**

[1] Cormen T.H., Leiserson C.E., Rivest L.E., Stein C. (2009), *Introduction to Algorithms (3e upplagan)*, MIT Press.

- Klassen ska ha följande publika metoder: **insert(), remove()**, **search()**, **path(), min(), max()**, **bfs()**
    a. **insert(value)** – stoppar in ett värde i **RedBlackTree**-objektet
    b. **remove(value)** – tar bort ett värde ur **RedBlackTree**-objektet
    c. **search(value)** – söker efter ett värde i **RedBlackTree**-objektet, och returnerar **True** eller **False**
    d. **path(value)** – returnerar vägen från roten av trädet till det specificerade värdet. Vägen ska vara i form av en lista som anger alla besökta noders värde från roten fram till och med det angivna värdet. Värdet hos rotnoden ska också finnas i listan.
    e. **min()** – returnerar det minsta värdet i trädet
    f. **max()** – returnerar det största värdet i trädet.
    g. **bfs()** – traverserar trädet i BFS-ordning (bredden- först-sökning) och returnerar resultatet som en lista av noder.
- **Beakta följande begränsningar:**
    a. **insert()** ska inte duplicera värden i trädet (eller ersätta en nod med en annan nod med samma värde)
    b. **bfs()** ska besöka noder från vänster till höger
    c. Varje nod i listan som returneras från **bfs()** ska beskrivas av en ”nod”-lista. En ”nod”-lista ska ha strukturen [*nodens värde*, *nodens färg*, *värde av nodens vänstra barn*, *värdet av nodens högra barn*]. Det betyder att listan som returneras från **bfs()** är en 2D-lista.
    d. Den i c)-punkten nämnda färgen ska vara en sträng med innehållet **’RED’** eller **’BLACK’.**
    e. Om en nod saknar barn ska värdet för det saknade barnet vara **None**.

För ytterligare instruktioner gällande implementation och testning av koden, se avsnittet ”Instruktioner för inlämning av programkod” nedan samt ta del av de olika frågor och svar som finns på kursens forum i Discord.

3. Vad gör filen test_redblacktree.py

När **test_redblacktree.py** körs så läses och testas implementationen av ditt röd-svarta träd från filen red_black_tree.py, varpå rapporter och figurer genereras.

- En kontroll utförs så att alla metoder som krävs finns implementerade.
- Ett antal värden (ca 20 000) sätts in, ett antal värden tas bort och en kontroll av att de är accessbara utförs (riktigheten hos **insert**, **remove** och **search** testas genom detta)
- En kontroll av **path** från roten till alla noder (inklusive roten själv) görs. Det ska gå att hitta path till alla noder.
- Ett histogram genereras över alla path-längder. En del annan statistiskdata åskådliggörs också såsom minimum, medelvärde, median, maximum teoretiskt maximum och standardavvikelse.
- Metoderna **max()** och **min()** testas och deras riktighet rapporteras
- Metoden **bfs()** testas och ett enkelt (men inte fullständigt) test av dess riktighet utförs.
- Testprogrammet använder bfs() en gång till med två mindre test-set av värden (*sample_rbtree_1* och *sample_rbtree_2* med 200-300 värden vardera). För varje test-

set genereras ett rödsvart träd i en Graphviz-fil. (**sample_rbtree1.gv** och **sample_rbtree_2.gv**).

- Du behöver en Graphviz-viewer för att kunna titta på dessa filer. Graphvis-viewer on-line: https://dreampuf.github.io/GraphvizOnline.
- Här testas två huvudregler för röd-svarta träd: 1 - Regeln för röda noders barn. 2 - Regeln om antalet svarta noder från roten till alla löv.

4. Tips för att köra test_redblacktree.py

- Du behöver Python 3.7 eller senare.
- Lägg filen **test_redblacktree.py** i samma mapp som du har din **red_black_treee.py** innehållande din egen klass **RedBlackTree**
- Du behöver ha modulerna **numpy** och **pandas** installerade (och eventuellt fler moduler, testkör på ditt system och installera de moduler du eventuellt saknar med **pip install**)
- När du testar din kod kan du ändra värdet på konstantvariabeln **SPEED_FACTOR**. Den är satt till 1 från början, värdet kan ökas exempelvis till 10 för att snabba upp testningen. Glöm inte att sätta tillbaka värdet till 1 innan du utför dina slutliga tester.
- Det genereras en log-fil när testerna körs som heter **test.log**. I den filen dokumenteras mer detaljerad information från körningen, och den kan vara till hjälp om du stöter på problem med att passera testerna.

5. Instruktioner för inlämning av programkod

Din implementation **ska** bestå av filen **red_black_tree.py** och lämnas in i Canvas på anvisad plats. Precis som under inlämningsuppgift A, kommer den programkod som lämnas in att direktkontrolleras av CodeGrade när inlämning sker i Canvas. Så om programmet inte klarar testerna som görs via CodeGrade eller om filen enligt ovan saknas, så kommer CodeGrade och Canvas inte att acceptera att inlämningen görs. Därför är det viktigt att ni inte väntar till precis innan deadline med att lämna in er programkod.