HEIMADÆMI 5

TÖL203G Tölvunarfræði 2

Kári Hlynsson[1]

Háskóli Íslands

2. mars 2023

Verkefni 1

Þið eigið að breyta táknatöfluútfærslunni `SequentialSearchST.java`, þannig að listinn sé sjálfskipandi (*self-organizing*). Sjálfskipandi gagnagrindur laga sig að notkunarmynstri notandans, þannig að lyklar sem oft er leitað að finnast hraðar en þeir sem sjaldan er leitað að. Þið eigið að útfæra tiltekna útgáfu sem kallast færa-fremst (*move-to-front*). Hún felst í því að þegar kallað er á `get(k)`, þá er hnúturinn með lyklinum `k` færður fremst í tengda listann (ef hann finnst). Það þýðir að ef leitað er aftur að `k` fljótlega þá finnst hann hratt.

Þið eigið að skila breytta fallinu get og skjáskoti af keyrslu á `main`-fallinu fyrir inntakið `A B R A C A D A B R A`, sem þið sláið inn eða pípið úr skrá. Útkoman ætti að vera `D 6, C 4, R 9, B 8, A 10`, þ.e. sætisnúmerin á síðasta tilvikinu af hverjum staf.

Lausn

FORRIT 1 á næstu síðu sýnir breyttu útfærsluna á `get` fallinu í `SequentialSearchST` klasanum. Hugmyndin er sú að ef við finnum stakið sem við erum að leita ítrum við í gegnum tengda listann fram að stakinu og skiptum á því og fyrsta stakinu í tengda listanum. Niðurstaðan er sú að við hliðrum öllum gildunum frá hinu fyrsta fram til þess sem gildið sem var leitað að er í. Síðan er skipt á því og fyrsta stakinu sem er það sem við viljum.

MYND 1 sýnir skjáskot af keyrslu.

[1] Slóð á Github kóða: `https://github.com/lvthnn/TOL203G/tree/master/HD6`

```
66  public Value get(Key key) {
67      if (key == null)
68          throw new IllegalArgumentException("argument to get() is null");
69
70      for (Node x = first; x != null; x = x.next) {
71          if (key.equals(x.key)) {
72            for (Node y = first; y != x; y = y.next) {
73              Value t = y.val;
74              y.val = first.val;
75              first.val = t;
76            }
77
78            Value t = first.val;
79            first.val = x.val;
80            x.val = t;
81
82            return first.val;
83          }
84      }
85      return null;
86  }
```

FORRIT 1: Útfærsla á færa-fremst aðferð í `get` fallinu

```
λ ~/Github/TOL203G/HD6/src/V1/ master* javac SequentialSearchST.java
λ ~/Github/TOL203G/HD6/src/V1/ master* java SequentialSearchST < sample.txt
D 6
C 4
R 9
B 8
A 10
λ ~/Github/TOL203G/HD6/src/V1/ master*
[0] 0:nvim- 1:zsh*                                  "Karis-MacBook-Air-2.l" 21:52 02-Mar-23
```

MYND 1: Keyrsla á `SequentialSearchST.java` í skel

Verkefni 2

Dæmi 3.1.28 á bls. 392 í kennslubók. Það á að breyta fallinu `put` í klasanum `BinarySearchST.java` þannig að ef nýr lykill er stærri en allir lyklarnir í töflunni þá er hann settur inn í föstum (þ.e. $\Theta(1)$ í stað $\Theta(\log N)$). Skilið breytta fallinu `put` og skjámynd af keyrslu á inntakinu `A B C D E F G H`.

Lausn

```
121 public void put(Key key, Value val)  {
122     if (key == null) throw new IllegalArgumentException("first argument
    ↪  to put() is null");
123 
124     if (val == null) {
125         delete(key);
126         return;
127     }
128 
129     int i = rank(key);
130 
131     if (n > 0 && keys[n - 1].compareTo(key) < 0) {
132       if (n >= keys.length) resize(2 * keys.length);
133       keys[n] = key; vals[n] = val; n++;
134       return;
135     }
136 
137     if (i < n && keys[i].compareTo(key) == 0) {
138         vals[i] = val;
139         return;
140     }
141 
142     if (n == keys.length) resize(2*keys.length);
143 
144     for (int j = n; j > i; j--)  {
145         keys[j] = keys[j-1];
146         vals[j] = vals[j-1];
147     }
148     keys[i] = key;
149     vals[i] = val;
150     n++;
151 
152     assert check();
153 }
```

FORRIT 2: Breytt útfærsla á `put` í `BinarySearchTree.java`

MYND 2: Keyrsla á `BinarySearchST.java`

Verkefni 3

Gefið er tvíleitartréð fyrir neðan. Notið Hibbard eyðingu (eins og sýnd er í bókinni og á glærunum) til þess að eyða lyklum út úr þessu tré:

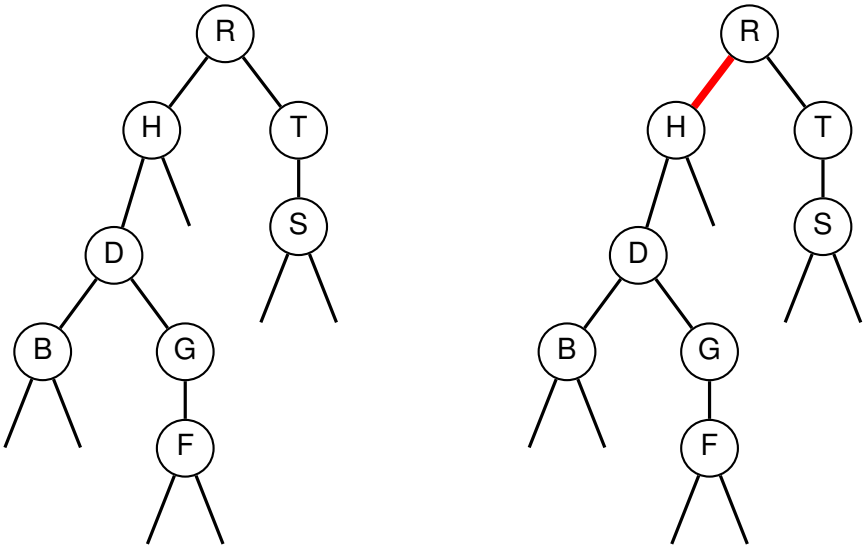
(a) Eyða lyklinum `H` úr trénu. Sýnið hvaða hnúta aðferðin skoðar og teiknið upp lokatréð.

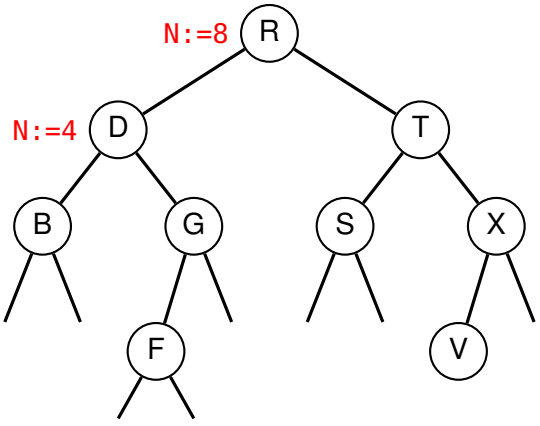
(b) Eyða lyklinum `D` úr *upphaflega* trénu. Sýnið hvaða hnúta aðferðin skoðar og teiknið upp lokatréð.

Lausn

Hluti (a)

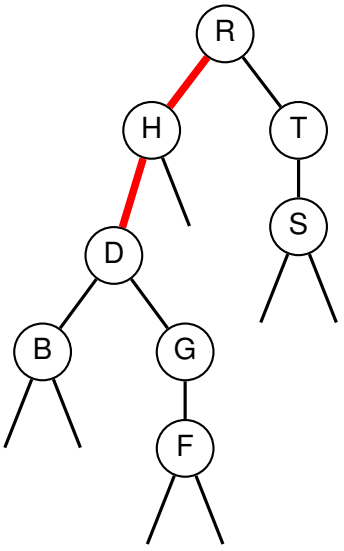
Við byrjum í R. Vegna þess að H $\preceq$ R, þá förum við í vinstra hluttréð. Þá er hægra hluttréð `null` svo við skilum vinstra hluttrénu. Rótin vísar á vinstra hluttréð.



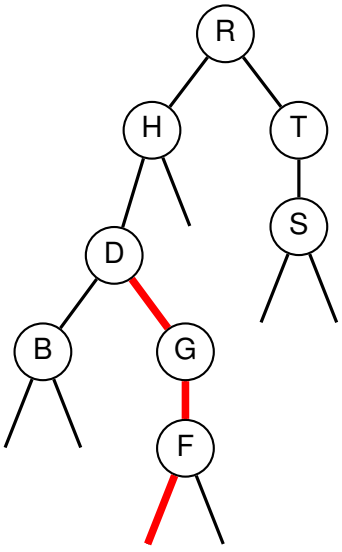


Síðan þurfum við bara að uppfæra `N` fyrir R og D og þetta er komið.

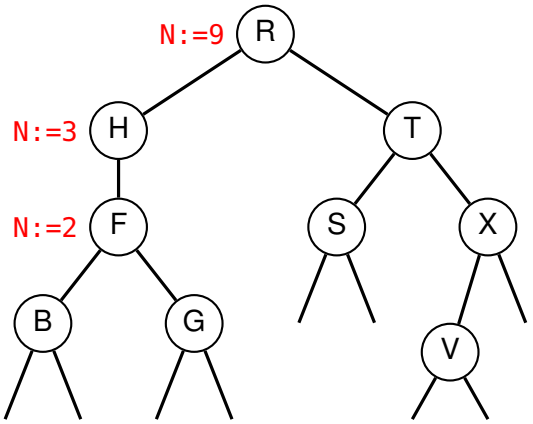
Hluti (b)



Finnum D



Eftirfari er F



Lokastaða

Verkefni 4

Notið áfram upphaflega tvíleitartréð í dæmi 3. Teljið upp hnútana sem skoðaðir eru þegar eftirfarandi tvíleitartrésaðgerðir eru framkvæmdar og gefið skilagildi fallsins.

(a) `ceiling("D")`

(b) `select(7)`

(c) `rank("T")`

(d) `floor(Ę")`

Lausn

Hluti (a)

Við förum í gegnum `R`, `H` og `D` en stoppum þar því `key.compareTo(x) == 0`, og skilum gildinu `D` því gildið er ekki `null`.

Hluti (b)

Við skoðum `R` þegar kallað er á `sleect(root, 7)` og förum þaðan í `T`. Þá gerum við `select(root.right, 1)` og vegna þess að `t` $=$ `k` skilum við gildinu `"T"`, sem er það stak með rankinn 7.

Hluti (c)

Byrjum í `R`, þar sem samanburðurinn er < 0. Skilagildið okkar er þá $1 + 5 +$ `rank(root.right)`. Þar er samanburðurinn $= 0$ og við förum í S. Samanburðurinn er minni en núll og við förum í `null` hnút og því eru skilin á `rank(root.right)` 0 svo við fáum rankinn $1 + 5 + 0 = 6$ fyrir stakið `T`.

Hluti (d)

Við ferðumst endurkvæmt í gegnum hnúta `R`, `H` og `H` með sama hætti og er ferðast í gegnum tvíleitartré. Þegar við erum komin í `G` er samanburðurinn < 0 svo við ferðumst í hægri hnútinn sem er `null`. Því skilum við gildinu `"G"`.

Verkefni 5

Í þessu dæmi eigið þið að skoða hversu há tvíleitartré verða á slembnu inntaki. Ljúkið við klasann `MeasureBST.java` sem býr til n-staka tvíleitartré með `Double` lykli og `Integer` gildi. Lykilgildið er fengið með `StdRandom.uniformDouble()` og `Integer` gildið getur verið hvað sem er. Í hverri tilraun (*trial*) er fundin hæð tvíleitartrésins og lokaniðurstöður forritsins eru meðalhæð tvíleitartrjánna. Forritið á líka að reikna út bestu mögulegu hæð tvíleitartrés með n stök sem er $\lfloor \lg n \rfloor$ og prenta út hversu miklu hærri slembitrén eru miðað við besta mögulegt. Skilið klasanum `MeasureBST` og skjáskoti með keyrslu með $n = 100.000$ og 10 tilraunum.

Lausn

```java
import edu.princeton.cs.algs4.*;

public class MeasureBST {

    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        int trials = Integer.parseInt(args[1]);

        double[] h = new double[trials];

        for (int t = 0; t < trials; t++) {
            BST<Double, Integer> bst = new BST<>();
            for (int i = 0; i < n; i++)
              bst.put(StdRandom.uniformDouble(), 1);

            h[t] = bst.height();
        }

        // Prenta út niðurstöður...
        int opt = (int) Math.floor(Math.log(n) / Math.log(2));
        double avg = StdStats.mean(h);

        System.out.println("For n = " + n + ", optimal height is " + opt);
        System.out.printf("Average height in " + trials + " is %3.2f,
↪  %3.2f times optimal", avg, avg/opt);

    }
}
```

FORRIT 3: Útfærsla á klasanum `MeasureBST.java`

MYND 3: Skjáskot af keyrslu á `MeasureBST.java`