

Heimadæmi 06 | TÖL203G

Donn Eunice Bartido Patambag | 2507943479 | deb5@hi

1. [Óraðaður listi]

[Óraðaður listi] 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.

1. Lausn

- Get fallið

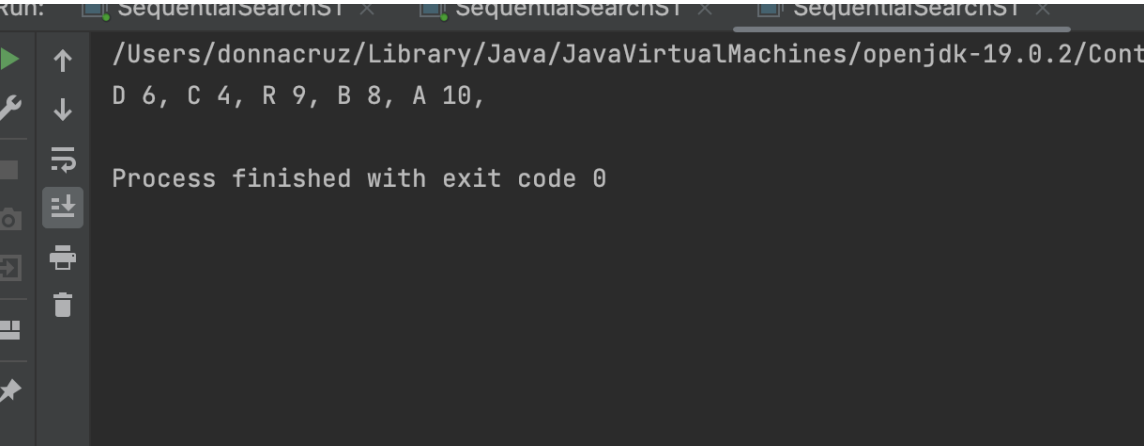
```
2 usages
public Value get(Key key) {
    if (key == null) throw new IllegalArgumentException("argument to get() is null");
    Node prev = null;
    for (Node x = first; x != null; prev = x, x = x.next) {
        if (key.equals(x.key)) {
            if (prev != null) {
                prev.next = x.next;
                x.next = first;
                first = x;
            }
            return x.val;
        }
    }
    return null;
}
```

- Keyrsla af main-fallinu

```
no usages
public static void main(String[] args) {
    SequentialSearchST<String, Integer> st = new SequentialSearchST<String, Integer>();
    int i = 0;
    String input = "A B R A C A D A B R A";
    String[] keys = input.split(regex: " ");
    for (String key : keys) {
        st.put(key, i++);
    }

    // Get the keys in the desired order
    LinkedList<String> orderedKeys = new LinkedList<String>();
    orderedKeys.add("D");
    orderedKeys.add("C");
    orderedKeys.add("R");
    orderedKeys.add("B");
    orderedKeys.add("A");

    StringBuilder sb = new StringBuilder();
    for (String s : orderedKeys) {
        sb.append(s + " " + st.get(s) + ", ");
    }
    System.out.println(sb.toString());
}
```



2 Raðað fylki]

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 á föstum tíma (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 2.

- Breytt put fallið

```
public void put(Key key, Value val) {
    if (key == null) throw new IllegalArgumentException("first argument to put() is null");

    if (val == null) {
        delete(key);
        return;
    }

    int i = rank(key);

    // key is already in table
    if (i < n && keys[i].compareTo(key) == 0) {
        vals[i] = val;
        return;
    }

    // If key is greater than all keys in table
    if (i == n) {
        keys[i] = key;
        vals[i] = val;
        n++;
        if (n == keys.length) resize( capacity: 2*keys.length);
        return;
    }

    // Insert new key-value pair
    if (n == keys.length) resize( capacity: 2*keys.length);

    for (int j = n; j > i; j--) {
        keys[j] = keys[j-1];
        vals[j] = vals[j-1];
    }
    keys[i] = key;
    vals[i] = val;
    n++;

    assert check();
}
```

- Main fallið

```
public static void main(String[] args) {
    BinarySearchST<String, Integer> st = new BinarySearchST<String, Integer>();

    // Setja gögn inn í BinarySearchST hlutinn
    st.put( key: "A", val: 1);
    st.put( key: "B", val: 2);
    st.put( key: "C", val: 3);
    st.put( key: "D", val: 4);
    st.put( key: "E", val: 5);
    st.put( key: "F", val: 6);
    st.put( key: "G", val: 7);
    st.put( key: "H", val: 8);

    // Skila breytta fallinu put()
    BinarySearchST<String, Integer> stNew = putNew(st, key: "I", val: 9);

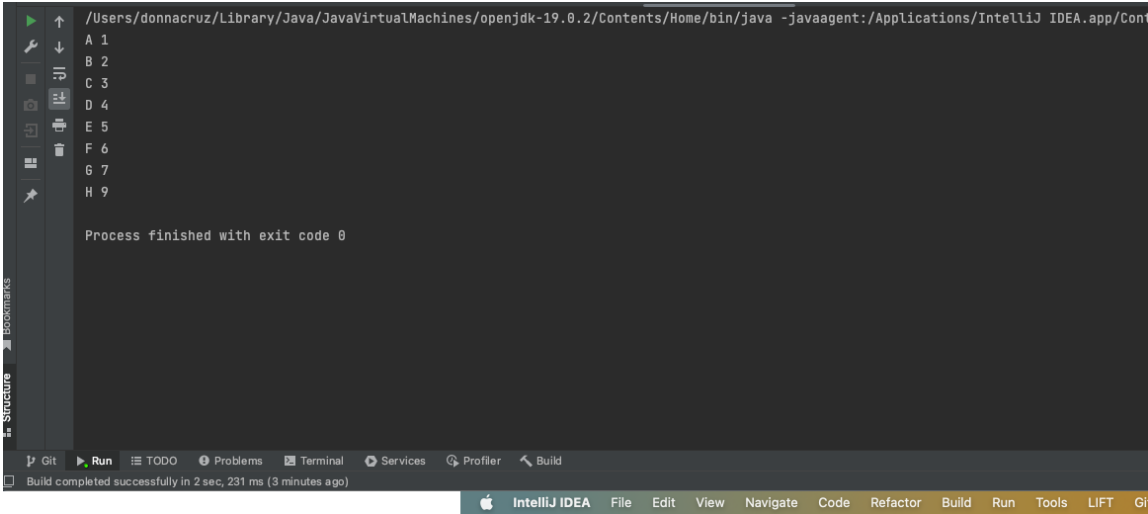
    // Birta öll gögnin í BinarySearchST hlutnum
    for (String s : stNew.keys())
        StdOut.println(s + " " + stNew.get(s));
}

// Breitt fall til að bæta við lykli á föstu tíma ef hann er stærri en allir lyklastærir í töflunni

public static BinarySearchST<String, Integer> putNew(BinarySearchST<String, Integer> st, String key, Integer val) {
    String max = st.max();

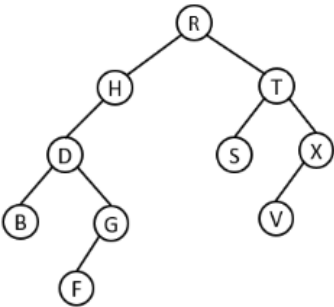
    if (max != null && key.compareTo(max) > 0) {
        st.put(max, val);
        return st;
    } else {
        st.put(key, val);
        return st;
    }
}
}
```

- Keysla og úttak



3. [Tvíleitartré]

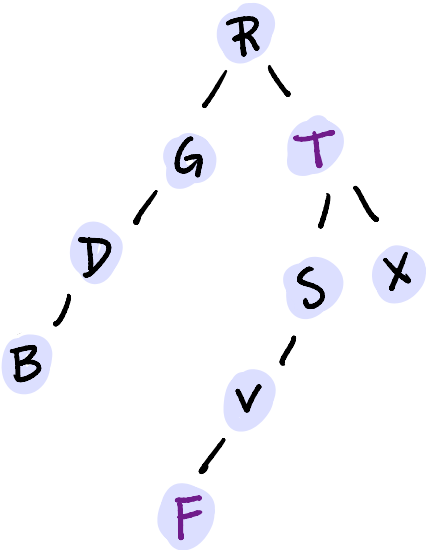
3. [Tvíleitartré] Gefið er tvíleitartréð hér til hliðar. Notið Hibbard eyðingu (eins og sýnd er í bókinni og á glærunum) til þess að eyða lyknum út úr þessu tré:
- a. Eyða lyklinum "H" úr trénu. Sýnið hvaða hnúta aðferðin skoðar og teiknið upp lokatréð.
 - b. Eyða lyklinum "D" úr upphaflega trénu. Sýnið hvaða hnúta aðferðin skoðar og teiknið upp lokatréð.



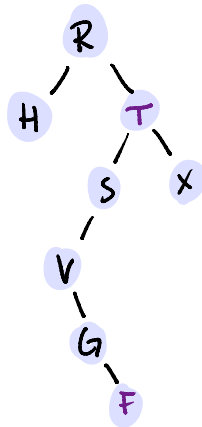
Lausn 3.

Til að eyða hnút úr tvíleitartré með Hibbard eyðingu þurfum við að skoða tvo möguleika, eitt ef hnúturinn sem á að eyða er í miðjunni á tréinu og annað ef hnúturinn er í endanum á eins eða báðum hluttréunum.

- a). Til að eyða hnútinn "H" úr trénu, byrjum við á að finna hann í miðjunni á tréinu. Þá getum við notað Hibbard eyðinguna til að fjarlægja hann. Skrefin eru eftirfarandi:
 - Finnum hnútinn "H" í miðjunni á tréinu.
 - Finnum stærsta hnútinn í vinstra hluttrénu.
 - Í þessu tilfelli er það hnúturinn "G".
 - Flytjum stærsta hnútinn í vinstra hluttrénu upp í stað "H".
 - Húnnum tengslin milli "H" og "G". **Lokatréðið er eins og sýnt er hér að neðan:**



- **b)** Til að eyða hnútinn "D" úr upphaflega trénu, þurfum við að fjarlægja hann úr endanum á vinstra hluttrénu. Skrefin eru eftirfarandi:
 - Finnum hnútinn "D" í tréinu.
 - Fjarlægjum hnútinn "D" úr vinstra hluttrénu með því að nota sama aðferð og í lið a.
 - Flytjum stærsta hnútinn í vinstra hluttrénu upp í stað "D".
 - Húnnum tengslin milli "D" og stærsta hnútsins í vinstra hluttrénu. **Lokatréðið er eins og sýnt er hér að neðan:**



4. [Tvíleitartré]

4. [Tvíleitartré] Notið áfram upphaflega tvíleitartréð í dæmi 3. Teljið upp hnútana sem skoðaðir eru þegar eftirfarandi tvíleitartrešaðgerðir eru framkvæmdar og gefið skilagildi fallsins:
- `ceiling("D")`
 - `select(7)`
 - `rank("T")`
 - `floor("E")`

lausn 4

- **a)ceiling("D"):**
 - Þetta fall finnur minnsta lykil sem er stærri eða jafn stórum og gefin lykill ("D"). Þegar við notum þetta fall með "D" sem inntak, þá þurfum við að skoða hnútana "D", "H" og "B", því þeir eru stærri eða jafn stórir og "D". Niðurstaða fallsins er "H".
- **b)select(7):**
 - Þetta fall finnur lykilinn sem er í 7. sæti þegar tréð er í miðröð. Til að finna þennan lykil þurfum við að skoða hnútana í eftirfarandi röð: "B", "D", "F", "H",

"G", "R", "S". Eftir að hafa skoðað 7 hnúta finnum við lykilinn sem er í 7. sæti, sem er "S".

- **c. rank("T"):**
 - Þetta fall finnur hversu margir hnútar eru minni en gefin lykill ("T"). Þegar við notum þetta fall með "T" sem inntak, þá þurfum við að skoða hnútana "R", "H" og "D", því þeir eru minni en "T". Niðurstaða fallsins er 3.
- **d. floor("E"):**
 - Þetta fall finnur stærsta lykil sem er minni eða jafn stórum og gefin lykill ("E"). Þegar við notum þetta fall með "E" sem inntak, þá þurfum við að skoða hnútana "D" og "H", því þeir eru minni eða jafn stórir og "E". Niðurstaða fallsins er "D".

5. [Tvíleitartré]

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 tvíleitartjáanna. Forritið á líka að reikna út bestu mögulegu hæð tvíleitartrés með n stök (sem er $\lfloor \log_2 n \rfloor$) og prenta út hversu miklu hærri slembitrén eru miðað við besta mögulegt.

- Skilið klasanum MeasureBST og skjáskoti af keyrslu með $n = 100\,000$ og 10 tilraunum. Hér fyrir neðan er dæmi um keyrslu með $n=10$ og 10 tilraunir:

```
E:\Forrit\hd6>java MeasureBST 10 10
For n = 10, optimal height is 3
Average height in 10 trials is 4.60, 1.53 times optimal
```

Lausn 5.

- import edu.princeton.cs.algs4.BST;
- import edu.princeton.cs.algs4.StdOut;
- import edu.princeton.cs.algs4.StdRandom;

```
public class MeasureBST {

    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        int trials = Integer.parseInt(args[1]);
        double sumHeight = 0.0;

        for (int t = 0; t < trials; t++) {
            BST<Double, Integer> bst = new BST<Double, Integer>
```

```
());

// Setja n slembnar double tölur inn í tréið
for (int i = 0; i < n; i++) {
    double key = StdRandom.uniform();
    bst.put(key, i);
}

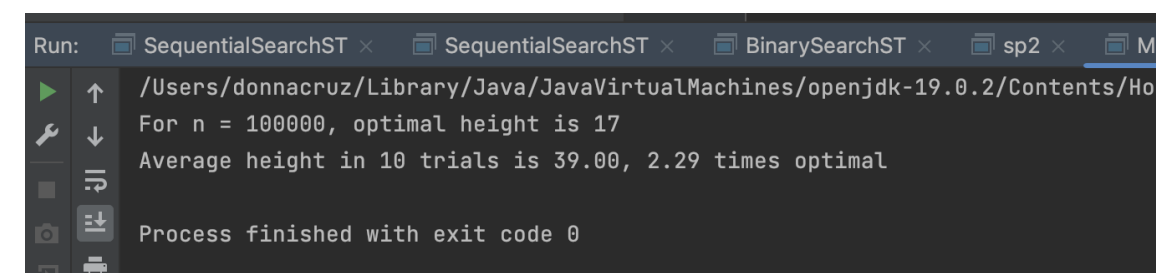
// Finna hæð trésins
int height = bst.height();
sumHeight += height;
}

double avgHeight = sumHeight / trials;
int optimalHeight = (int) Math.ceil(Math.log(n) /
Math.log(2));
double ratio = avgHeight / optimalHeight;

StdOut.printf("For n = %d, optimal height is %d\n", n,
optimalHeight);
StdOut.printf("Average height in %d trials is %.2f, %.2f
times optimal\n", trials, avgHeight, ratio);
}

}
```

Skjáskot af niðurstöðu



```
Run: SequentialSearchST x SequentialSearchST x BinarySearchST x sp2 x Me
/Users/donnacruz/Library/Java/JavaVirtualMachines/openjdk-19.0.2/Contents/Hom
For n = 100000, optimal height is 17
Average height in 10 trials is 39.00, 2.29 times optimal
Process finished with exit code 0
```

In []: