TÖL101G - Tölvunarfræði 1

Vikublað 13

Lausn

Æfingar

4.3.3

Til að leysa verkefnið er gott að nota eftirfarandi reiknirit til að fá útkomuna af staflanum. Skoðum fyrsta stakið í listanum, í dæmi a. er það 4, þá þarf að setja 0-4 upp á staflann til að 4 sé efst og framkvæma svo pop. Því þarf að gera push 4 sinnum og þá er staflinn (0,1,2,3,4) (þ.e. 0 er á botninum og 4 er efst) og pop() prentar svo út 4. Næst þarf að prenta út 3 og því er nóg að gera pop (því 3 er efst á staflanum) o.s.frv.

Í byrjun er staflinn tómur. Við lítum á fyrsta stakið og gerum push() þar til að stakið er komið á toppinn á staflanum (mögulega 0 push aðgerðir). Þá er framkvæmt pop. Ef stakið er í staflanum en ekki á toppnum er ekki hægt að prenta þessa röð.

Í lausninni er sýnt röð push og pop aðgerða til að prenta út listann eða sýnt fram á að það sé ekki hægt.

(a) 5x push, 5x pop (prentar 4 3 2 1 0), 5x push, 5x pop (prentar 9 8 7 6 5)

(b) 5x push, 1x pop (prenta 4), 2x push, 1x pop (prentar 6), 2x push, 5x pop (prenta 8 7 5 3 2), 1x push, 1x pop (prentar 9). Nú er staflinn (0,1) og þar sem 0 er á botninum er ekki hægt að prenta út 0 án þess að prenta 1 fyrst.

(c) 3x push, 1x pop (prentar 2), 3x push, 1x pop (prentar 5), 1x push, 1x pop (prentar 6), 1x push, 2x pop (prentar 7 4), 1x push, 1x pop (prentar 8), 1x push, 4x pop (prentar 9 3 1 0)

(d) 5x push, 5x pop (prentar 4 3 2 1 0), endurtökum (1x push , 1x pop) 5 sinnum (prentar 5 6 7 8 9)

(e) 1x push, endurtökum (1x push, 1x pop) 6 sinnum (prentar 1 2 3 4 5 6), 4x push, 5x pop (prentar 9 8 7 0)

(f) 1x push, 1x pop (prentar 0), 4x push, 1x pop (prentar 4), 2x push, 3x pop (prentar 6 5 3), 2x push, 1x pop (prentar 8). Nú er staflinn (1 2 7) og því ekki hægt að prenta út 1 á undan 7.

(g) 2x push, 1x pop (prentar 1), 3x push, 1x pop (prentar 4), 3x push, 1x pop (prentar 7), 2x push, 5x pop (prentar 9 8 6 5 3). Nú er staflinn (0,2) og því ekki hægt að prenta 0 á undan 2.

(h) 3x push, 2x pop (prentar 2 1), 2x push, 2x pop (prentar 4 3), 2x push, 2x pop (prentar 6 5), 2x push, 2x pop (prentar 8 7), 1x push, 2x pop (prentar 9 0)

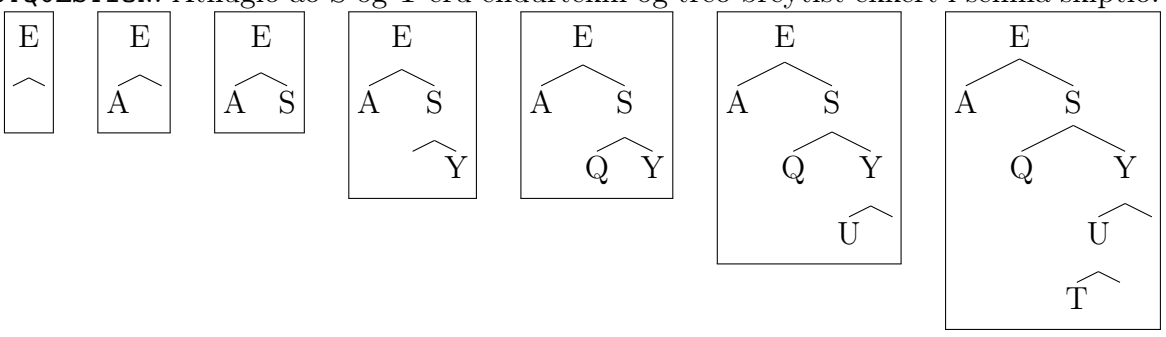
4.3.13

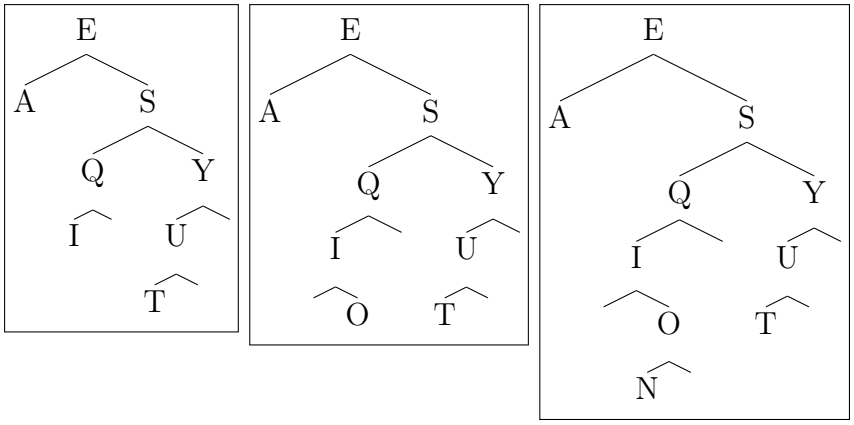
Við breytum `Evaluate` forritunu á bls. 572 í bók og látum `vals` vera stafla af strengjum. Í stað þess að reikna út úr segðunum sem við fáum búum við þess í stað til strenginn sem inniheldur segðina með svigum á réttum stað. Þessu gildi er síðan skellt upp á staflan rétt eins og útkoman var sett efst áður.

```
import java.util.Stack;

public class EvaluateMissing {
    public static void main(String[] args) {
        Stack<String> ops  = new Stack<String>();
        Stack<String> vals = new Stack<String>();

        while (!StdIn.isEmpty()) {
            String s = StdIn.readString();
            if (s.equals("+"))    ops.push(s);
            else if (s.equals("-"))    ops.push(s);
            else if (s.equals("*"))    ops.push(s);
            else if (s.equals("/"))    ops.push(s);
            else if (s.equals("sqrt")) ops.push(s);
            else if (s.equals(")")) {
                String op = ops.pop();
                String v2 = vals.pop();
                String v1 = vals.pop();
                String ex = "( " + v1 + " " + op + " " + v2 + " )";
                vals.push(ex);
            }
            else vals.push(s);
        }
        StdOut.println(vals.pop());
    }
}
```

4.4.8

Eftirfarandi myndir sýna stöðu trésins eftir hverja innsetningu af stöfunum `EASYQUESTION`. Athugið að S og T eru endurtekin og tréð breytist ekkert í seinna skiptið.





4.4.9

Í þessu dæmi vitum við ekki hvernig tvíleitartréð lítur út, til að útiloka leitarrunur höldum við utan um það bil sem hvert hluttré samsvarar. Ef við leitum að 363 og efsti hnúturinn er 2 þá vitum við að við þurfum að leita í hægra hluttrénu og allir hnútar þar eru á bilinu [3-1000]. Ef við förum eitt skref í viðbót og sjáum hnút með 252 þurfum við að fara hægra megin og þá þrengist bilið [253-1000]. Í þriðja skrefi er hnúturinn 401 skoðaður og við förum vinstra megin, þá er bilið [253-400] o.s.frv þar til við finnum 363.

(a) Bilin eftir hvern hnút eru 2: [3-1000], 252: [253-1000], 401: [253-400], 398: [253-397], 330: [331-337], 363 lykill fundinn.

(b) 399: [0-398], 387: [0-386], 216: [217-386], 266: [267-386], 382: [267-381], 278: [279-381], 363 lykill fundinn.

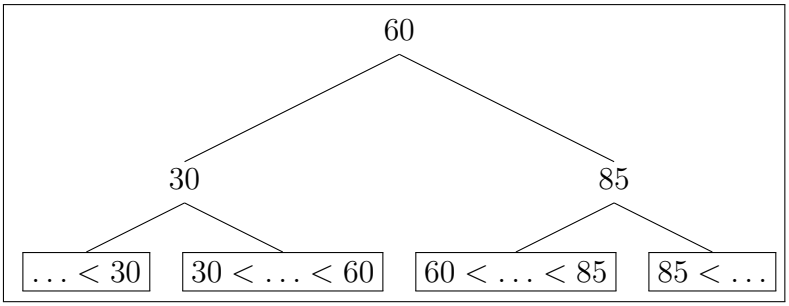
(c) 3: [4-1000], 923: [4-922], 220: [221-922], 911: [221-910], 244: [245-910], 898: [245-897], 258: [259-897], 362: [363-897], 363 lykill fundinn.

(d) 4: [5-1000], 924: [5-923], 278: [279-923], 347: [348-923], 621: [348-620], 299: Þessi lykill getur ekki verið í trénu þar sem hann er utan þess bils sem allir lyklar þurfa að vera á í hluttrénu.

(e) 5: [6-1000], 925: [6-924], 202: [203-924], 910: [203-923], 245: [246-922], 363 lykill fundinn

4.4.10

Þar sem tvíleitartré með hæð k getur í mesta lagi haft $2^k - 1$ hnúta og $31 = 2^5 - 1$ sjáum við að tréð verður að vera fullt og jafnað. Því hljóta að vera 15 lyklar vinstra megin við rót og 15 lyklar hægra megin, rótin verður því að vera miðgildið í listanum sem er 60. Eins verður vinstra hluttréð að vera fullt og rótin þar miðgildið af stökunum sem eru minni en 60, sem er 30 og í hægra hluttrénu við rótina verður að vera 85. Tréð lítur þá út nokkurn vegin svona



Verkefni

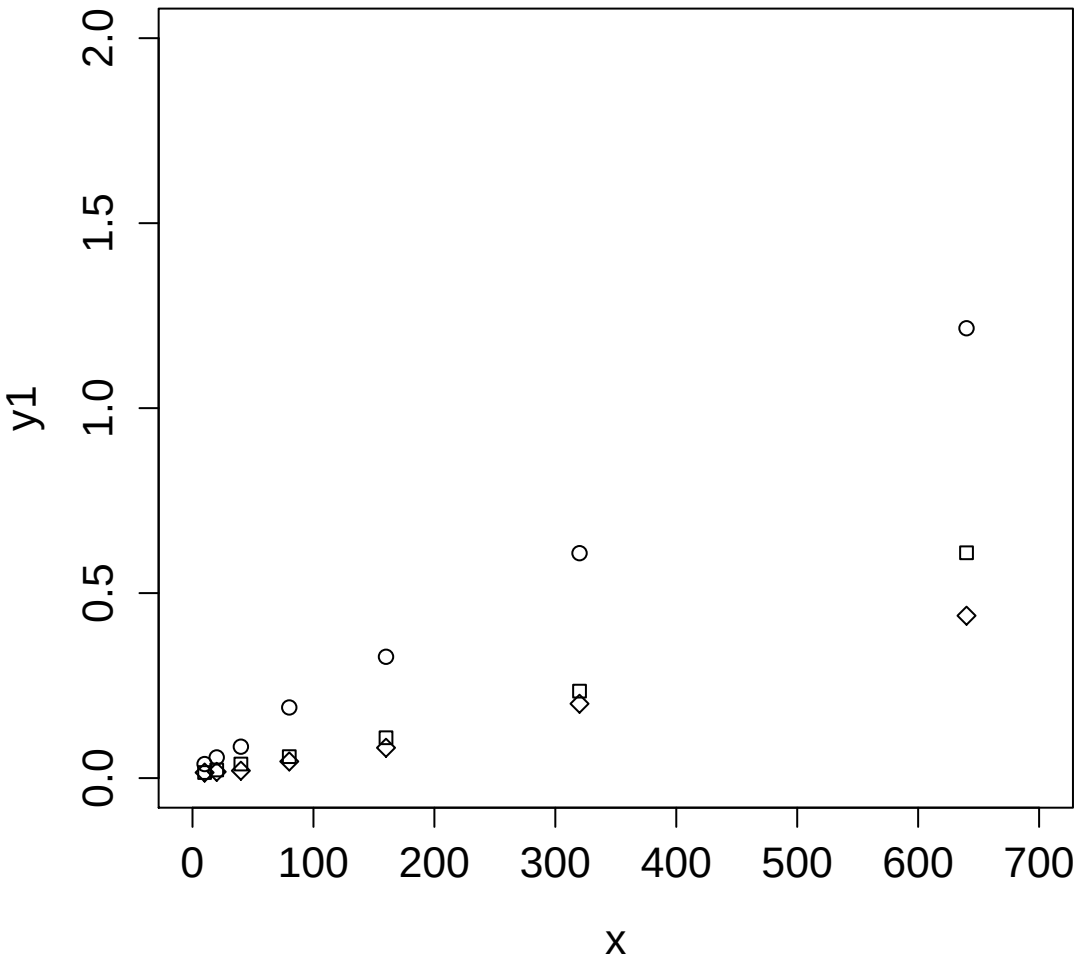
```
public class HashST<Key, Value> {
    private int N = 0;
    private Key[] keys;
    private Value[] vals;
    // Fastayrðing gagna: N >= 0, keys.length == vals.length.
    // N stök eru í töflunni og N <= 0.9*keys.length. Plássið i í töflunni
    // er tómt ef keys[i] == null og þá verður vals[i] == null líka að gilda.
    // Stakið (key,val) geymt í keys[i] og  val[i] þar
    // sem i == key.hashCode() % keys.length. Ef ekki er
    // pláss til að geyma stakið í sæti i skal það vera
    // í sæti i+1, i+2, ... (mod keys.length). Ef stakið er geymt
    // í sæti j !=i þá má ekkert af plássunum i,i+1,...,j-1 vera tómt.

    // Notkun: x = new HashST<K,V>()
    // Fyrir:  ekkert
    // Eftir:  x er tóm hakkatafla
    public HashST() {
        this(16);
    }

    // Notkun: x = new HashST<K,V>(size)
    // Fyrir:  size > 0
    // Eftir:  x er tóm hakkatafla sem rúmar a.m.k. size stök
    @SuppressWarnings({"unchecked"})
    public HashST(int initCapacity) {
        keys = (Key[]) new Object[initCapacity];
        vals = (Value[]) new Object[initCapacity];
    }

    // Notkun: c = x.isEmpty()
    // Fyrir:  ekkert
    // Eftir:  c er true ef engin stök eru í x
    public boolean isEmpty() {return N==0;}

    // Notkun: s = x.size()
    // Fyrir:  ekkert
    // Eftir:  s er fjöldi staka í x
```

```
public int size() {return N;}

// Notkun: c = x.contains(key)
// Fyrir:  ekkert
// Eftir:  c er satt ef lykillinn key er í x
public boolean contains(Key key) {
    return get(key) != null;
}

// Notkun: x.resize(size)
// Fyrir:  size > x.size()
// Eftir:  x rúmar size stök og inniheldur öll
//         gömlu stökin
@SuppressWarnings({"unchecked"})
private void resize(int SIZE) {
    Key[] temp_keys = this.keys;
    Value[] temp_vals = this.vals;

    this.keys = (Key[]) new Object[SIZE];
    this.vals = (Value[]) new Object[SIZE];
    this.N = 0;

    for (int i = 0; i < temp_keys.length; i++) {
        if (temp_keys[i] != null) {
            this.put(temp_keys[i],temp_vals[i]);
        }
    }
}

// Notkun: oldval = x.put(key,val)
// Fyrir:  ekkert
// Eftir:  ef key er í töflunni er gildið val sett í stað
//         gamla gildisins, annars er nýju staki bætti við
//         í töfluna með lykilinn key og gildið val. Ef gamalt
//         var í töflunni fyrir lykilinn key skal skila gamla gildinu
//         en annars skila null.
public Value put(Key key, Value val) {
    if (N > 0.9 * keys.length) {
        resize(2*this.keys.length);
    }

    int i;
    Value old = null;

    int h = hashval(key) % keys.length;
    for (i = h; keys[i] != null; i = (i+1) % keys.length) {
      if (keys[i].equals(key)) {
        old = vals[i];
```

```
            break;
          }
        }
        keys[i] = key;
        vals[i] = val;
        N++;
        return old;
    }

    // Notkun: v = x.get(key)
    // Fyrir:  ekkert
    // Eftir:  ef stakið (key,val) er í töflunni x þá verður v=val
    //         annars er v = null
    public Value get(Key key) {
      int h = hashval(key) % keys.length;
      for (int i = h; keys[i] != null; i = (i+1) % keys.length) {
        if (keys[i].equals(key)) {
          return vals[i];
        }
      }
      return null;
    }

    // Notkun: h = x.hashval(key)
  // Fyrir: ekkert
  // Eftir: h>=0 og h er "slembin" heiltala
  private int hashval(Key key) {
    int h = (key.hashCode() & Integer.MAX_VALUE);
    h ^= (h>>>20) ^ (h>>>12);
    h ^= (h>>>7) ^ (h>>>4);
    return h;
  }
}
```

1. Put aðferðin byrjar á stað h í töflunni og skoðar lykilinn í sæti h. Ef lykillinn er null, er sætið laust og við setjum lykillinn þar inn. Ef eitthvað er fyrir berum við saman við key með `keys[i].equals(key)`, ef lyklarnir eru eins er gildinu breitt og hætt í fallinu, annars er kíkt á næsta sæti. Þetta er endurtekið þar til við finnum lykilinn eða autt pláss í töflunni. Get virkar á svipaðan hátt, nema breytir ekki lyklunum og gildunum í töflunni.

2. Grafið hér að neðan sýnir tíma fyrir innsetningu á N stökum og leit fyrir $10 * N$ stök. `TreeMap` eru hringir, `HashMap` er kassar og `HashST` er tíglar



Við sjáum að í öllum tilfellum er tíminn línulegur en `HashST` er þó hraðvirkara en hinar tvær útfærslurnar.

3. Fyrir get eru framkvæmdar í a) besta tilfelli $O(1)$ aðgerðir þegar við þurfum ekki að leita lengi að réttum stað, b) $O(N)$ í versta tilfelli, t.d. þegar fyrsti helmingur töflunnar er fullur og við setjum inn stak sem á að fara í sæti 0, þá þarf að leita í gegnum helminginn af töflunni til að finna autt pláss. c) að meðaltali eru 1 sæti af hverjum 10 autt (miðað við minna en 90% nýtingu) og því þarf aðeins að fara 10 sæti að meðaltali til að finna autt pláss, meðaltals tíminn er því $O(1)$.

   Ef hakkafallið er gott og dreifir lyklunum vel þá er afar ólíklegt að við lendum í versta tilfelli í b). Þá er versta tilfellis tíminn í réttu hlutfalli við stærsta bilið í töflunni þar sem engir auðir lyklar eru. Það má sýna fram á að lengd þessa bils er $O(\log N)$ með líkum sem stefna á 1 þegar N stefnir á óendanlegt, t.d. eru líkurnar $\geq 1 - \frac{1}{N^2}$ ef fastinn við $\log_2 N$ er nógu stór.

4. Nákvæmlega sama svar og fyrir get.