Tehtävän V.1 ratkaisuehdotus
Tietorakenteet, syksy 2003

Matti Nykänen

5. joulukuuta 2003

1 Satelliitit

Muunnetaan luennoilla luonnosteltua toteutusta seuraavaksi:

Korvataan puusolmun p kentät $p.\mathrm{key}$ ja $p.\mathrm{item}$ kentällä $p.\mathrm{satellite}$ joka on osoitin solmua p vastaavaan *satelliittisolmuun* q.

Tähän satelliittisolmuun q sijoitetaan kentät $q.\mathrm{key}$ ja $q.\mathrm{item}$ jotka vastaavat puusolmussa p olleita saman nimisiä kenttiä.

Satelliittisolmu q luodaan ja tuhotaan yhtä aikaa puusolmun p kanssa.

Tämä muunnos yksinkertaistaa kekokäytön vaatimia muutoksia niihin punamustien puiden käsittelyalgoritmeihin, jotka luennoilla esiteltiin alun perin hakupuukäyttöön.

2 Kahvat

Luvun 1 satelliittisolmut yksinkertaistavat myös kahvojen toteutusta: kahva alkioon a on yksinkertaisesti viite siihen satelliittisolmuun joka sisältää alkion a.

Tätä käyttöä varten lisätään satelliittisolmuun q vielä seuraavat kentät:

- $q.\mathrm{back}$ on osoitin takaisin siihen puusolmuun p jonka satelliittisolmu q on.

  Siis yhteys puusolmun p ja sen satelliittisolmun q välillä on *kaksisuuntainen*.

- $q.\mathrm{root}$ on viite siihen paikkaan, jossa pidetään yllä osoitinta tämän punamustan puun nykyiseen juureen.

  Tämä sama tieto tarvitaan myös silloin, kun toteutetaan kahvoja hakupuun solmuihin — sellaisiakin tarvitaan joissakin algoritmeissa.

Nämä lisäkentät ovat ne parametrit, joita tarvitaan kutsuttaessa punamustan puun perusoperaatioita kahvan kautta: juuriosoitinviite $q.\mathrm{root}$ ja solmuosoitin $q.\mathrm{back}$. Esimerkki tällaisesta kutsusta on luku 16.

3 Satelliittien vaihto

Luentokalvoilla vaihdettiin kokonaisia puusolmuja keskenään, tässä muunnoksessa vaihdetaan vain niiden satelliitteja.

Hakupuukäytössä sama puusolmu ja satelliitti pysyvät aina yhdessä.

Siksi luentojen hakupuussa ei tarvittu erillisiä satelliiteja.

Kekokäytössä korvataan kekotaulukkopaikkojen keskinäiset vaihdot vastaavien satelliittien vaihdoilla.

Luvuissa 4 ja 5 toteutetaan kekojärjestyksen ylläpitoalgoritmit tällä periaattella.

Vaihtoalgoritmi saa puusolmut p ja q, ja vaihtaa näiden solmujen satelliitit keskenään.

procedure SwapSatellites(p, q: **solmuosoitin**)
Vaihda kenttien $p.\,\text{satellite}\,.\,\text{back}$ ja $q.\,\text{satellite}\,.\,\text{back}$ sisällöt keskenään**;**
Vaihda kenttien $p.\,\text{satellite}$ ja $q.\,\text{satellite}$ sisällöt keskenään**.**

4 Keko-ominaisuuden ylläpito

procedure Heapify(i: **solmuosoitin**)
1: l := $i.\,\text{left}$**;**
2: r := $i.\,\text{right}$**;**
3: **if** $l \neq \text{NULL}$ **and** $l.\,\text{satellite}\,.\,\text{key} > i.\,\text{satellite}\,.\,\text{key}$ **then**
4: $j := l$
5: **else**
6: $j := i$
7: **end if;**
8: **if** $r \neq \text{NULL}$ **and** $r.\,\text{satellite}\,.\,\text{key} > j.\,\text{satellite}\,.\,\text{key}$ **then**
9: $j := r$
10: **end if;**
11: **if** $j \neq i$ **then**
12: SwapSatellites(i, j)**;**
13: Heapify(j)
14: **end if.**

5 Avaimen nosto

procedure ShiftUp(i: **solmuosoitin**)
1: **if** $i.\,\text{parent} \neq \text{NULL}$ **and** $i.\,\text{satellite}\,.\,\text{key} > i.\,\text{parent}\,.\,\text{satellite}\,.\,\text{key}$ **then**
2: SwapSatellites($i, i.\,\text{parent}$)**;**
3: ShiftUp($i.\,\text{parent}$)
4: **end if.**

6 Alkion nosto keossa sen kahvasta

Eli Dijkstran algoritmin tarvitsema operaatio.

procedure HandleShiftUp(h: **satelliittiosoitin**, a: **avainarvo**)

```
1: h.key := a;
2: ShiftUp(h.back).
```

7 Kierto vasemmalle

Kekokäytön ainoa lisäys hakupuukäyttöön on uusi viimeinen rivi 16, joka palauttaa kekoehdon.

Luvun 1 satelliittimuunnos takaa, että rivin 16 jälkeenkin puun solmut ovat siten, kuin kutsuja niiden olettaa olevan.

procedure LeftRotate(r: **juuriosoitinviite**, x: **solmuosoitin**)

```
1: y := x.right;
2: x.right := y.left;
3: if y.left ≠ NULL then
4:     y.left.parent := x
5: end if;
6: y.parent := x.parent;
7: if x.parent = NULL then
8:     r := y
9: else if x = x.parent.left then
10:    x.parent.left := y
11: else
12:    x.parent.right := y
13: end if;
14: y.left := x;
15: x.parent := y;
16: Heapify(x.parent).
```

8 Kierto oikealle

Vasen/oikea-symmetrisesti luvusta 7.

procedure RightRotate(r: **juuriosoitinviite**, x: **solmuosoitin**)

```
1: y := x.left;
2: x.left := y.right;
3: if y.right ≠ NULL then
4:     y.right.parent := x
5: end if;
6: y.parent := x.parent;
7: if x.parent = NULL then
8:     r := y
9: else if x = x.parent.right then
```

```
10:    x.parent.right := y
11: else
12:    x.parent.left := y
13: end if;
14: y.right := x;
15: x.parent := y;
16: Heapify(x.parent).
```

9 Solmun lisäys punamustaan hakupuuhun

Tätä algoritmia ei oikeastaan tarvita kekokäytössä. Se on mukana, jotta nähdään luvun 1 satelliittimuunnoksen vaikutukset.

procedure RbTreeInsert(r: **juuriosoitinviite**, a: **avainarvo**, b: **sisältö**)

```
 1: if r = NULL then
 2:    r := new juurisolmu jossa r.left := NULL, r.right := NULL, r.parent := NULL,
       r.color := BLACK ja r.satellite := new satelliittisolmu jossa r.satellite.key := a,
       r.satellite.item := b, r.satellite.back := juuri luotu juurisolmu r ja r.satellite.root := r
 3: else
 4:    x := r;
 5:    y := x.parent (eli aluksi NULL);
 6:    while x ≠ NULL and a ≠ x.satellite.key do
 7:       y := x;
 8:       if a < x.satellite.key then
 9:          x := x.left
10:       else
11:          x := x.right
12:       end if
13:    end while;
14:    if x ≠ NULL then
15:       x.satellite.item := b
16:    else
17:       t := new lehtisolmu jossa t.left := NULL, t.right := NULL, t.parent := y, t.color := RED
          ja t.satellite := new satelliittisolmu jossa t.satellite.key := a, t.satellite.item := b,
          t.satellite.back := juuri luotu lehtisolmu t ja t.satellite.root := r;
18:       if a < y.satellite.key then
19:          y.left := t
20:       else
21:          y.right := t
22:       end if;
23:       RbInsertFixup(r, t)
24:    end if
25: end if.
```

10 Solmun lisäys kekoon

Luvun 9 algoritmi seuraavin muutoksin:

- Enää ei tarvitse etsiä ”oikeaa” lisäyspaikkaa rivien 4–16 tapaan.
- Sen sijaan voidaan edetä mielivaltaiseen lehteen ja lisätä sinne. Tässä algoritmissa lisätään aina puun vasempaan laitaan.
- Rivillä 8 palautetaan kekoehto takaisin voimaan.
- Lopuksi palautetaan juuri luotu luvun 1 satelliitti jotta kutsuja voi käyttää sitä luvun 2 kahvana juuri lisättyyn alkioon.

function HeapInsert(r: **juuriosoitinviite**, a: **avainarvo**, b: **sisältö**): **satelliittisolmuviite**

1: **if** $r = \text{NULL}$ **then**
2: r := **new** juurisolmu jossa $r.\text{left}$:= NULL, $r.\text{right}$:= NULL, $r.\text{parent}$:= NULL, $r.\text{color}$:= BLACK ja $r.\text{satellite}$:= **new** satelliittisolmu jossa $r.\text{satellite}.\text{key}$:= a, $r.\text{satellite}.\text{item}$:= b, $r.\text{satellite}.\text{back}$:= juuri luotu juurisolmu r ja $r.\text{satellite}.\text{root}$:= r;
3: **return** $r.\text{satellite}$
4: **else**
5: y := First(r);
6: t := **new** lehtisolmu jossa $t.\text{left}$:= NULL, $t.\text{right}$:= NULL, $t.\text{parent}$:= y, $t.\text{color}$:= RED ja $t.\text{satellite}$:= **new** satelliittisolmu jossa $t.\text{satellite}.\text{key}$:= a, $t.\text{satellite}.\text{item}$:= b, $t.\text{satellite}.\text{back}$:= juuri luotu lehtisolmu t ja $t.\text{satellite}.\text{root}$:= r;
7: $y.\text{left}$:= t;
8: ShiftUp(t);
9: RbInsertFixup(r, t);
10: **return** $t.\text{satellite}$
11: **end if.**

11 Tasapainotus solmun lisäyksen jälkeen

Muuten suoraan luentokalvoilta, mutta nyt myös vasen/oiken-symmetrinen haara auki kirjoitettuna.

procedure RbInsertFixup(r: **juuriosoitinviite**, z: **solmuosoitin**)

1: **while** $z.\text{parent} \neq \text{NULL}$ **and** $z.\text{parent}.\text{color} = \text{RED}$ **do**
2: **if** $z.\text{parent} = z.\text{parent}.\text{parent}.\text{left}$ **then**
3: y := $z.\text{parent}.\text{parent}.\text{right}$;
4: **if** $y \neq \text{NULL}$ **and** $y.\text{color} = \text{RED}$ **then**
5: $y.\text{color}$:= BLACK;
6: $z.\text{parent}.\text{color}$:= BLACK;
7: $z.\text{parent}.\text{parent}.\text{color}$:= RED;
8: z := $z.\text{parent}.\text{parent}$
9: **else**
10: **if** $z = z.\text{parent}.\text{right}$ **then**

```
11:             z := z.parent;
12:             LeftRotate(r, z)
13:           end if;
14:           z.parent.color := BLACK;
15:           z.parent.parent.color := RED
16:           RightRotate(r, z.parent.parent)
17:         end if
18:       else
19:         y := z.parent.parent.left;
20:         if y ≠ NULL and y.color = RED then
21:           y.color := BLACK;
22:           z.parent.color := BLACK;
23:           z.parent.parent.color := RED;
24:           z := z.parent.parent
25:         else
26:           if z = z.parent.left then
27:             z := z.parent;
28:             RightRotate(r, z)
29:           end if;
30:           z.parent.color := BLACK;
31:           z.parent.parent.color := RED
32:           LeftRotate(r, z.parent.parent)
33:         end if
34:       end if
35: end while;
36: r.color := BLACK.
```

12 Ensimmäisen solmun haku

Suoraan luentokalvoilta.

function First(r: **solmuosoitin**): **solmuosoitin**

```
1: if r ≠ NULL then
2:   while r.left ≠ NULL do
3:     r := r.left
4:   end while;
5: end if;
6: return r.
```

13 Seuraavan solmun haku

Suoraan luentokalvoilta.

function Next(i: **solmuosoitin**): **solmuosoitin**

```
1: if i.right ≠ NULL then
2:   return First(i.right)
3: else
```

```
 4:    p := i.parent;
 5:    while p ≠ NULL and i = p.right do
 6:       i := p;
 7:       p := i.parent
 8:    end while
 9:    return p
10: end if.
```

14 Solmun poisto puusta

Kuten luentokalvoilla seuraavin muutoksin:

- Rivillä 5 poistettava z vaihdetaan sen seuraajaan y vaihtamalla niiden satelliitit luvusta 1.
- Jos tämä vaihto tehtiin, niin rivillä 23 palautetaan kekoehto voimaan.

  Tämä on ainoa ero satelliittipohjaisen hakupuun ja keon välillä.
- Lopuksi riveillä 28–29 vapautetaan sekä solmun y että sen nykyisen satelliitin varaama muisti.

procedure RbTreeDelete(r: **juuriosoitinviite**, z: **solmuviite**)

```
 1: if z.left = NULL or z.right = NULL then
 2:    y := z
 3: else
 4:    y := next(z);
 5:    SwapSatellites(y, z)
 6: end if;
 7: if y.left ≠ NULL then
 8:    x := y.left
 9: else
10:    x := y.right
11: end if;
12: if x ≠ NULL then
13:    x.parent := y.parent
14: end if;
15: if y.parent = NULL then
16:    r := x
17: else if y = y.parent.left then
18:    y.parent.left := x
19: else
20:    y.parent.right := x
21: end if;
22: if y ≠ z then
23:    Heapify(z)
24: end if;
25: if y.color = BLACK then
26:    RbDeleteFixup(r, x, y.parent)
```

```
27: end if;
28: delete y.satellite;
29: delete y.
```

15 Tasapainotus solmun poiston jälkeen

Muuten suoraan luentokalvoilta, mutta nyt myös vasen/oiken-symmetrinen haara auki kirjoitettuna.

procedure RbDeleteFixup(r: **juuriosoitinviite**, x: **solmuviite**, p: **solmuviite**)

```
1: while p ≠ NULL and (x = NULL or x.color = BLACK) do
2:   if x = p.left then
3:     w := p.right;
4:     if w.color = RED then
5:       w.color := BLACK;
6:       p.color := RED;
7:       LeftRotate(r, p);
8:       w := p.right
9:     end if;
10:    if w.left.color = BLACK and w.right.color = BLACK then
11:      w.color := RED;
12:      x := p;
13:      p := x.parent
14:    else
15:      if w.right.color = BLACK then
16:        w.left.color := BLACK;
17:        w.color := RED;
18:        RightRotate(r, w)
19:        w := p.right
20:      end if;
21:      w.color := p.color;
22:      p.color := BLACK;
23:      w right.color := BLACK;
24:      LeftRotate(r, p);
25:      x := r;
26:      p := x.parent
27:    end if
28:  else
29:    w := p.left;
30:    if w.color = RED then
31:      w.color := BLACK;
32:      p.color := RED;
33:      RightRotate(r, p);
34:      w := p.left
35:    end if;
36:    if w.right.color = BLACK and w.left.color = BLACK then
37:      w.color := RED;
```

```
38:         x := p;
39:         p := x.parent
40:       else
41:         if w.left.color = BLACK then
42:           w.right.color := BLACK;
43:           w.color := RED;
44:           LeftRotate(r, w)
45:           w := p.left
46:         end if;
47:         w.color := p.color;
48:         p.color := BLACK;
49:         w left.color := BLACK;
50:         RightRotate(r, p);
51:         x := r;
52:         p := x.parent
53:       end if
54:     end if
55: end while;
56: if x ≠ NULL then
57:     x.color := BLACK
58: end if.
```

16 Kahvan osoittaman solmun poisto keosta

Annetun kahvan h osoittaman alkion poisto keostaan on suoraan vastaavan puuoperaation kutsu.

procedure HandleDelete(h: **satelliittiosoitin**)

```
1: RbTreeDelete(h.root, h.back).
```

17 Onko keko tyhjä?

Eli onko puu tyhjä?

function HeapIsEmpty(r: **juuriosoitinviite**): **boolean**

```
1: return r ≠ NULL.
```

18 Kahva epätyhjän keon päällimmäiseen alkioon

function HeapMaximum(r: **juuriosoitinviite**): **satelliittiosoitin**

```
1: return r.satellite.
```

19 Päällimmäisen alkion poisto epätyhjästä keosta

procedure HeapDeleteMaximum(r: **juuriosoitinviite**)

1: HandleDelete(HeapMaximum(r)).