

# Hrúgur

Bergur Snorrason

17. febrúar 2023

# Hrúgur

- ▶ Rótartvíundatré sem uppfyllir að sérhver nóða er stærri en börnin sín er sagt uppfylla *hrúguskilyrðið*.
- ▶ Við köllum slík tré *hrúgur* (e. *heap*).
- ▶ Hrúgur eru heppilega auðveldar í útfærslu.
- ▶ Við geymum tréð sem fylki og eina erfiðið er að viðhalda hrúguskilyrðinu.

- ▶ Þegar við geymum tréð sem fylki notum við eina af tveimur aðferðum.
- ▶ Sú fyrri:
  - ▶ Rótin er í staki 1 í fylkinu.
  - ▶ Vinstra barn staks  $i$  er stak  $2 \cdot i$ .
  - ▶ Hægra barn staks  $i$  er stak  $2 \cdot i + 1$ .
  - ▶ Foreldri staks  $i$  er stakið  $\lfloor i/2 \rfloor$ .
- ▶ Sú seinni:
  - ▶ Rótin er í staki 0 í fylkinu.
  - ▶ Vinstra barn staks  $i$  er stak  $2 \cdot i + 1$ .
  - ▶ Hægra barn staks  $i$  er stak  $2 \cdot i + 2$ .
  - ▶ Foreldri staks  $i$  er stakið  $\lfloor (i - 1)/2 \rfloor$ .
- ▶ Takið eftir í fyrri aðferðinni notum við ekki stak 0 í fylkinu.
- ▶ Þetta má sjá sem bæði kost og galla.
- ▶ Það stak má nota til að geyma, til dæmis, stærðina á trénu.

- ▶ Bein afleiðing af hrúguskilyrðinu er að rótin er stærsta stakið í trénu.
- ▶ Við getum því alltaf fengið skjótan aðgang að stærsta stakinu í trénu.
- ▶ Algengt er að *forgangsbiðraðir* (e. *priority queues*) séu útfærðar með hrúgum.

```

8 #define PARENT(i) ((i)/2)
9 #define LEFT(i)    ((i)*2)
10 #define RIGHT(i)   ((i)*2 + 1)
11 int h[MAXN + 1];
12 void swap(int* x, int* y) { int t = *x; *x = *y; *y = t; }
13 void fix_down(int i)
14 {
15     int mx = i;
16     if (RIGHT(i) <= h[0] && h[mx] < h[RIGHT(i)]) mx = RIGHT(i);
17     if (LEFT(i) <= h[0] && h[mx] < h[LEFT(i)]) mx = LEFT(i);
18     if (mx != i) swap(&h[i], &h[mx]), fix_down(mx);
19 }
20
21 void fix_up(int i)
22 {
23     if (i == 1 || h[i] <= h[PARENT(i)]) return;
24     swap(&h[i], &h[PARENT(i)]), fix_up(PARENT(i));
25 }
26
27 void pop()
28 {
29     h[1] = h[h[0]--];
30     fix_down(1);
31 }
32
33 void push(int x)
34 {
35     h[++h[0]] = x;
36     fix_up(h[0]);
37 }
38
39 int peek() { return h[1]; }
40 int size() { return h[0]; }
41 void init() { h[0] = 0; }

```

- ▶ Gerum ráð fyrir að við séum með  $n$  stök í hrúgunni.
- ▶ Þá er hæð trésins  $\mathcal{O}(\log n)$ .
- ▶ Þar sem `pop()` þarf aðeins að ferðast einu sinni niður að laufi er tímaflækjan  $\mathcal{O}(\log n)$ .
- ▶ Þar sem `push(...)` þarf aðeins að ferðast einu sinni upp að rót er tímaflækjan  $\mathcal{O}(\log n)$ .
- ▶ Nú þarf `peek()` ekki að gera annað en að lesa fremsta stakið í fylki svo tímaflækjan er  $\mathcal{O}(1)$ .
- ▶ Við getum útfært þetta með ítrun, í stað þess að nota endurkvæmni.

```

13 void pop()
14 {
15     h[1] = h[h[0]--];
16     for (int i = 1, mx = i; ; swap(&h[i], &h[mx]), i = mx)
17     {
18         if (RIGHT(i) <= h[0] && h[mx] < h[RIGHT(i)]) mx = RIGHT(i);
19         if (LEFT(i) <= h[0] && h[mx] < h[LEFT(i)]) mx = LEFT(i);
20         if (i == mx) return;
21     }
22 }
23
24 void push(int x)
25 {
26     h[++h[0]] = x;
27     for (int i = h[0]; i > 1 && h[i] > h[PARENT(i)]; i = PARENT(i))
28         swap(&h[i], &h[PARENT(i)]);
29 }
30
31 int peek() { return h[1]; }
32 int size() { return h[0]; }
33 void init() { h[0] = 0; }

```

