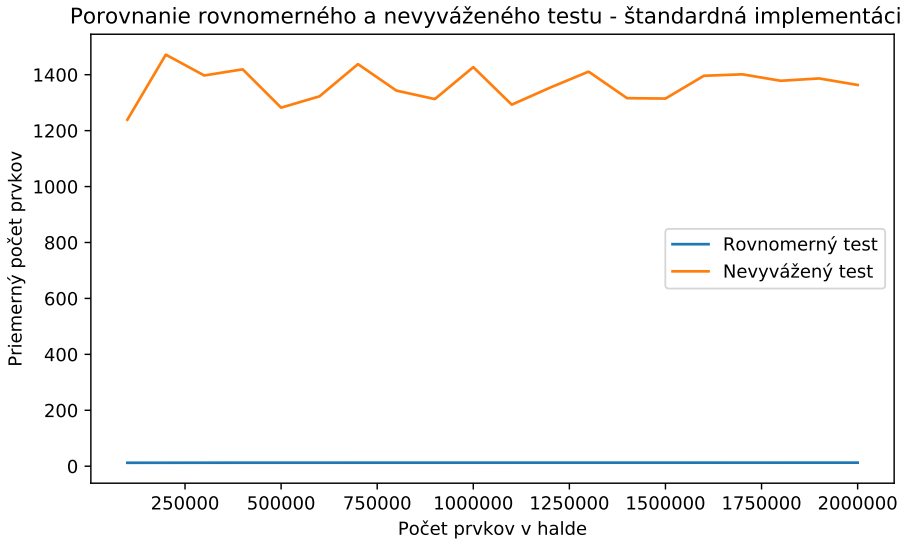
1 Fibbonacciho halda

1.1 Porovnanie počtu krokov operácie delete min rovnomerného a nevyváženého testu pri štandardnej implementácii

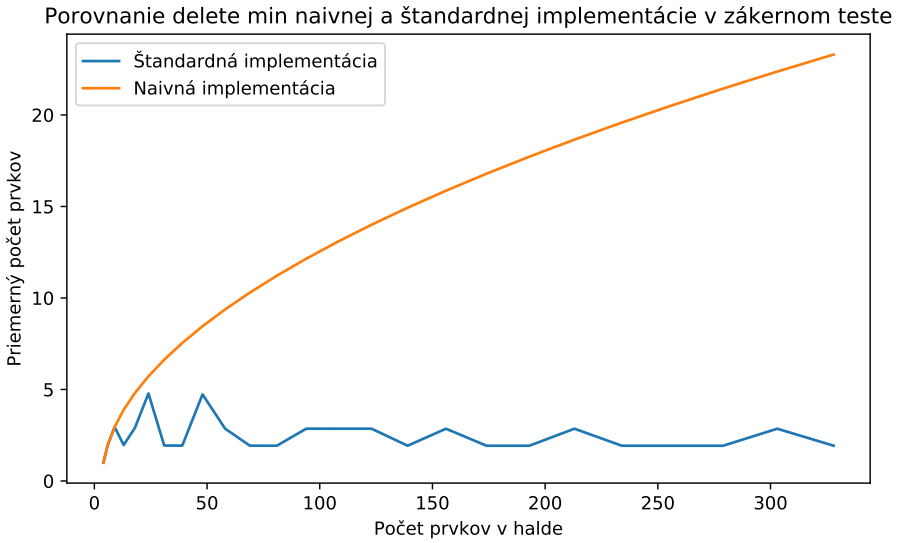


Obe krivky vyzerajú na grafe konštantne vzhľadom na počet prvkov v halde. Pri nevyváženom teste majú hodnoty príliš veľký rozptyl aby sa o nich niečo dalo definitívne povedať, čo je spôsobené tým, že pri generovaní používame náhodu. Pri bližšom skúmaní priemerného počtu krokov v rovnomernom teste môžeme povedať, že počet krokov rastie.

Z teórie vieme, že amortizovaná časová zložitosť operácie delete min je $\mathcal{O}(\log n)$, čo na grafe nevieme pozorovať. Dôvodom je pravdepodobne veľká konštanta skrytá v asymptotickej zložitosti.

V nevyváženom teste je taký veľký priemer, pretože sa volá operácie delete min menej často a teda vždy potrebuje veľa krokov na konsolidáciu haldy (medzitým sa vykonali iné operácie).

1.2 Porovnanie implementácii v zákernom teste



Tento test demonštruje to, že zložitosť operácie delete min v najhoršom prípade pri naivnej halde je $\mathcal{O}(\sqrt{n})$, a nepomôže nám ani amortizácia.

Pri naivnej verzii dokážeme istou postupnosťou operácii dosiahnuť, že halda bude obsahovať (rádovo) $\sqrt{n}$ stromov hĺbky 2, každý iného rádu. Prvý strom pozostáva iba z jedného vrchola, druhý je vrchol s jedným synom, tretí je s 2 synmi, atd. Posledný má rádovo $\sqrt{n}$ synov. V takejto halde dokážeme opakovať ľubovoľne veľa operácii `Insert x` a `DeleteMin`, kde x je nejaký kľúč menší ako všetky ostatné v halde. Delete min musí nájsť nové minimum a preto musí prejsť až $\sqrt{n}$ synov.

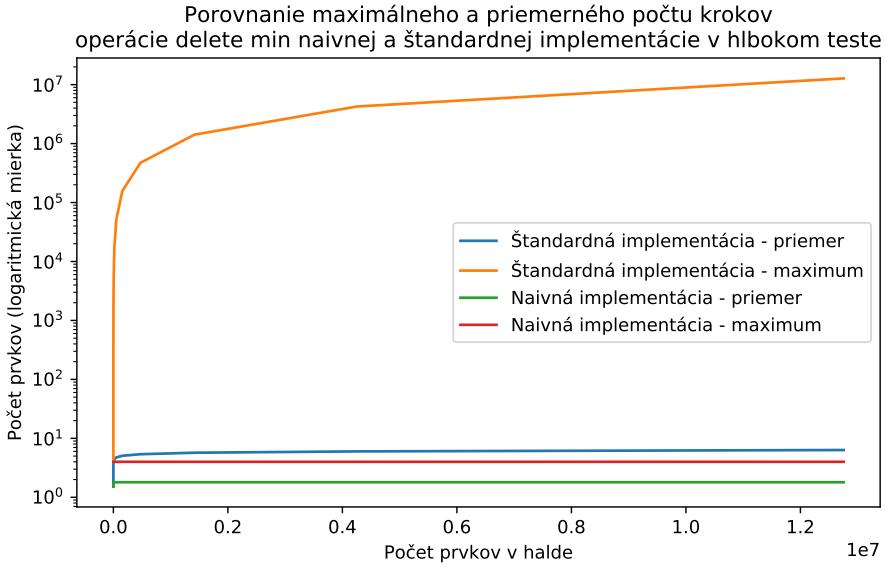
V implementácii však nepočítame kroky pri hľadaní minima. Generátor najskôr uvedie haldu do stavu opísaného vyššie a potom posiela halde veľa krát za sebou túto postupnosť krokov:

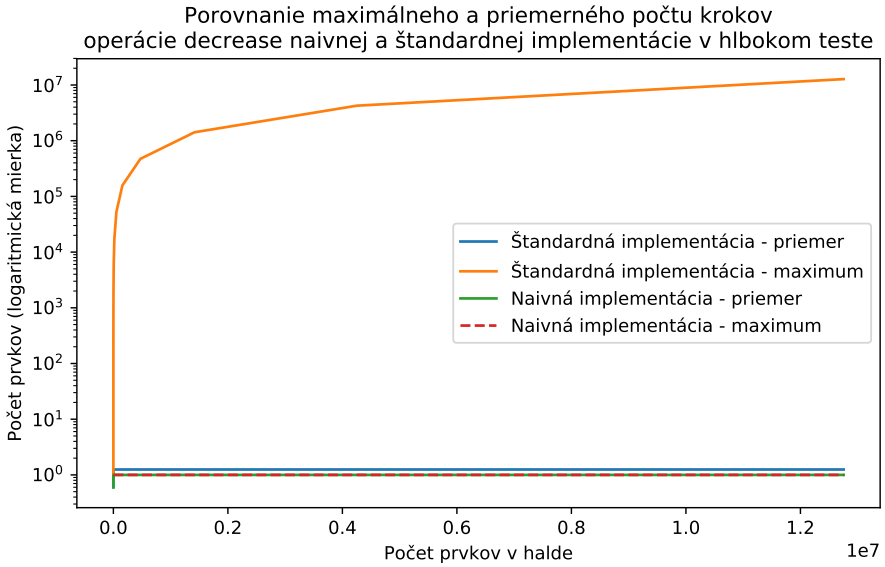
1. `Insert x`
2. `Insert x-1`
3. `DeleteMin`
4. `DeleteMin`

kde x a x-1 sú dva najmenšie prvky v halde. Prvé `DeleteMin` spôsobí konsolidáciu v halde, výsledkom ktorej je jeden strom, ktorý má ako deti korene stromov pred konsolidáciou. Druhé `DeleteMin` odstráni koreň jediného stromu, a vráti haldu do pôvodného stavu. Prvé `DeleteMin` spravilo (rádovo) $\sqrt{n}$ krokov.

V štandardnej implementácii tento postup nevytvorí príliš veľa plytkých stromov, a teda pozorujeme konštantný počet krokov. Pri opakovaní vyššie vymenovaných krokov nastane konsolidácia pri prvom `DeleteMin`, ktorá trvá, kým sa nenájde pre novo zlúčený strom miesto. Mohla by trvať až log n krokov, no zjavne sa vždy nájde miesto medzi stromami malého rádu.

1.3 Porovnanie maximálneho a priemerného počtu krokov operácii naivnej a štandardnej implementácie v hlbokom teste





Z grafov vidíme, že maximum krokov oboch operácii štandardnej implementácie rastie lineárne (logaritmickú mierku som použil, by bolo vidieť rozdiel medzi zvyšnými krivkami). Ostatné závislosti sú konštantné.

Operácie v hlbokom teste postavia z vrcholov haldy cestu, v ktorej je každý vrchol označený. Na konci zavolá operácie `Decrease` na najhlbší prvok cesty a `DeleteMin`. Počas stavania cesty potrebuje každá operácia konštantný počet krokov.

V štandardnej implementácii `Decrease` spôsobí, že celá cesta sa rozpadne na stromy rádu 0, na čo potrebujeme (približne) n krokov. Následný `DeleteMin` spôsobí konsolidáciu, ktorá trvá (približne) n krokov. To teda potvrdzuje worst case zložitosť operácii: $\mathcal{O}(n)$. V priemere však má `Decrease` zaručenú zložitosť $\mathcal{O}(1)$, čo potvrdzuje graf. `DeleteMin` má v tomto špeciálnom teste konštantnú amortizovanú zložitosť.

V naivnej implementácii `Decrease` jednoducho odtrhne posledný vrchol cesty a `DeleteMin` odstráni buď tento vrchol alebo koreň cesty, na čo potrebujeme konštantný počet krokov. Decrease naivnej implementácie potrebuje vždy práve jeden krok.