Fibonacciho halda

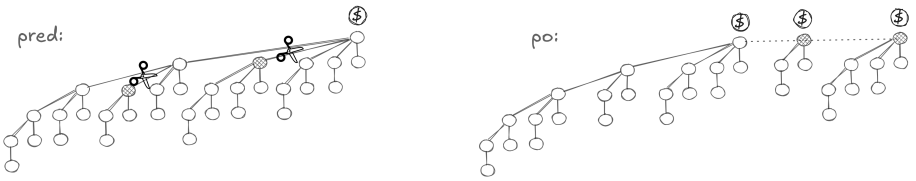
V predchádzajúcej kapitole sme videli, že binomiálna halda a jej lenivá verzia umožňujú veľmi efektívne operácie spájania a vkladania. Otázka teda znie: Dá sa to lepšie? Vieme aj `decrease-key` urýchliť na konštantný čas?

Na prvý pohľad to vyzerá beznádejne: stromy môžu mať logaritmickú hĺbku, takže ak chceme znížený kľúč „prebublať" až do koreňa, bude to trvať pomerne dlho. Napriek tomu odpoveď znie: áno, dá sa to.

Mohli by sme si hneď ukázať finálne riešenie, no bolo by to trochu neuspokojivé. Prečo práve takto? Prečo tak komplikovane? Ako na to vôbec niekto prišiel? Aby sme pochopili motiváciu, ukážeme si najskôr jeden *neúspešný pokus*. Je poučný a zároveň nám pomôže pochopiť, aké prekážky treba pri operácii `decrease-key` prekonať.

Takto to nejde

Prvý nápad, ako sa vyhnúť pomalému bublaniu, je naplno prijať lenivý prístup a odkladanie práce na neskôr. Ak v nejakom vrchole x znížime kľúč a ten sa stane menším než kľúč jeho rodiča, poruší sa haldovité usporiadanie. S vrcholom x teda musíme niečo urobiť. Čo keby sme ho jednoducho *odstrihli* aj s celým jeho podstromom a presunuli ho medzi korene? Pri najbližšom upratovaní sa stromy s rovnakým rádom opäť pospájú do väčších a štruktúra sa obnoví.

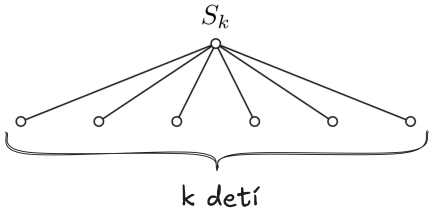


Obr. 1: Prvý nápad: Pri `decrease-key` vrchol jednoducho odstrihneme a pridáme medzi korene. Na obrázku je príklad `decrease-key` na dva označené vrcholy.

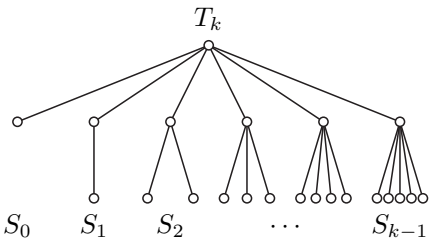
Jednoduché. Za túto operáciu si vyúčtujeme 2$. Jeden dolár pokryje samotnú prácu (odrezanie vrcholu a jeho pripojenie medzi korene) a druhý dolár uložíme novému koreňu na účet, aby sme zachovali invariant. Zvyšok dôkazu pre operáciu `extract-min` bude prebiehať úplne rovnako ako v predchádzajúcej kapitole (viď amortizovaná analýza lenivej binomiálnej haldy).

Je tento dôkaz správny? Ak nie, kde presne sa to pokazí? A ak táto štruktúra nedosahuje želaný výkon, aká je skutočná zložitosť `extract-min`? Skúste sa nad tým na chvíľu zamyslieť, než si pozriete riešenie.

V prvom rade si všimnime, že po odrezaní nejakého podstromu už zvyšok pôvodného stromu nespĺňa definíciu binomiálneho stromu. Predstavme si extrémny príklad: začneme s binomiálnym stromom rádu k a koreňu postupne



(a) Strom S_k definujeme ako koreň a pod ním k vrcholov.



(b) Strom z T_k rádu k sa skladá z koreňa a jeho deti sú $S_0, S_1, \ldots, S_{k-1}$. Tento strom má rádovo $\Theta(k^2)$ vrcholov.

(pomocou operácie `decrease-key`) odstrihneme všetkých vnukov. Výsledkom je strom S_k (pozri obr. 2a).

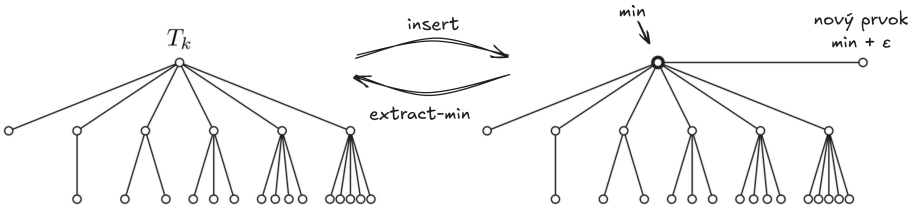
Zatiaľ čo v binomiálnej halde mal strom rádu k výšku k a obsahoval 2^k vrcholov, strom S_k má výšku 1 a obsahuje len $k + 1$ vrcholov. Tento problém pretrvá aj po upratаní. V binomiálnej halde spájame stromy rovnakého rádu a výsledkom je strom dvojnásobnej veľkosti. Po rezoch však veľkosť stromov už nezodpovedá ich rádu a zlučovanie preto nezaručuje zdvojnásobenie veľkosti. Dôsledky sú zásadné: nedokážeme garantovať, že maximálny rád bude logaritmický, že každý strom má len logaritmický počet detí, alebo že štruktúra sa skladá len z logaritmického počtu stromov.

„A nedá sa ten dôkaz nejako zachrániť?“ povie si optimista. Možno keby sme upravili invarianty, šetrili inak alebo úplne zmenili analýzu, dalo by sa *nejakým spôsobom* dokázať, že zložitosť `extract-min` je logaritmická. Odpoveď je však opäť nie.

Tu je konkrétny protipríklad (pozri obr. 3): Nech T_k je strom, ktorý sa skladá z koreňa a jeho deťmi sú stromy $S_0, S_1, \ldots, S_{k-1}$ (pozri obr. 2b). Takýto strom T_k má $\Theta(k^2)$ vrcholov a vieme ho zostrojiť pomocou $O(k^3)$ operácií. Všimnime si, že ak pripojíme S_k pod koreň T_k, dostaneme strom T_{k+1} s rádom o jeden vyšším.

Uvažujme teraz nasledujúcu postupnosť operácií: Začneme so stromom T_k, kde koreň má hodnotu 0 a všetky ostatné vrcholy majú kľúče väčšie než, povedzme, milión. Do haldy vložíme nový kľúč, ktorý je len o ε väčší než hodnota v koreni, ale zároveň menší než hodnoty všetkých jeho synov. Výsledkom je halda zložená zo stromu T_k a jedného nového vrcholu, teda T_0.

Ak teraz zavoláme `extract-min`, najmenším prvkom je koreň T_k. Ten odstránime a jeho synov $S_0, S_1, \ldots, S_{k-1}$ vložíme medzi korene. Pri následnom upratovaní sa postupne pospájajú všetky tieto stromy s T_0. Kľúč sme zvolili tak, aby T_0 vyhral pri každom porovnaní, takže stromy $S_0, \ldots, S_{k-1}$ sa jeden po druhom pripoja pod neho. Výsledkom je séria spojení: $T_0 + S_0 \to T_1$, $T_1 + S_1 \to T_2$, a tak ďalej, až kým opäť nedostaneme strom T_k s rovnakým tvarom, len s koreňom o trochu väčším.



Obr. 3: Protipríklad, ktorý ukazuje, že pri naivnom odstrihávaní vrcholov trvá operácia `extract-min` aspoň $\Omega(\sqrt{n})$, a to aj v amortizovanom zmysle.

Dostali sme sa späť do pôvodného stavu, ale algoritmus pritom vykonal $\Theta(k) = \Theta(\sqrt{n})$ operácií, čo je priveľa. Túto dvojicu operácií môžeme opakovať ľubovoľne dlho, a preto je amortizovaná zložitosť `insert` + `extract-min` aspoň $\Omega(\sqrt{n})$. Ak by sme chceli dokázať, že je to menej, museli by sme zvyšnú prácu do $\sqrt{n}$ zakaždým doplácať z našetrených dolárov. To sa však nedá robiť donekonečna – skôr či neskôr by sme zbankrotovali.

Mimochodom, dá sa pomerne jednoducho ukázať, že $\sqrt{n}$ je nielen dolný, ale aj horný odhad pre zložitosť operácie `extract-min`. Dostávame tak haldu, ktorá podporuje všetky operácie v čase $O(1)$, až na `extract-min`, ktorá má zložitosť $O(\sqrt{n})$.

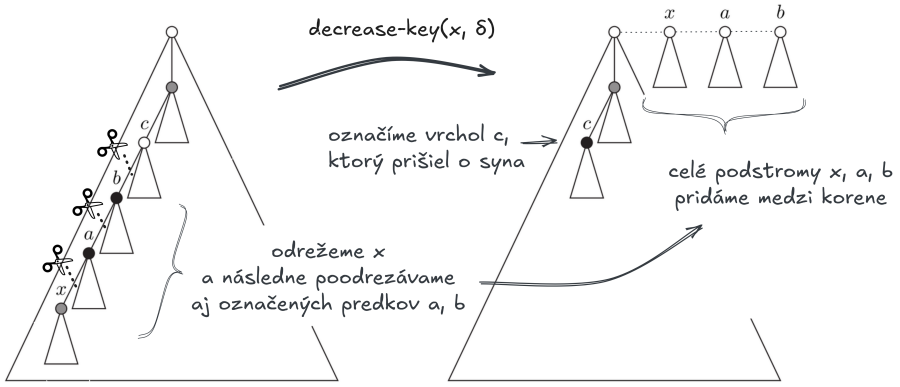
Poučenie, ktoré si z tohto neúspešného pokuku odnášame, je jasné: nechceme mať stromy s vysokým rádom, ktoré pritom obsahujú len málo vrcholov. Ak má byť štruktúra rýchla, musíme zabezpečiť, aby každý strom obsahoval *exponenciálne veľa vrcholov vzhľadom na svoj rád*. Inak povedané: pre strom rádu k potrebujeme, aby mal aspoň c^k vrcholov pre nejakú konštantu $c > 1$. Z toho priamo plynie, že maximálny rád stromu v halde s n prvkami je nanajvýš $\log_c n$ a po upratаní sa celá halda skladá z najviac $\log_c n$ stromov. Práve tieto vlastnosti sú kľúčové v dôkaze, že operácia `extract-min` má v lenivej binomiálnej halde logaritmickú amortizovanú zložitosť.

Skúsme to teda z opačnej strany: ku každému vrcholu si budeme pamätať veľkosť jeho podstromu a *rád* zadefinujeme ako logaritmus tejto veľkosti (zaokrúhlený nadol). Týmto zaručíme, že veľkosť podstromu rastie exponenciálne s rádom jeho koreňa.

Má to však háčik: pri každom `decrease-key`, keď nejaký vrchol odstrihneme, by sme museli aktualizovať veľkosti všetkých predkov na ceste ku koreňu. To je znova *logaritmicky veľa práce* – presne toľko, ako keby sme vrchol prebublávali nahor.

Riešenie: Kaskádové rezy

Prvý známy spôsob, ako zvládnuť operáciu `decrease-key` v konštantnom a zároveň `extract-min` v logaritmickom čase, navrhli Michael L. Fredman a Robert



Obr. 4: Priklad operácie `decrease-key`(x, δ). Čierne vrcholy a a b majú poznačené, že už prišli o jedného syna, biele vrcholy majú všetkých synov a v prípade šedých vrcholov na obrázku je to nepodstatné pre túto operáciu. Odstrihnutie x spôsobí kaskádu: a stratí druhého syna, takže ho odrežeme a tým pádom aj b stratí druhého syna a aj tento podstrom odrežeme. Zastavíme sa až na prvom bielom vrchole c, ktorý, keďže prišiel o syna, ofarbíme na čierno.

E. Tarjan v roku 1984. Ich technika dostala názov *kaskádové rezy* (*cascading cuts*).

Odvtedy vzniklo viacero ďalších prístupov, ktoré dosahujú rovnaký cieľ (Driscoll et al. 1988; Kaplan a Robert Endre Tarjan 2008; Elmasry 2010; Haeupler, Sen a Robert E Tarjan 2011; Chan 2013; Kaplan, Robert E Tarjan a Zwick 2014; Hansen et al. 2017; Elmasry, Jensen a Katajainen 2008; Brodal 1996; Brodal a Okasaki 1996; Brodal, Lagogiannis a Robert E Tarjan 2012). My si však ukážeme práve pôvodný nápad Fredmana a Tarjana.

Definujeme *rád* vrcholu ako *počet jeho detí*. Myšlienka je nasledovná: Pri `decrease-key` vrchol odstrihneme a pripojíme do zoznamu koreňov. Zároveň si však pre jeho otca poznačíme, že sme mu odrezali syna. Budeme dodržiavať invariant, že každý vrchol okrem koreňov má najviac jedného odrezaného syna. Každý vrchol (okrem koreňa), ktorému odrežeme dvoch synov, odrežeme tiež a pripojíme ho ku koreňom.

Jedna operácia `decrease-key` teda môže spustiť celú kaskádu rezov (pozri obr. 4). V najhoršom prípade môže takáto kaskáda trvať až logaritmický čas (úmerný hĺbke vrcholu). Ukážeme si však, že amortizovaná zložitosť ostáva konštantná.

Následne si ukážeme, že vďaka invariantu, že každý vrchol prišiel najviac o jedného syna, zostávajú stromy dostatočne „husté“ a ich veľkosť bude exponenciálna vzhľadom na počet detí koreňa.

Na prvý pohľad to môže pôsobiť paradoxne: pri operáciach `decrease-key`

odrezávame podstromy, pritom sa chceme vyhnúť stromom s veľkým rádom a malým počtom vrcholov. Riešením má byť odrezať *ešte viac* podstromov?

Keď odrežeme veľa podstromov, strom sa prirodzene zmenší, a preto by mal mať aj menší rád. Potrebujeme teda, aby sa tento „signál“ – informácia o odstránených vrcholoch – prenášal smerom nahor, ku koreňu. A práve na to slúžia kaskádové rezy: ak odstránime príliš veľa vrcholov, budeme nútení postupne odstrániť aj syna samotného koreňa. Keďže *rád* vrcholu definujeme ako *počet jeho detí*, rád koreňa (a teda aj celého stromu) sa primerane zníži.

To je aspoň intuícia, ktorá stojí za kaskádovými rezmi. Teraz sa už môžeme pozrieť na formálne dôkazy, ktoré ukážu, že operácia `decrease-key` je skutočne konštantná z amortizovaného hľadiska a že `extract-min` si zachováva logaritmickú zložitosť.

Veta 1. *Operácia* `decrease-key` *s kaskádovými rezmi trvá* $O(1)$ *amortizovane.*

■ **Dôkaz #1.** Použijeme účtovnícku metódu. Za každú operáciu si vypýtame 4$.

Invariant. Každý označený vrchol (teda taký, ktorému už bol odstrihnutý jeden syn) má nasporené 2$ na zaplatenie prípadného budúceho kaskádového rezu. Navyše tak ako v predošlej kapitole, každý koreň má odložený 1$, ktorým prispieva na upratovanie.

Vyúčtovanie. Pri jednej operácii `decrease-key`:

- 1$ použijeme na samotnú prácu ($O(1)$ úkonov: odstrihnutie, pripojenie medzi korene, označenie otca),
- 1$ odložíme na účet tohto nového koreňa,
- 2$ pošleme jeho otcovi.

Ak mal otec doteraz všetkých synov, teraz má na účte 2$, invariant zostáva zachovaný a operácia končí.

Ak mu už jeden syn chýbal, mal uložené 2$ a práve dostal ďalšie 2$ – spolu 4$. Tie mu postačia na zaplatenie vlastného rezu:

- 1$ použije na samotnú prácu,
- 1$ si nechá ako nový koreň,
- a zvyšné 2$ posunie *svojmu* otcovi.

Takto pokračuje celá kaskáda: Každý označený vrchol si odrezanie pokryje z vlastných nasporených peňazí a zároveň pošle 2$ vyššie. Proces pokračuje, až kým nenarazíme na koreň, alebo na neoznačený vrchol, ktorý iba označíme a prispejeme mu 2$. V oboch prípadoch sa operácia končí a invarianty ostanú zachované. □

Ten istý dôkaz pomocou potenciálovej metódy:

■ **Dôkaz #2.** Zvoľme si potenciál haldy ako

$$\Phi(H) = \textit{počet koreňov} + 2 \times \textit{počet označených vrcholov}.$$

Ak počas jednej operácie odrežeme k vrcholov, reálna práca zaberie $O(k)$ času. Pozrime sa, ako sa zmení potenciál:

$+\,k$	pretože pribudne k nových koreňov,
$-\,2 \times (k-1)$	pretože okrem prvého vrcholu boli všetky ostatné označené a po odrezaní už nie sú,
$+\,2$	pretože na konci kaskády jeden označený vrchol pribudol.

Celkovo potenciál klesne o $k-4$, takže

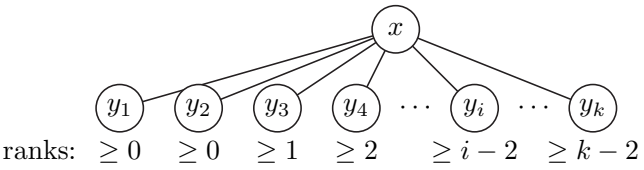
$$T_{\text{amort}} = T_{\text{skutočný}} + \Delta\Phi = O(k) - (k-4) = O(1). \qquad \square$$

Z hľadiska amortizovanej analýzy sú teda kasakádové rezy „zadarmo“ – vrcholy dokážu zaplatiť samy za seba.

Veľkosť stromov

Ostáva nám ešte ukázať, že pri takomto urezávaní bude mať koreň rádu k stále exponenciálne veľa vrcholov v závislosti od k.

Lema 1. *Nech vrchol x má synov $y_1, \ldots, y_m$, pričom ich označíme v poradí, v akom sa pripojili ku x. Potom platí rád$(y_i) \geq i-2$, teda tretí syn má aspoň jedného syna, štvrtý syn aspoň dvoch, a tak ďalej.*

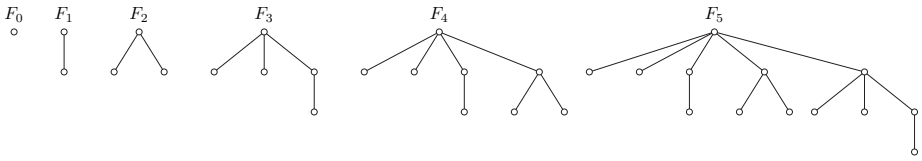


■ **Dôkaz.** Vráťme sa naspäť do momentu, keď sme pripájali y_i pod x. V tom čase mal vrchol x už aspoň $i-1$ detí (konkrétne $y_1, \ldots, y_{i-1}$; prípadne aj ďalšie, ktoré sa neskôr odrezali, ale minimálne tieto zostali zachované).

Pri spájaní však vždy spájame iba stromy s rovnakým rádom, a teda *y_i v tom čase muselo mať tiež aspoň $i-1$ detí.* Odvtedy mu mohol ubudnúť nanajvýš jeden syn (v opačnom prípade by sme ho takisto odrezali) a preto v súčasnosti platí rád$(y_i) \geq i-2$. $\square$

Túto lemu môžeme aplikovať rekurzívne na všetky vrcholy stromu. Zamyslime sa, aká je najmenšia možná veľkosť stromu, ktorého koreň má rád n (t. j. má n detí). Takýto minimálny strom označme F_n.

Pre $n = 0, 1, 2$ je F_n len koreň a jeho n synov. A čo F_3? Podľa lemy musí mať tretí syn rád aspoň 1, teda má aspoň jedného syna.



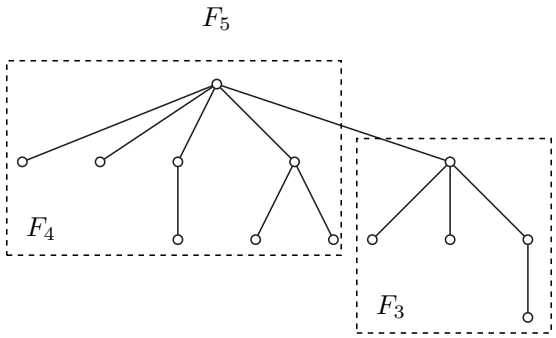
V strome F_4 má koreň štyroch synov; tretí má rád aspoň 1 a štvrtý aspoň 2.
V strome F_5 pribudne piaty syn s rádom aspoň 3.
Našťastie, najmenšie stromy rádov 1, 2 a 3 sme už identifikovali.

Spočítajme počet vrcholov:

strom	F_0	F_1	F_2	F_3	F_4	F_5
#vrcholov	1	2	3	5	8	13

Hmmmm... Náhoda? Nemyslím si.

Veľkosť stromu F_n je $(n+1)$-vé Fibonacciho číslo (odtiaľ pochádza názov Fibonacciho halda). Dôvod je jedoduchý: Najmenší strom rádu n sa skladá z koreňa a jeho n detí. Prvých $n-1$ synov vyzerá rovnako ako v strome F_{n-1}. Posledný, n-ty syn, má podľa lemy rád aspoň $n-2$, a teda najmenší taký strom je práve F_{n-2}. Z toho vyplýva, že strom F_n vznikne pripojením F_{n-2} ku F_{n-1}, čo je rovnaká rekurencia, akú spĺňajú Fibonacciho čísla.



Fibonacciho čísla rastú exponenciálne rýchlo. Asi najjednoduchšie to vidno z rekurencie

$$F_n = F_{n-1} + F_{n-2} \geq 2 \times F_{n-2},$$

čo znamená, že $F_n \geq 2^{n/2} = (\sqrt{2})^n$. Presný odhad je $F_n = \Theta(\phi^n)$, kde $\phi = (1+\sqrt{5})/2 \approx 1.618$ je hodnota zlatého rezu.

Odtiaľ vyplývajú nasledovné tvrdenia:

Veta 2. *Každý vrchol rádu k má pod sebou* exponenciálne veľa *vrcholov v závislosti od k; presnejšie, aspoň $\Omega(\phi^k)$ vrcholov. (Rád vrcholu je definovaný ako počet jeho detí.)*

Veta 3. *Vo Fibonacciho halde majú operácie nasledujúcu amortizovanú zložitosť:*

- *`insert`, `merge` a `decrease-key`* – $O(1)$,
- *`extract-min`* – $O(\log n)$.

■ **Dôkaz.** Veľkosť stromu rádu k je aspoň ϕ^k, takže maximálny možný rád je logaritmický (pri základe ϕ namiesto 2):

$$\log_\phi n \leq \lg n / \lg \phi \leq 1.45 \lg n.$$

Každý koreň má teda najviac $O(\log n)$ detí a po upratarní ostane najviac $O(\log n)$ stromov. Zvyšok argumentu je rovnaký ako pri lenivej binomiálnej halde (líšia sa len konštanty). □

Referencie

Brodal, Gerth Stølting (1996). „Worst-case efficient priority queues“. In: SODA.

Brodal, Gerth Stølting, George Lagogiannis a Robert E Tarjan (2012). „Strict fibonacci heaps“. In: *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, s. 1177–1184.

Brodal, Gerth Stølting a Chris Okasaki (1996). „Optimal purely functional priority queues“. In: *Journal of Functional Programming* 6.6, s. 839–857.

Driscoll, James R et al. (1988). „Relaxed heaps: An alternative to Fibonacci heaps with applications to parallel computation“. In: *Communications of the ACM* 31.11, s. 1343–1354.

Elmasry, Amr (2010). „The violation heap: A relaxed Fibonacci-like heap“. In: *International Computing and Combinatorics Conference*. Springer, s. 479–488.

Elmasry, Amr, Claus Jensen a Jyrki Katajainen (2008). „Two-tier relaxed heaps“. In: *Acta Informatica* 45.3, s. 193–210.

Haeupler, Bernhard, Siddhartha Sen a Robert E Tarjan (2011). „Rank-pairing heaps“. In: *SIAM Journal on Computing* 40.6, s. 1463–1485.

Hansen, Thomas Dueholm et al. (2017). „Hollow heaps“. In: *ACM Transactions on Algorithms (TALG)* 13.3, s. 1–27.

Chan, Timothy M (2013). „Quake heaps: A simple alternative to Fibonacci heaps“. In: *Space-Efficient Data Structures, Streams, and Algorithms: Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*. Springer, s. 27–32.

Kaplan, Haim, Robert E Tarjan a Uri Zwick (2014). „Fibonacci heaps revisited“. In: *arXiv preprint arXiv:1407.5750*.

Kaplan, Haim a Robert Endre Tarjan (2008). „Thin heaps, thick heaps“. In: *ACM Transactions on Algorithms (TALG)* 4.1, s. 1–14.