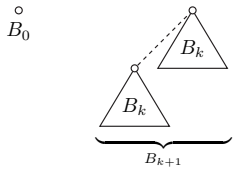
Binomiálna halda

Binomiálnu haldu vynašiel Jean Vuillemin v roku 1976. V praxi síce často prehráva s obyčajnou binárnou haldou (najmä kvôli väčším konštantám skrytým v O-notácii), no má jednu unikátnu schopnosť: dokáže efektívne zlúčiť dve haldy v logaritmickom čase, na rozdiel od klasickej binárnej haldy, ktorá na to potrebuje lineárny čas.

Pre nás sú binomiálne haldy zaujímavé najmä ako odrazový mostík na ceste k sofistikovanejším štruktúram, konkrétne k Fibonacciho halde, ku ktorej sa postupne prepracujeme v ďalšej kapitole.

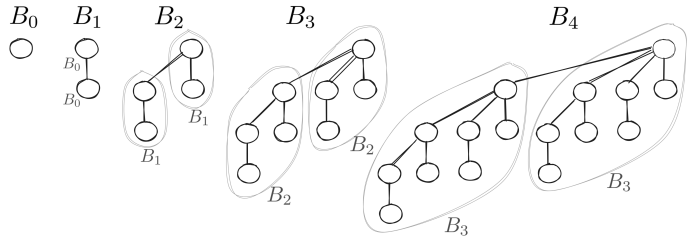
Binomiálne stromy

Základným stavebným prvkom binomiálnej haldy je *binomiálny strom*. Binomiálny strom rádu k (označujeme B_k) definujeme rekurzívne:



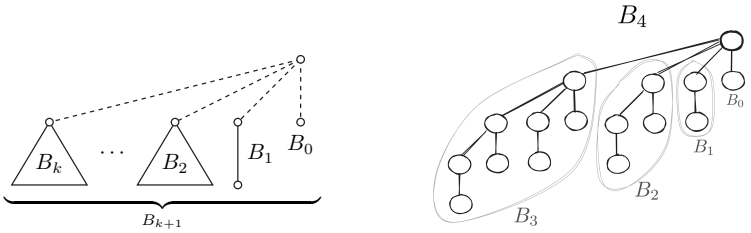
- B_0 je strom s jediným vrcholom,
- B_{k+1} vznikne tak, že zoberieme dva stromy B_k a jeden z nich pripojíme ako najľavejšie dieťa koreňa druhého.

Takto vyzerajú binomálne stromy rádu 0 až 4:



Z definície vieme hneď odvodiť niekoľko dôležitých vlastností: Binomiálny strom rádu k má

- presne 2^k vrcholov,
- výšku k,
- a deti koreňa tvoria (zľava doprava) stromy rádu $B_{k-1}, B_{k-2}, \ldots, B_0$.



V každom vrchole binomiálneho stromu je uložený práve jeden prvok haldy. Okrem toho budeme vyžadovať, aby bol každý binomiálny strom *haldovito usporiadaný*: kľúč vo vrchole je vždy menší alebo rovný kľúčom všetkých jeho detí (a teda aj všetkých potomkov). Z tohto *min-heap invariantu* vyplýva, že najmenší prvok celého stromu sa nachádza vždy v jeho koreni.

Štruktúra binomiálnej haldy

Binomiálna halda s n prvkami sa skladá z viacerých binomiálnych stromov. Ako presne? Každý binomiálny strom má veľkosť rovnú mocnine 2, potrebujeme teda z mocnín 2 „poskladať“ číslo n...

Vezmime napríklad $n = 41$. V binárnom zápise je to `101001`, pretože číslo 41 vieme jednoznačne rozložiť na súčet mocnín dvojky:

$$41 = 32 + 8 + 1 = 2^5 + 2^3 + 2^0.$$

Binomiálnu haldu so 41 vrcholmi preto môžeme poskladať zo stromov B_5, B_3 a B_0 (veľkostí 2^5, 2^3 a 2^0, spolu 41 vrcholov).

Podobne napríklad $n = 75$ má binárny zápis `1001011`, lebo

$$75 = 64 + 8 + 2 + 1 = 2^6 + 2^3 + 2^1 + 2^0.$$

Halda s $n = 75$ vrcholmi preto obsahuje stromy B_6, B_3, B_1 a B_0.

Vo všeobecnosti môžeme *binomiálnu haldu* definovať ako zoznam niekoľkých binomiálnych stromov B_i, pričom každý z nich má *iný rád* i. Vďaka tejto vlastnosti má binomiálna halda s n vrcholmi vždy pevne určený tvar, ktorý priamo zodpovedá binárnemu zápisu čísla n: ak je k-ty bit zápisu rovný `1`, halda obsahuje strom B_k; ak je rovný `0`, strom B_k sa v halde nenachádza.

Z tejto súvislosti vidíme, že binomiálna halda s n prvkami obsahuje najviac $\log_2 n$ stromov a ich rády sa pohybujú od 0 po $\lfloor \log_2 n \rfloor$. Keďže výška každého stromu sa rovná jeho rádu, aj výška stromov je nanajvyš logaritmická. To je dôležité, pretože všetky základné operácie s binomiálnou haldou majú zložitosť úmernú buď počtu stromov, alebo ich výške, teda v najhoršom prípade $O(\log n)$.

Spájanie háld

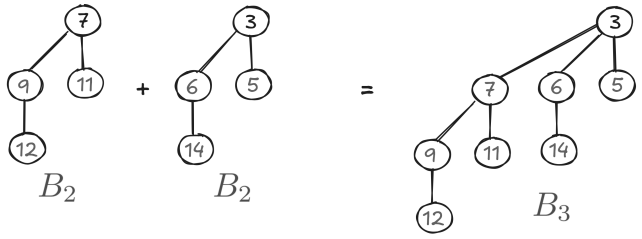
Spájanie dvoch binomiálnych háld prebieha veľmi podobne ako sčítanie čísel v dvojkovej sústave. Ukážme si to na príklade $41 + 75$:

$$\begin{array}{cccccccc} & & 1 & 0 & 1 & 0 & 0 & 1 \\ + & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ \hline & 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{array}$$

Začíname sprava: $1 + 1$ je 2, takže do výsledku zapíšeme 0 a prenesieme 1 do vyššieho rádu. Na ďalšej pozícii máme $0 + 1 +$ prenesená $1 = 2$, takže opäť zapíšeme 0 a prenesieme 1 ďalej. Takto pokračujeme, kým nesčítame všetky bity oboch čísel.

Pri spájaní binomiálnych háld postupujeme analogicky, len namiesto sčítania bitov porovnávame a prípadne spájame binomiálne stromy. Ak sa na určitej „pozícii“ stretnú dva stromy rovnakého rádu, zlúčime ich do jedného stromu o jeden rád väčšieho. Pri zlúčení porovnáme kľúče v koreňoch a väčší koreň zavesíme pod menší, čím zachováme min-heap invariant. Novovzniknutý strom sa posunie na ďalšiu pozíciu presne tak, ako sa pri binárnom sčítaní prenáša jednotka do vyššieho rádu.

Tu je príklad zlúčenia dvoch B_2 stromov do jedného B_3:



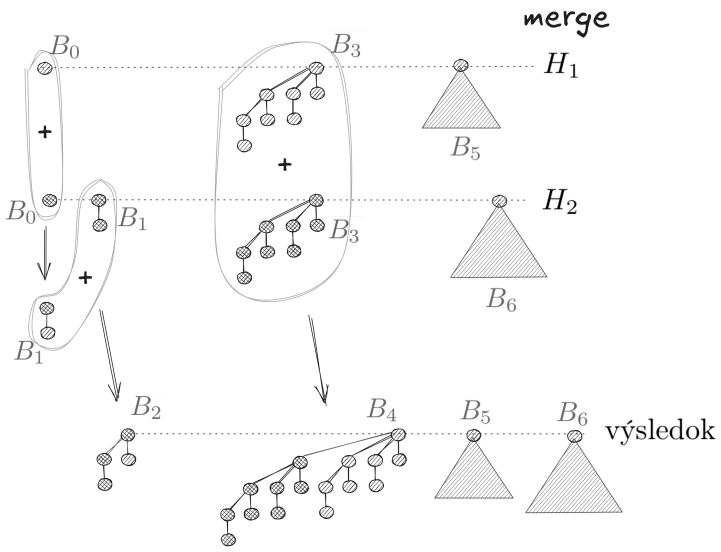
Keďže $3 < 7$, menší kľúč ostáva v koreni a vrchol 7 pripojíme pod neho. Táto operácia trvá len konštantný čas – vyžaduje len jedno porovnanie a úpravu niekoľkých smerníkov.

Vezmime ako príklad dve haldy veľkosti $n = 41$ a $n = 75$. Už vieme, že sa budú skladať z binomálnych stromov (B_5, B_3, B_0) a (B_6, B_3, B_1, B_0). Spájanie prebehne nasledovne (pozri tiež obr. 1):

`merge` H_1	–	B_5	–	B_3	–	–	B_0
a H_2	B_6	–	–	B_3	–	B_1	B_0
výsledok	B_6	B_5	B_4	–	B_2	–	–

Rád 0: Obe haldy obsahujú stromy $B_0 \rightarrow$ zlúčime ich na B_1 a prenesieme do vyššieho rádu,

Rád 1: B_1 z druhej haldy + prenesený $B_1 \rightarrow$ zlúčime na B_2 a prenesieme do vyššieho rádu,



Obr. 1: „Sčítanie“ binomiálnych háld $(B_5,\ B_3,\ B_0)$ a $(B_6,\ B_3,\ B_1,\ B_0)$. Výsledkom je $(B_6,\ B_5,\ B_4,\ B_2)$.

Rád 2: B_2 v pôvodných haldách nie je $\rightarrow$ vo výsledku ostane prenesený B_2,

Rád 3: obe haldy majú $B_3 \rightarrow$ zlúčime na B_4 a prenesieme

Rád 4: B_4 sa inde nenachádza $\rightarrow$ vo výsledku ostane prenesený B_4,

Rád 5: B_5 je len v prvej halde $\rightarrow$ ostáva vo výsledku,

Rád 6: B_6 je len v druhej halde $\rightarrow$ ostáva vo výsledku.

Zložitosť. Pri zlučovaní prechádzame iba zoznam koreňov oboch háld a tých je len logaritmicky veľa. Každý krok spájania (porovnanie dvoch koreňov a prípadné zlúčenie stromov rovnakého rádu) trvá pri vhodnej reprezentácii v pamäti konštantný čas (implementačné detaily prenecháme ako cvičenie cteným čitateľom). Celková časová zložitosť spájania je preto $O(\log n_1 + \log n_2)$, kde n_1 a n_2 sú veľkosti spájaných háld.

Ostatné operácie

Všetky ostatné operácie binomiálnej haldy už sú jednoduché a vieme ich realizovať pomocou spájania.

Vloženie prvku (insert). Na vloženie nového prvku x vytvoríme binomiálnu haldu obsahujúcu jediný vrchol (B_0 s prvkom x v koreni), a túto jednoprvkovú haldu zlúčime s pôvodnou. Časová zložitosť vloženia je $O(\log n)$.

Nájdenie minima (get-min). Najmenší prvok sa vždy nachádza v jednom z koreňov binomiálnych stromov. Stačí teda prejsť všetky korene a vybrať ten s najmenším kľúčom. Keďže koreňov je najviac $\log_2 n$, hľadanie trvá $O(\log n)$.

Efektívnejšia možnosť je udržiavať spolu s haldou aj smerník na koreň s minimom a pri operáciách, ktoré ho môžu zmeniť, smerník aktualizovať. Takto vieme nájsť minimum v konštantnom čase, keďže výsledok bude vždy predpočítaný.

Odstránenie minima (extract-min). Vezmeme koreň s minimálnym kľúčom a odstránime ho. Jeho deti sú rôzne binomiálne stromy nižších rádov – spolu teda tvoria haldu, ktorú následne jednoducho zlúčime s pôvodnou haldou (pozri obr. 2). Výsledná časová zložitosť je $O(\log n)$.

Zníženie kľúča (decrease-key). Ak chceme zmenšiť kľúč v niektorom vrchole, nahradíme ho menšou hodnotou a „prebubleme“ ho nahor (podobne ako v klasickej binárnej halde). To znamená, že opakovane vymieňame vrchol so svojím rodičom a stúpame nahor, až kým nenarazíme na rodiča s menším kľúčom alebo sa nedostaneme až do koreňa. Týmto obnovíme min-heap invariant.

Výška stromu rádu k je k, pričom najväčší možný rád v halde s n prvkami je $\lfloor \log_2 n \rfloor$. Preto je časová zložitosť operácie $O(\log n)$.

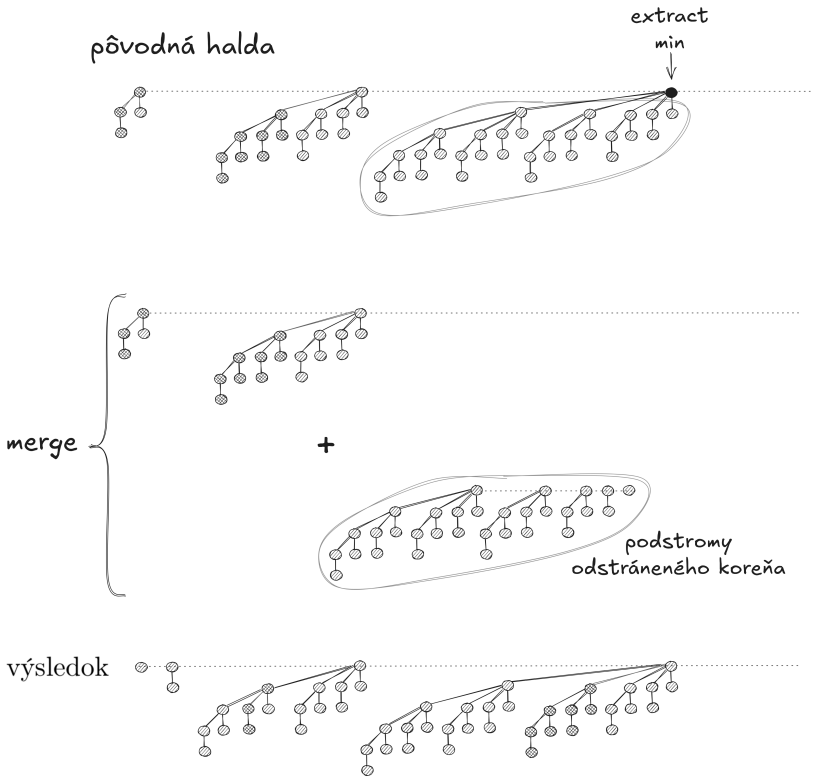
Zhrnutie

Operácia	Časová zložitosť
insert	$O(\log n)$
merge	$O(\log n_1 + \log n_2)$
get-min	$O(1)$
extract-min	$O(\log n)$
decrease-key	$O(\log n)$

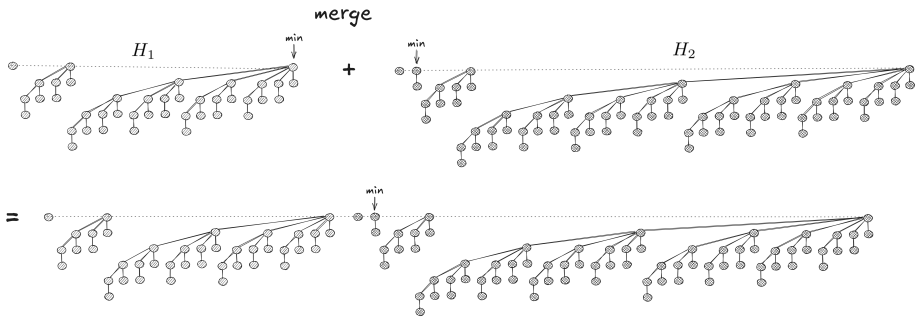
Lenivá binomiálna halda

Spájanie v logaritmickom čase je už samo osebe veľký pokrok. Natíska sa však otázka: nedalo by sa to ešte rýchlejšie? Na prvý pohľad to znie nereálne – zoznam koreňov má predsa logaritmickú dĺžku a pri spájaní ho musíme celý prejsť. Ukáže sa však, že ak si dovolíme trochu „porušiť poriadok“ a časť práce odložíme na neskôr, vieme `merge` (a teda aj `insert`) vykonať dokonca v *konštantnom amortizovanom čase*.

Základná myšlienka je jednoduchá: pri klasickej binomiálnej halde sú korene stromov pekne usporiadané a každý strom má *iný rád*. Pri spájaní vždy okamžite „upraceme“ všetky kolízie – ak sa stretnú dva stromy rovnakého rádu, zlúčime ich do jedného stromu s vyšším rádom.



Obr. 2: Odstránenie minima: koreň s najmenším kľúčom sa odstráni. Jeho podstromy tvoria samostatnú haldu, ktorú zlúčime s pôvodnou. Skúška správnosti: v dvojkovej sústave $\texttt{11100} - \texttt{1} = \texttt{11011}$, takže ak z $(B_4,\ B_3,\ B_2)$ odstránime jeden vrchol a upraceme, ostane nám $(B_4,\ B_3,\ B_1,\ B_0)$



Obr. 3: Spojenie dvoch lenivých háld: jednoducho spojíme dva zoznamy. Minimum vyberieme porovnaním predpočítaných minimím H_1 a H_2.

V *lenivej* verzii však na toto upratovanie nemáme čas. Pri spájaní jednoducho pripojíme zoznam koreňov druhej haldy k zoznamu prvej a upratovanie odložíme na neskôr. Ak haldy reprezentujeme ako *spájané zoznamy* binomiálnych stromov, celé spájanie sa obmedzí na nastavenie zopár smerníkov, čo zvládneme v konštantnom čase.

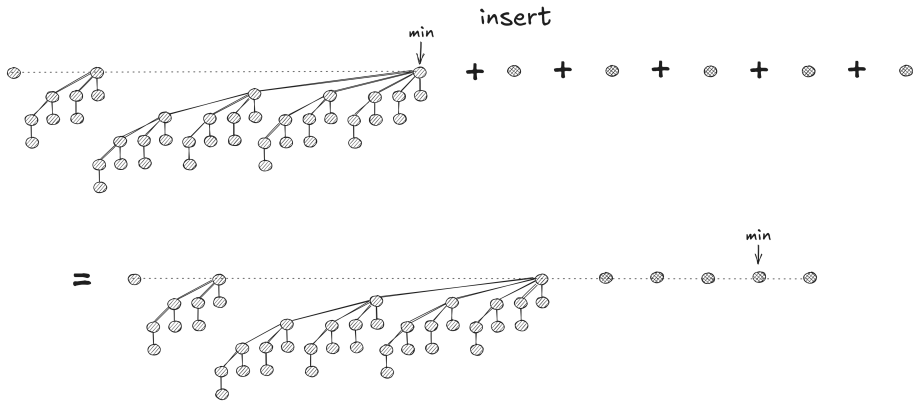
Táto stratégia má však svoju cenu: V halde sa môžu nachádzať viaceré stromy rovnakého rádu (pozri obr. 3). Špeciálne ak do haldy vložíme n nových prvkov, vznikne nám reťaz n koreňov (pozri obr. 4). Ak začíname z prázdnej haldy, dostaneme veľmi degenerovanú verziu „binomiálnej haldy“: všetky stromy majú nulovú hĺbku, nulový rád a počet koreňov je lineárny. Takáto štruktúra je v podstate obyčajný spájaný zoznam.

„No moment. Ale ako potom efektívne vyberieme minimum?“ môže napadnúť pozorného čitateľa. Dobrá otázka! Pri odstránení minima totiž musíme nájsť nový najmenší prvok, čo v prípade spájaného zoznamu koreňov znamená lineárny prechod. Znamená to, že sme prišli o logaritmickú zložitosť pre túto kľúčovú operáciu prioritnej fronty?

Odpoveď znie: *nie tak celkom*. Prišli sme síce o logaritmický čas v najhoršom prípade, no ukážeme si, ako dosiahnuť, že operácia `extract-min` zostane rýchla *z amortizovaného hľadiska*.

Keďže pri `extract-min` aj tak musíme prejsť všetky korene, aby sme určili nové minimum, využijeme tento nevyhnutný lineárny prechod rovno aj na *upratovanie*. Počas neho postupne pozlučujeme stromy rovnakého rádu do väčších, čím obnovíme tvar klasickej binomiálnej haldy. Výsledkom je, že po náročnom prvom upratovaní je halda opäť „v poriadku“ a *nasledujúce* operácie `extract-min`, spojené s upratovaním a hľadaním minima, už prebiehajú omnoho rýchlejšie.

Napríklad ak vložíme n prvkov a následne n-krát vyberieme minimum, všetky vkladania zaberú konštantný čas. Prvý výber spolu s upratаním trvá lineárny čas, no každý ďalší už len logaritmický čas. Všimnite si, že takýto prístup je



Obr. 4: Lenivé vkladanie prvkov. Keďže upratovanie odkladáme na neskôr, vzniká neutriedený spájaný zoznam koreňov.

pravdepodobne aj prakticky rýchlejší: namiesto n-krát [vloženie s upratаním] spravíme len [n-krát vloženie] a jedno veľké upratovanie. Počas prvých n rýchlych operácií si stihneme „nasporiť“ dostatok kreditov, ktorými potom zaplatíme jednu drahšiu operáciu. Samozrejme, jeden príklad ešte nie je dôkaz. V nasledujúcej sekcii si ukážeme *poriadny* dôkaz, že `merge` má amortizovanú zložitosť $O(1)$ a `extract-min` $O(\log n)$, a to pre ľubovoľnú postupnosť operácií.

Ešte predtým než sa do toho pustíme, skúste si rozmyslieť, ako toto upratovanie implementovať v čase lineárnom od počtu koreňov. Uvedomte si, že viacero koreňov môže mať rovnaký rád a môžu byť ľubovoľne rozhádzané po celom zozname.

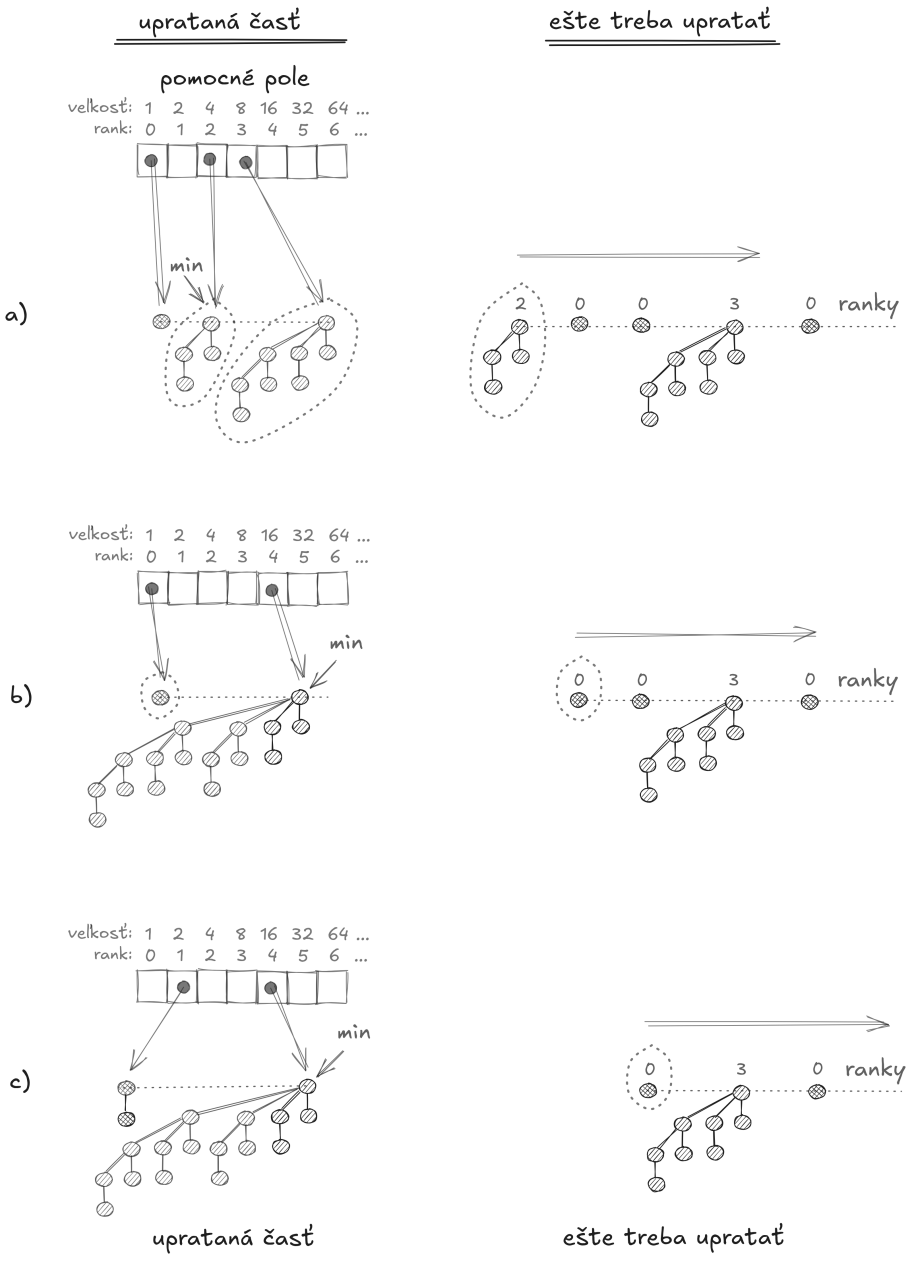
Amortizovaná analýza

Veta 1. *V lenivej binomiálnej halde majú operácie nasledujúcu amortizovanú zložitosť:*

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

Najskôr si ukážeme dôkaz pomocou účtovníckej metódy a následne ho prerozprávame rečou potenciálovej analýzy.

Predstavme si, že za každú vykonanú jednotkovú prácu platíme 1$ (t.j. 1$ pokryje $O(1)$ času). Za každú operáciu dostaneme určitý počet dolárov a našou úlohou bude ukázať, že tieto peniaze vždy postačia na zaplatenie všetkej práce a nikdy sa nedostaneme do mínusu. Pripomeňme, že pri amortizovanej analýze nemusíme minúť všetky peniaze hneď – časť si môžeme odložiť na neskôr, aby sme ňou pokryli náklady pomalších operácií.



Obr. 5: Upratovanie

■ **Dôkaz #1.** Ukážeme, že ak za každú operáciu dostaneme nasledovné množstvo peňazí:

- za `insert` 2$,
- za `merge` 1$ a
- za `extract-min` dostaneme $2\lfloor \lg n \rfloor + 1$ dolárov,

dokážeme nimi pokryť všetku potrebnú prácu.

Pri účtovaní sa budeme opierať o nasledujúci invariant:

Invariant. Každý koreň má vždy odložený 1$ na budúce upratovanie.

Operácia `merge`. Spojenie dvoch zoznamov a výber nového minima zaberie len konštantný čas. Túto prácu zaplatíme z prideleného 1$. Invariant ostáva automaticky zachovaný, pretože každý koreň mal už pred operáciou a stále má svoj 1$ odložený.

Operácia `insert`. Vytvorenie nového vrcholu a jeho pripojenie ako koreňa trvá $O(1)$. Jeden pridelený dolár zaplatí za túto prácu, druhý uložíme na účet nového koreňa, aby sme zachovali invariant.

Operácia `extract-min`. Rozdelíme ju na tri fázy:

1. *Odstránenie minima.* Odstránime koreň s najmenším kľúčom a jeho deti vložíme medzi ostatné korene. Tento krok zaplatí odstránený koreň zo svojho odloženého dolára.
2. *Príspevok novým koreňom.* Každé dieťa sa stane novým koreňom, preto mu vložíme na účet 1$. Spolu takto prerozdelíme najviac $\lfloor \lg n \rfloor$ dolárov.
3. *Upratovanie.* Nech t je počet koreňov pred upratovaním. Celé upratovanie trvá čas $O(t)$, respektíve stojí t dolárov.

   Označme ℓ počet koreňov, ktoré sa pripoja pod iné a r je počet tých, ktoré ostanú koreňmi. Zjavne $t = \ell + r$.

   Každý z tých ℓ koreňov má uložený 1$, a keďže po uprataní už koreňmi *nebudú*, tieto peniaze môžeme použiť na zaplatenie práce. *Jednu časť upratovania teda pokryje halda sama zo svojich úspor.*

   Po uprataní zostane najviac $r \leq \lceil \lg n \rceil \leq \lfloor \lg n \rfloor + 1$ koreňov. To je tak málo, že *túto druhú časť výdavkov môžeme pohodlne uhradiť z peňazí, ktoré sme dostali na samotnú operáciu.*

Prvá fáza sa zaplatí sama, na nové korene prispievame najviac $\lfloor \lg n \rfloor$ dolárov a samotné upratovanie pokryjeme z odložených dolárov odchádzajúcich koreňov plus z ďalších najviac $\lfloor \lg n \rfloor + 1$ dolárov pridelených na operáciu. Preto celkovo postačí $2\lfloor \lg n \rfloor + 1$ dolárov vyčlenených pre `extract-min`.

Tým je dokázané, že pridelené rozpočty (2$ na `insert`, 1$ na `merge` a $2\lfloor \lg n \rfloor + 1$ na `extract-min`) postačujú na zaplatenie všetkej práce a pritom sa invariant vždy zachováva. □

Dôkaz účtovníckou metódou je veľmi názorný, pretože si vieme presne odsledovať kam sa každý dolár uloží a na čo sa neskôr použije. Poďme si však ukázať ten istý dôkaz v jazyku potenciálovej metódy.

Pripomeňme, že amortizovanú zložitosť počítame ako skutočnú zložitosť plus rozdiel potenciálov. Potenciál $\Phi(H)$ si môžeme predstaviť ako celkovú sumu našetrených dolárov v danej chvíli: ak potenciál rastie, šetríme, ak klesá, znamená to, že prácu platíme z úspor. Dôležité je, že potenciál dátovej štruktúry vždy začína na nule (pri prázdnej halde) a nikdy nesmie klesnúť pod nulu – inak by sme minuli viac, než máme na účte.

■ **Dôkaz #2.** Zvoľme si potenciál haldy $\Phi(H)$ ako *počet binomiálnych stromov*, teda počet koreňov.

Operácia `merge` má $T_{\text{skutočný}} = O(1)$ a $\Delta\Phi = 0$; `insert` má $T_{\text{skutočný}} = O(1)$ a $\Delta\Phi = +1$ (pribudne nový koreň). Obe tieto operácie teda trvajú $O(1)$ amortizovane.

Pri `extract-min`, nech t_{pred} označuje počet stromov *na začiatku* a t_{po} počet stromov *na konci* operácie.

Skutočný čas operácie je

$$T_{\text{skutočný}} = O(t_{\text{pred}} + \log n),$$

pretože k pôvodným t_{pred} stromom môže pribudnúť najviac $\log n$ detí odstráneného koreňa a všetky tieto stromy je potrebné počas upratovania spracovať.

Zmena potenciálu je

$$\Delta\Phi = (t_{\text{po}} - t_{\text{pred}})\$.$$

Amortizovaný čas je teda

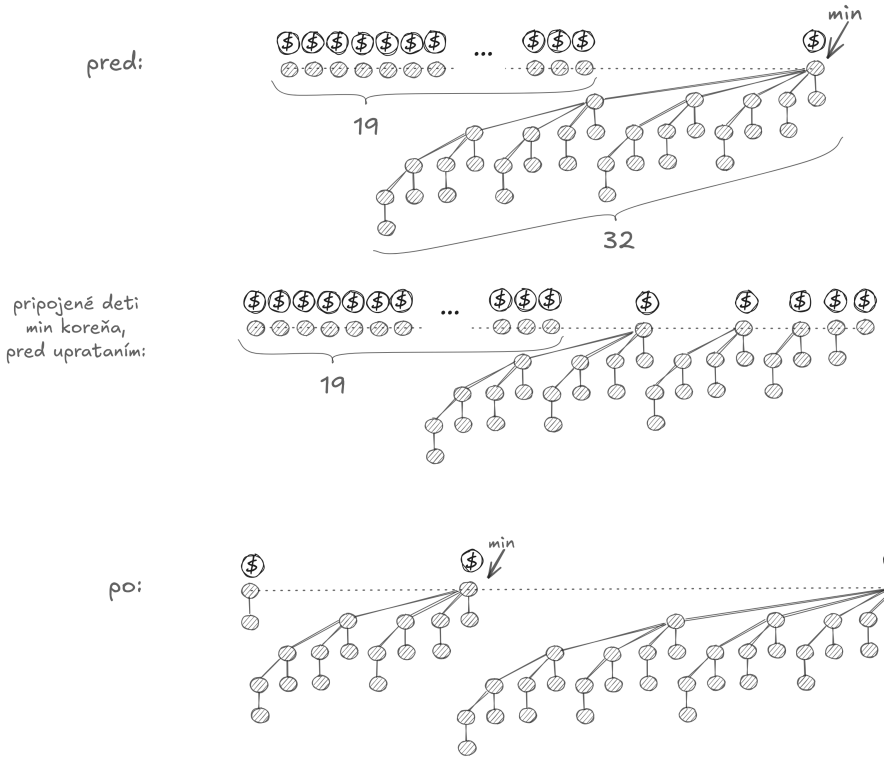
$$T_{\text{amort}} = T_{\text{skutočný}} + \Delta\Phi = O[(\cancel{t_{\text{pred}}} + \log n) + (t_{\text{po}} - \cancel{t_{\text{pred}}})] = O(\log n),$$

za predpokladu, že 1$ postačí na zaplatenie $O(1)$ inštrukcií. Počet stromov t_{pred} sa vyruší a zároveň platí $t_{\text{po}} \leq \log n$. □

Zhrnutie

V časti o lenivej binomiálnej halde sme videli, že klasickú štruktúru možno ďalej zefektívniť. Kým pri bežnej binomiálnej halde trvá spájanie $O(\log n)$ (čo je už samo o sebe pokrok oproti binárnej halde), stále to nie je ideálne. Myšlienka lenivého prístupu je jednoduchá: spájanie spravíme okamžite v čase $O(1)$, ale „upratovanie“ (zlúčenie stromov rovnakého rádu) odložíme na neskôr.

Takto síce získame $O(1)$ čas pre operácie `merge` aj `insert`, no v halde sa hromadí neporiadok (viaceré stromy rovnakého rádu). To znamená, že neskoršie upratovanie môže stáť až lineárny čas. Z amortizovaného hľadiska sa však ukáže,



	#stromov	reálna práca	vyúčtovanie
pred:	20	odstránenie minima + pripojenie detí: 1$	1$ zaplatí samotný koreň 5$ z peňazí na operáciu vložíme deťom na účet
po pripojení detí:	$-1+5=24$	upratovanie + nájdenie minima: 24$	21$ zaplatia bývalé korene 3$ zaplatíme z peňazí na operáciu
po uprataní:	$-21=3$		
spolu:		25$	8$ z peňazí na operáciu 17$ z našetrených peňazí

Obr. 6: Konkrétny príklad operácie `extract-min` spolu s vyúčtovaním. Na vstupe je halda s 19 stromami rádu 0 a jedným rádu 5 (spolu $19 + 2^5 = 51$ vrcholov). Po odstránení minima a pripojení jeho 5 detí (1$) bude treba upratať 24 binomiálnych stromov. Reálna práca teda stojí 25$. Na operáciu však dostaneme pridelených iba $2\lfloor \lg n \rfloor + 1 = 13$$. Kľúčové je, že počet stromov klesne z 20 pred operáciou na iba 3 po nej. Rozdiel 17 stromov predstavuje 17 našetrených dolárov, ktoré vieme použiť na zaplatenie zvyšku.

že cena je iba logaritmická: pred každou drahou operáciou predchádzalo veľa lacných, takže si dokážeme potrebné zdroje postupne „našetriť“.

Lenivá binomiálna halda je pekný príklad toho, že ak si dovolíme odložiť časť práce na neskôr a pozrieme sa na zložitosť z pohľadu amortizovanej analýzy, môžeme získať podstatne rýchlejšie priemerné časy operácií, ako keby sme sa snažili štruktúru neustále udržiavať úplne upratanú.

A to nie je všetko. V nasledujúcej kapitole sa pozrieme na to, ako zlepšiť operáciu `decrease-key`.

Operácia	**Binomiálna halda**	**Lenivá binomiálna halda**
insert	$O(\log n)$	$O(1)$
merge	$O(\log n_1 + \log n_2)$	$O(1)$
get-min	$O(1)$	$O(1)$
extract-min	$O(\log n)$	$O(\log n)$ amort.
decrease-key	$O(\log n)$	$O(\log n)$