1. Binomiální haldy

V této kapitole popíšeme datovou strukturu zvanou binomiální halda. Základní funkcionalita binomiální haldy je podobná binární haldě, nicméně jí dosahuje jinými metodami a navíc podporuje funkci BHMerge, která umí rychle sloučit dvě binomiální haldy do jedné.

Shrňme na začátek podporované operace spolu s jejich časy. Číslo N udává počet prvků v haldě a haldu zde chápeme jako minimovou.

Operace	*Čas*	*Komentář*
BHInsert	$\Theta(\log N)$, $\Theta^*(1)$	Vloží nový prvek.
BHGetMin	$\Theta(1)$	Vrátí minimum množiny.
BHExtractMin	$\Theta(\log N)$	Vrátí a odstraní minimum množiny.
BHMerge	$\Theta(\log N)$	Sloučí dvě haldy do jedné.
BHBuild	$\Theta(N)$	Postaví z N prvků haldu.
BHDecreaseKey	$\Theta(\log N)$	Sníží hodnotu klíče prvku.
BHIncreaseKey	$\Theta(\log^2 N)$	Zvýší hodnotu klíče prvku.
BHDelete	$\Theta(\log N)$	Smaže prvek.

Notací $\Theta^*(1)$ rozumíme amortizovanou složitost.

1.1. Zavedení binomiální haldy

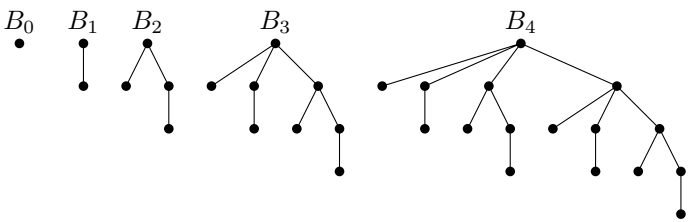
Namísto jediného stromu (jako má binární halda) sestává binomiální halda ze sady tzv. binomiálních stromů.

Binomiální stromy

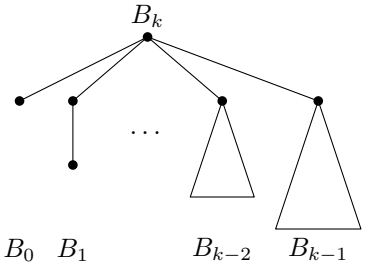
Definice: *Binomiální strom řádu k* (značíme B_k) je zakořeněný strom, pro který platí následující pravidla.

1. strom B_0 (řádu 0) obsahuje pouze kořen
2. strom B_k pro $k > 0$ má kořen, který má právě k synů, přičemž tito synové jsou zároveň kořeny binomiálních stromů po řadě B_0, B_1 až B_{k-1}.

Náhled na strukturu binomiálního stromu získáme z obrázku 1.1. Také se podívejme, jak budou vypadat některé nejmenší binomiální stromy (viz obr. 1.2).



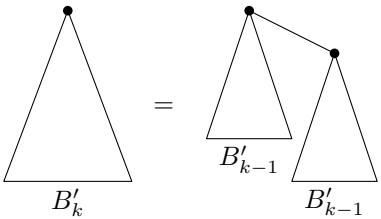
Obr. 1.2: Příklady binomiálních stromů



Obr. 1.1: Binomiální strom řádu k

Podáme ještě jednu definici binomiálních stromů (tzv. rekurzivní definici binomiálních stromů), pro níž následně ukážeme její ekvivalenci s předchozí definicí.

Definice: Zakořeněné stromy B'_k jsou definovány takto: B'_0 obsahuje pouze kořen a pro $k > 0$ se B'_k skládá ze stromu B'_{k-1}, pod jehož kořenem je napojený další strom B'_{k-1}.



Obr. 1.3: Rekurzivní definice binomiálního stromu

Lemma 1: Stromy B_k a B'_k jsou izomorfní.

Důkaz: Postupujme matematickou indukcí. Pro $k = 0$ tvrzení zjevně platí. Zvolme $k > 0$. Pod kořenem stromu B_k jsou dle definice zavěšeny stromy $B_0, \ldots, B_{k-1}$. Odtržením posledního podstromu B_{k-1} od B_k však dostáváme strom B_{k-1}. To dává přesně definici stromu B'_k. Naopak, uvážíme-li strom B'_k, z indukce vyplývá, že B'_{k-1} je izomorfní B_{k-1}, pod jehož kořen jsou dle definice napojeny stromy $B_0, \ldots, B_{k-2}$. Pod kořen B'_k jsou tudíž napojeny stromy $B_0, \ldots, B_{k-1}$. □

Lemma 2: Počet hladin stromu B_k je roven $k + 1$ a počet jeho vrcholů je roven 2^k.

Důkaz: Dokážeme matematickou indukcí. Strom B_0 má jistě 1 hladinu a $2^0 = 1$ vrchol. Zvolme $k > 0$. Z indukčního předpokladu vyplývá, že hloubka B_{k-1} je k a počet vrcholů je 2^{k-1}. Užitím předchozího lemmatu dostáváme, že strom B_k je složený ze dvou stromů B_{k-1}, z nichž jeden je o hladinu níže než druhý, což dává počet hladin $k + 1$ stromu B_k. Složením dvou stromu B_{k-1} dostáváme $2 \cdot 2^{k-1} = 2^k$ vrcholů. □

Důsledek: Binomiální strom s N vrcholy má hloubku $\mathcal{O}(\log N)$ a počet synů kořene je taktéž $\mathcal{O}(\log N)$.

Od stromu k haldě

Z binomiálních stromů nyní zkonstruujeme binomiální haldu. Pro uložení $N = 2^\ell$ prvků stačí zvolit strom B_ℓ. Pro $N \neq 2^\ell$ využijeme vlastností dvojkového zápisu čísla N: haldu sestavíme z binomiálních stromů takový řádů, pro něž jsou nastaveny příslušné bity v čísle N.

Definice: *Binomiální halda* obsahující N prvků se skládá ze souboru stromů $\mathcal{T} = T_1, \ldots, T_\ell$, kde

1. Uchovávané prvky jsou uloženy ve vrcholech stromů T_i. Prvek uložený ve vrcholu $v \in V(T_i)$ značíme $h(v)$.
2. Pro každý strom T_i platí tzv. haldová podmínka, neboli pro každý $v \in V(T_i)$ a jeho syny $s_1, \ldots, s_k$ platí $h(v) \leq h(s_j)$, $j = 1, \ldots, k$.
3. Každý strom T_i je binomiální strom.
4. V souboru $\mathcal{T}$ se žádné dva řády binomiálních stromů nevyskytují dvakrát.
5. Soubor stromů $\mathcal{T}$ je uspořádán vzestupně podle řádu binomiálního stromu.

Jako vhodný způsob uložení souboru stromů $\mathcal{T}$ tedy poslouží například spojový seznam. Ve spojovém seznamu lze i jednoduše udržovat seznamy synů jednotlivých vrcholů v binomiálním stromě.

Tvrzení: Binomiální strom řádu k se vyskytuje v souboru stromů N-prvkové binomiální haldy právě tehdy, když je v dvojkovém zápisu čísla N nastavený k-tý nejnižší bit na 1.

Důkaz: Z definice binomiální haldy vyplývá, že binomiální stromy dohromady dávají $\sum_{i=1}^{k} b_i 2^i = N$, kde k je maximální řád stromu v $\mathcal{T}$ a $b_i = 0$ nebo $b_i = 1$. Z vlastností zápisu čísla v dvojkové soustavě vyplývá, že pro dané N jsou čísla b_i (a tím i řády binomiálních stromů v $\mathcal{T}$) určena jednoznačně. Čísla $b_k, b_{k-1}, \ldots, b_1$ tedy tvoří zápis N v dvojkové soustavě. □

Důsledek: N-prvková binomiální halda sestává z $\mathcal{O}(\log N)$ binomiálních stromů.

1.2. Operace s binomiální haldou

Nalezení minima

Jak jsme již ukázali u binární haldy, pokud strom splňuje haldovou podmínku, musí se minimum v něm uložené nacházet v jeho kořeni. Minimum cele haldy se tedy musí nacházet v jednom kořenů stromů T_i. Operaci BHGetMin tedy postačí projít seznam $\mathcal{T}$, což bude trvat čas $\mathcal{O}(\log N)$.

Operaci BHGetMin lze urychlit na čas $\Theta(1)$ tím, že binomiální haldu rozšíříme o ukazatel na globální minimum. Při každé další operaci nad binomiální haldou potom tento ukazatel přepočítáme, například průchodem seznamu $\mathcal{T}$. Čtenář nechť si povšimne, že s vyjímkou amortizovaného BHInsert na to každá operace bude mít dostatek času.

Slévání

Operaci BHMerge poněkud netypicky popíšeme jako jednu z prvních, protože ji budeme nadále používat jako podproceduru ostatních operací.

Algoritmus slévání vezme dvě binomiální haldy a vytvoří z nich jedinou, která obsahuje prvky obou hald a přitom zachovává všechny vlastnosti haldy popsané výše. V první fázi provedeme klasické slití dvou uspořádaných seznamů. Takto slitý seznam bude obsahovat stromy z obou hald, takže se může stát, že od některých řádů budou v seznamu stromy dva. V druhé fázi provedeme *konsolidaci*, která pospojuje stromy stejných řádů tak, aby zbyl od každého nejvýše jeden.

Konsolidace bude probíhat následovně. Připravíme pomocné pole obsahující $\lceil \log_2 N \rceil + 1$ přihrádek (číslovaných od 0). Stromy ze seznamu rozdělíme do přihrádek tak, že v i-té přihrádce jsou všechny stromy řádu i. Poté projdeme všechny přihrádky počínaje od 0. Pokud v některé nalezneme alespoň dva stromy, spojíme tyto stromy do jednoho o řád většího stromu a nový strom vložíme do přihrádky s o jedna vyšším indexem. Na konec obsahy přihrádek pospojujeme ve správném pořadí do výsledného spojového seznamu.

Algoritmus BHMerge
Vstup: Binomiální haldy H_1, H_2
Výstup: Binomiální halda H_{out} poskládaná z prvků H_1 a H_2

1. Pokud H_1 nebo H_2 je prázdná:
2. Vyřešíme triviálně a skončíme. *(Slévání se nekoná.)*
3. Slij seznamy H_1 a H_2 setříděně do H_{tmp}.
4. Připrav pole $P[0 \ldots \lceil \log_2 |H_{tmp}| \rceil]$ spojových seznamů.
5. Pro všechny stromy $s \in H_{tmp}$:
6. Odtrhni s ze seznamu H_{tmp}.
7. Vlož s do $P[\check{r}\acute{a}d(s)]$.
8. $H_{out} \leftarrow$ nová prázdná halda
9. Pro všechny přihrádky i v P (počínaje od 0):
10. Pokud má $P[i]$ alespoň dva stromy:
11. $b_1, b_2 \leftarrow$ odtrhni první dva stromy z $P[i]$.
12. $b \leftarrow$ BHMerge*Tree*(b_1, b_2)
13. Vlož b do $P[i+1]$.
14. Obsah $P[i]$ připoj na konec H_{out}.
15. Přepočítej v H_{out} ukazatel na minimální prvek

Spojení dvou stromů, které používáme v předchozím algoritmu je velice jednoduché. Při spojování je potřeba napojit kořen jednoho stromu jako posledního syna kořene druhého stromu. Přitom musíme dát pozor, aby zůstala zachována haldová podmínka, takže vždy zapojujeme kořen s větším prvkem pod kořen s menším prvkem.

Procedura MergeTree
Vstup: Stromy b_1, b_2 z binomiální haldy ($\check{r}\acute{a}d(b_1) = \check{r}\acute{a}d(b_2)$)
Výstup: Výsledný strom b_{out}

1. Pokud $ko\check{r}en(b_1) \leq ko\check{r}en(b_2)$:
2. Připoj $ko\check{r}en(b_2)$ jako posledního syna pod $ko\check{r}en(b_1)$.
3. $b_{out} \leftarrow b_1$
4. Jinak:
5. Připoj $ko\check{r}en(b_1)$ jako posledního syna pod $ko\check{r}en(b_2)$.
6. $b_{out} \leftarrow b_2$

Tvrzení: Algoritmus BHMerge je korektní a jeho časová složitost je $\Theta(\log N)$.

Důkaz: Algoritmus konsolidace používá pouze cykly pevných délek (přes všechny prvky resp. přes všechny přihrádky), takže je zcela jistě konečný.

Ukažme, že po konsolidaci nebudou ve výsledném spojovém seznamu dva stromy stejného řádu: V každé přihrádce se nikdy nevyskytují více než tři stromy. Na začátku mohou být nejvýše dva a nejvýše jeden může být přidán po slévání z přihrádky s o jedna menším indexem. Díky tomu je jasné, že po průchodu přihrádkami bude v každé nejvýše jeden strom, takže ve výsledné haldě se nemohou vyskytovat dva stromy se stejným řádem.

Složitost slévání je přímo úměrná počtu přihrádek, které vytvoříme. V každé přihrádce jsou nejvýše 3 stromy (tedy konstantní počet) a za každou přihrádku provedeme také nejvýše jedno spojení stromů. Celková složitost v nejhorším případě bude tedy $\Theta(\log N)$. □

Vkládání prvků a postavení haldy

Operaci BHInsert vyřešíme snadno. Vytvoříme novou binomiální haldu obsahující pouze vkládaný prvek a následně zavoláme slévání hald.

Snadno nahlédneme, že pouhé přeuspořádání stromů při přidání nového prvku tak, aby v seznamu nebyly dva stromy stejného řádu, může v nejhorším případě vyžadovat čas $\Theta(\log N)$ operací.

Algoritmus BHInsert
Vstup: Binomiální halda H, vkládaný prvek x
Výstup: Binomiální halda H s vloženým prvkem

1. Vytvoř binomiální haldu H_{tmp} s jediným prvkem x.
2. $H \leftarrow BH$BHMerge(H, H_{tmp})

Tvrzení: Operace BHInsert má časovou složitost $\Theta(\log N)$ worst-case. Pro na počátku N-prvkovou haldu trvá libovolná posloupnost K volání operace BHInsert čas $\mathcal{O}(N + K)$; BHInsert má tedy amortizovanou časovou složitost $\Theta(1)$.

Důkaz: Jednoprvkovou haldu umíme určitě vytvořit v konstantním čase, takže těžiště práce bude ve slévání hald. Slévání zvládneme v čase $\Theta(\log N)$, tedy i vkládání prvku bude mít v nejhorším případě logaritmickou časovou složitost.

Slévání dvou hald skutečně pracuje v logaritmickém čase, avšak při vkládání má jedna ze slévaných hald pouze jeden prvek. V nejhorším případě se samozřejmě může stát, že původní halda má všechny stromy od B_0 až po $B_{\lceil \log_2 N \rceil - 1}$, takže při slévání dojde k řetězové reakci a postupně se všechny stromy sloučí do jediného, což si vyžádá $\Theta(\log N)$ operací. Nicméně pokud je například počet prvků v původní haldě sudý, pak strom B_0 v jejím spojovém seznamu chybí a slévání s jednoprvkovou haldou je možné provést v konstantním čase.

Poznamenejme ještě, že při slévání původní a nové jednoprvkové haldy, lze spojení seznamů provést v konstantním čase, takže nás budou z hlediska asymptotické složitosti zajímat pouze operace spojení dvou stromů.

Pro amortizovanou analýzu využijeme skutečností dokázaných pro binární sčítačku. Připomeňme, že posloupnost K inkrementů v N-bitové binární sčítačce trvá čas $\mathcal{O}(N + K)$. Nyní si všimněme, že slití dvou binomiálních stromů přesně nastane v okamžiku, kdy se v inkrementované binární sčítačce dojde k součtu dvou jedničkových bitu. Počet bitových změn je tak úměrný počtu spojení binomiálních stromů.

Z toho už jednoduše vyplývá celková časová složitost $\mathcal{O}(N + K)$ pro K volání operace BHInsert. □

Předešlá amortizovaná analýza operace BHInsert dává návod na realizaci rychlé operace BHBuild pro postavení binomiální haldy opakovaným voláním operace BHInsert.

Důsledek: Posloupnost N volání operace BHInsert trvá čas $\Theta(N)$.

Všimněme si, že narozdíl od binární haldy, jejíž rychlá stavba vyžadovala speciální postup, zde nám pro rychlé postavení binomiální haldy stačilo pouze lépe analyzovat časovou složitost.

Odstranění minima

Odstranění minima je nepatrně komplikovanější než vkládání prvků, nicméně opět použijeme operaci BHMerge. Při odstraňování nejprve nalezneme strom M, jehož kořen je minimem, a tento strom odpojíme z haldy. Následně odtrhneme všechny syny (včetně jejich podstromů) kořene M a vložíme je do nové binomiální haldy. Tato operace je poměrně jednoduchá, neboť se mezi syny dle definice binomiálního stromu nikdy nevyskytují dva stromy stejného řádu. Na konec slijeme novou haldu s původní, čímž se odtržené prvky začlení zpět.

Algoritmus BHExtractMin
Vstup: Binomiální halda H
Výstup: Binomiální halda H s odstraněným minimem

1. $m \leftarrow$ strom s nejmenším kořenem v haldě H.
2. Odeber m z H.
3. Vytvoř prázdnou binomiální haldu H_{tmp}.
4. Pro každého syna s kořene stromu m:

5. Odtrhni podstrom s kořenem v s a vlož jej do H_{tmp}.
6. Odstraň m. *(Zbyde nám jen kořen, který odstraníme.)*
7. $H \leftarrow BH\text{BHMerge}(H, H_{tmp})$

Tvrzení: Časová složitost operace BHExtractMin v N-prvkové binomiální haldě je $\Theta(\log N)$. Lepší časové složitosti nelze dosáhnout.

Důkaz: Vytvoření dočasné haldy pro podstromy odstraňovaného kořene zabere nejvýše tolik času, kolik podstromů do ni vkládáme – tedy $\mathcal{O}(\log N)$. Slévání hald má také logaritmickou složitost, takže celková složitost algoritmu v nejhorším případě je $\Theta(\log N)$. Nemůžeme činit žádné předpoklady o tom, kolik synů má kořen obsahující minimum (nejvýše však $\mathcal{O}(\log N)$, takže na rozdíl od vkládání zde bude amortizovaná složitost rovna složitosti v nejhorším případě.

Dolní odhad složitosti odstraňování minima získáme z dolního odhadu složitosti třídění. Kdyby existoval rychlejší algoritmus na odstranění minima, zkonstruovali bychom rychlejší třídící algoritmus než $\mathcal{O}(N \log N)$ vložením tříděných prvků operací BHBuild do haldy a následně N-násobým odstraněním minima, což by dalo lepší časovou složitost než $\mathcal{O}(N \log N)$. □

V úvodu této kapitoly jsme uvedli ještě operace BHDecreaseKey, BHIncreaseKey a BHDelete, které dostanou ukazatel na binomiální haldu a ukazatel na prvek v ní, a provedou po řadě snížení jeho klíče, zvýšení klíče a smazání prvku. Tyto operace přenecháme čtenáři jako cvičení 1.2.7, 1.2.8 a 1.2.9.

Implementace a srovnání s regulární haldou

Nyní je správný čas zeptat se, jaké výhody nám přinese binomiální halda oproti např. klasické binární. Pomineme-li, že binomiální haldy jsou zajímavé z hlediska teoretického, zbývají nám poměrně jasné ukazatele kvality – časová a paměťová složitost. Časové složitosti základních operací přehledně shrnuje následující tabulka (hvězdičkou jsou označeny amortizované složitosti).

Operace	binární	d**-regulární**	**binomiální**
Přístup k minimu	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Vkládání prvku	$\Theta(\log N)$	$\Theta(\log_d N)$	$\Theta(\log N)$, $\Theta^*(1)$
Odstranění minima	$\Theta(\log N)$	$\Theta(d \log_d N)$	$\Theta(\log N)$

Binomiální halda se od klasické binární haldy liší pouze v amortizované složitosti vkládání. Zde se ještě sluší připomenout, že binární haldu umíme postavit v lineárním čase, takže pokud bychom prvky nejprve vkládali do prázdné haldy a až po vložení všech prvků je začali odebírat, dostaneme se i s binární haldou na konstantní amortizovanou složitost vkládání. Binomiální halda může mít na druhou stranu navrch v situacích, kdy se často střídají operace vkládání a vypouštění prvků.

Abychom mohli řádně porovnat paměťové nároky, potřebujeme nejprve upřesnit jak reprezentovat binomiální strom v paměti. Regulární haldy jsme bez potíží

zvládli uložit do pole. Pokud bychom se pokusili vtěsnat do pole binomiální strom, některé operace (např. spojení dvou stromů) nám značně podraží co do časové složitosti. Zkusíme tedy přímočarý přístup. Jednotlivé uzly stromu budou dynamicky alokované struktury, které provážeme ukazateli. Každý uzel pak bude obsahovat jeden prvek a pole odkazů na všechny své syny.

Na první pohled by se mohlo zdát, že na takovou reprezentaci budeme potřebovat poměrně velké množství paměti. Každý uzel může mít až $\mathcal{O}(\log N)$ synů, takže celý strom pak zabere $\mathcal{O}(N \log N)$ paměti. Pozorný čtenář už jistě tuší, že se jedná pouze o hrubý horní odhad a že by mohl jít ještě vylepšit. Právě polovina uzlů jsou[1] totiž listy a nepotřebují tedy žádnou paměť na odkazy na syny. Naopak pouze jeden uzel (kořen) bude mít $\log_2 N$ synů. Zkusme se podívat, kolik takových ukazatelů bude v jednom binomiálním stromě. Na každý prvek odkazuje právě jeden ukazatel, takže bez ohledu na to, kolik má který uzel synů, v celém stromě je $\Theta(N)$ ukazatelů. Celková složitost pak bude $\Theta(N)$ na reprezentaci stromů a $\Theta(\log N)$ na spojový seznam kořenů, což je ve výsledku $\Theta(N + \log N) = \Theta(N)$. V asymptotické paměťové složitosti si tedy binomiální halda v ničem nezadá s klasickou, avšak kvůli potřebě ukazatelů na prvky bude mít binomiální halda horší multiplikativní konstantu.

Poslední věc, kterou je třeba vzít v úvahu je složitost implementace. Naprogramovat operace na klasické haldě je mnohem snazší, než naprogramovat haldu binomiální. Je tedy potřeba zvážit, zda se tato práce navíc vyplatí.

Cvičení:

1. Přeformulujte všechny definice a operace pro maximovou haldu.
2. Dokažte, že libovolné přirozené číslo x lze zapsat jako konečný součet mocnin dvojky $2^{k_1} + 2^{k_2} + \ldots$ tak, že každé $k_i \neq k_j$ pro různá i, j.
3. Ukažte, že sčítanců v předchozím cvičení je nejvýše $\lceil \log_2 x \rceil$
4. Upravte algoritmus BHMerge tak, aby nepotřeboval pole přihrádek, ale vystačil s konstantní pomocnou pamětí. Časovou složitost musíte pochopitelně zachovat.
5. U binomiální haldy jsme naznačili, že minimum (přesněji referenci na kořen obsahující minimum) můžeme udržovat stranou, abychom k němu mohli přistupovat v konstantním čase. Upravte operace slití, vkládání prvků a vypouštění minima tak, aby zároveň udržovali odkaz na minimum. Ukažte, že tato práce navíc nezhorší časové složitosti operací.
6. Mějme modifikaci binomiální haldy, ve které jsou stromy setříděné sestupně podle řádu (nikoli vzestupně). Opravte operaci slévání hald pro tuto reprezentaci, abyste přitom zachovali její časovou i paměťovou složitost.
7. Navrhněte operaci BHDecreaseKey s časovou složitostí $\Theta(\log N)$.
8. Navrhněte operaci BHIncreaseKey s časovou složitostí $\Theta(\log^2 N)$.
9. Navrhněte operaci BHDelete s časovou složitostí $\Theta(\log N)$.

[1] Pokud má strom řád alespoň 1.

1.3. Líná binomiální halda

Alternativou k „pilné“ binomiální haldě je tzv. líná („lazy“) binomiální halda. Její princip spočívá v odložení některých úkonů při vkládání prvků a odstranění minima, dokud nejsou opravdu potřeba.

Změny v reprezentaci

Líná binomiální halda se téměř neliší od pilné. Pouze povolíme, že se ve v souboru stromů může vyskytovat více stromů stejného řádu. Pro jednoduchost budeme navíc předpokládat, že soubor stromů je uložen v obousměrném kruhovém seznamu. Podívejme se, jaké výhody nám to přinese.

Operace slití dvou hald, kterou využívá vkládání prvku i vypuštění minima se značně zjednoduší. Vzhledem k tomu, že se stromy mohou v haldě opakovat, slití není ničím jiným než spojením dvou seznamů, což jistě zvládneme v konstantním čase. Aby ale halda nezdegenerovala v obyčejný spojový seznam, musíme čas od času provést *konsolidaci* a stromy sloučit tak, aby jich bylo v seznamu co nejméně. Nejvhodnější čas na tento úklid je při hledání minima. Při hledání beztak procházíme všechny stromy v seznamu, takže je můžeme zároveň spojovat.

Pozorný čtenář jistě nad předchozím odstavcem pozvedne obočí. Proč bychom nemohli použít stejný trik jako u pilné haldy a pamatovat si neustále ukazatel na strom s nejmenším kořenem? Takové vylepšení by bylo samozřejmě možné a konsolidaci bychom pak prováděli při odstranění minima (místo při hledání). Tímto vylepšením se však nebudeme zabývat a ponecháme jej na rozmyšlenou do cvičení.

Konsolidace

Konsolidace je velice podobná druhé fázi algoritmu Merge. Všechny stromy rozdělíme do $\lceil \log N \rceil + 1$ přihrádek (číslovaných od 0) tak, že v i-té přihrádce se budou nacházet všechny stromy řádu i. Bohužel již nemůžeme činit žádné předpoklady o počtech stromů v jednotlivých přihrádkách. Z každé přihrádky budeme tedy odebírat stromy po dvou a slučovat je dokud to bude možné (zbude pouze jeden nebo žádný strom).

Algoritmus LazyHeapConsolidation
Vstup: Líná implementace binomiální haldy H o N prvcích
Výstup: Zkonsolidovaná halda H

1. Pokud je H prázdná: Konec.
2. Připrav pole $P[0 \ldots \lceil \log N \rceil]$ spojových seznamů.
3. Pro všechny stromy s v H:
4. Odtrhni s z H.
5. Vlož s do $P[\check{r}\acute{a}d(s)]$.
6. Pro všechny přihrádky i v P:
7. Dokud má $P[i]$ alespoň dva stromy:
8. $b_1, b_2 \leftarrow$ odtrhni první dva stromy z $P[i]$.
9. $b \leftarrow BH$BHMerge$Tree(b_1, b_2)$

10. Vlož b do $P[i+1]$.
11. Pokud v $P[i]$ zbyl strom s:
12. Odtrhni s z $P[i]$ a vlož jej do H.

Funkčnost algoritmu dokážeme velmi podobně, jako u slévání hald. V každé přihrádce nám zbude nejvýš jeden strom, takže se ve výsledné haldě nemohou nacházet dva stromy stejného řádu. O něco komplikovanější bude analyzovat časovou složitost.

Časová složitost konsolidace

Tvrzení: Časová složitost konsolidace líné binomiální haldy je $\Theta(N)$ v nejhorším případě a $\Theta(\log N)$ amortizovaně.

Důkaz: Pokud jsme do haldy jen vkládali, bude se skládat z N jednoprvkových stromů. Všechny stromy musíme nejprve vložit do správných přihrádek (což nám zabere$\Theta(N)$). Následně každý strom buď sloučíme s jiným (tzn. zapojíme pod kořen jiného stromu), nebo jej vložíme do výsledného spojového seznamu. Na to budeme potřebovat opět právě $\Theta(N)$ operací. Samozřejmě bychom neměli zapomenout na inicializaci a procházení všech přihrádek, avšak to nám zabere pouze $\Theta(\log N)$, což je menší než výsledná lineární složitost.

Na první pohled vidíme, že zřídka kdy bude konsolidace skutečně trvat $\Theta(N)$. Vždy musíme alespoň inicializovat a projít všechny přihrádky, takže konsolidace bude trvat nejméně $\Omega(\log N)$.

Amortizovanou složitost analyzujeme penízkovou metodou. Zavedeme pravidlo, že každý strom v haldě musí mít neustále uložen na svém účtu jeden peníz. Za tento peníz zvládne zaplatit libovolnou konstantní operaci[2]. Spočteme, kolik operací se bude s každým stromem provádět při konsolidaci.

Nejprve vložíme každý strom do příslušné přihrádky, což je určitě trvá $\Theta(1)$. Na následné spojování stromů můžeme nahlížet tak, že napojujeme jeden strom pod jiný (tedy jeden zanikne a jeden se pouze zvětší). Práci za toto napojení a následný přesun vzniklého stromu do nové přihrádky zaplatíme penízem stromu, který je napojován. Obě tyto operace jsou konstantní a každý strom může být napojen nejvýše jednou. Každý strom tedy ze svého konta zaplatí nejvýše konstantní počet operací.

Na konci je potřeba ještě přesunout zbývající stromy do spojového seznamu a také zajistit, aby na svých kontech měly znovu jeden peníz. Na to již nemáme nikde našetřeno a budeme muset tuto složitost vykázat. Zbývajících stromů je však již nejvýš $\Theta(\log N)$. Příprava a úklid přihrádek nám zabere také logaritmický čas, takže i vykázaná složitost bude $\Theta(\log N)$. □

Na závěr se ještě podívejme, jak se nám promítnou tyto předpoklady do ostatních operací (vkládání prvku a odstranění minima).

[2] Dokonce libovolné (konstantní) množství konstantních operací, neboť $k \cdot \Theta(1) = \Theta(k) = \Theta(1)$.

Důsledek: Časová složitost vkládání prvku do N-prvkové líné binomiální haldy je $\Theta(1)$ worst-case a časová složitost odstraňování minima je $\Theta^*(\log N)$ amortizovaně.

Důkaz: Při vkládání prvku vytvoříme jeden strom B_0, kterému dáme do vínku jeden peníz, a ten vložíme do haldy. Všechny úkony zvládneme v konstantním čase, takže celková složitost vkládání je $\Theta(1)$. Zde bychom měli zdůraznit, že se jedná o složitost v nejhorším případě, nikoli o amortizovanou složitost, jako tomu bylo u pilné haldy.

Při odstraňování minima budeme předpokládat, že neustále udržujeme ukazatel na strom s nejmenším kořenem. Po odebrání minima a opětovném začlenění odtržených podstromů do haldy provedeme konsolidaci, při které ukazatel na minimum aktualizujeme. Po odtržení synů nejmenšího kořene musíme každému z nich dát na konto jeden peníz a vložit je do spojového seznamu. To nám zabere právě $\Theta(\log N)$ času. Následná konsolidace bude trvat nejdéle $\Theta(N)$, avšak amortizovaně pouze $\Theta(\log N)$, jak jsme již ukázali dříve. □

Srovnání pilné a líné binomiální haldy

Líná binomiální halda má na rozdíl od pilné rychlejší vkládání prvků a garantuje, že i v nejhorším případě nebudeme potřebovat na vložení víc než konstantně mnoho času. Naproti tomu odebrání minima se může díky konsolidaci zpomalit až na $\Theta(N)$ v nejhorším případě. Naštěstí amortizovaná složitost odebírání zůstane na $\Theta(\log N)$. Pro přehlednost se podívejme do následující tabulky (hvězdičkou jsou označeny amortizované složitosti):

Operace	**Pilná halda**	**Líná halda**
Vkládání prvku	$\Theta(\log N)$, $\Theta^*(1)$	$\Theta(1)$
Odstranění minima	$\Theta(\log N)$	$\Theta(N)$, $\Theta^*(\log N)$

Význam líné binomiální haldy vzroste především v další kapitole, kde slouží jako předstupeň pro návrh tzv. Fibonacciho haldy.

Cvičení:

1. Zjistěte, jak by se změnily složitosti jednotlivých operací, kdybychom v implementaci používali místo obousměrného kruhového seznamu pouze jednosměrný lineární.
2. Zjistěte, jak by se změnily složitosti jednotlivých operací, kdybychom v implementaci používali místo obousměrného kruhového seznamu pouze jednosměrný lineární a udržovali zároveň ukazatel na poslední prvek seznamu.
3. Předpokládejme, že bychom chtěli neustále udržovat ukazatel na strom obsahující minimum jako v pilné implementaci. Upravte podle toho všechny operace s haldou.
4. Ukažte, že provedené modifikace nezhoršily časové složitosti jednotlivých operací.