1. Pokročilé haldy

Při analýze Dijkstrova algoritmu v oddílu **??** jsme zatoužili po haldě, která by některé operace uměla rychleji než obyčejná binární halda. V této kapitole postupně odvodíme tři datové struktury: binomiální haldu, línou binomiální haldu a Fibonacciho haldu. Poslední z nich bude mít vytoužené vlastnosti.

1.1. Binomiální haldy

Základní funkce binomiální haldy jsou podobné binární haldě, nicméně jich dosahuje jinými metodami. Navíc podporuje operaci MERGE, která umí rychle sloučit dvě binomiální haldy do jedné.

Shrňme na začátek podporované operace spolu s jejich worst-case časovými složitostmi (tedy složitostmi v nejhorším případě). Číslo n udává počet prvků v haldě a haldu zde chápeme jako minimovou.

operace	*složitost*	*činnost*
INSERT	$\Theta(\log n)$	vloží nový prvek
MIN	$\Theta(1)$	vrátí minimum množiny
EXTRACTMIN	$\Theta(\log n)$	vrátí a odstraní minimum množiny
MERGE	$\Theta(\log n)$	sloučí dvě haldy do jedné
BUILD	$\Theta(n)$	postaví z n prvků haldu
DECREASE	$\Theta(\log n)$	sníží hodnotu klíče prvku
DELETE	$\Theta(\log n)$	smaže prvek
INCREASE	$\Theta(\log n)$	zvýší hodnotu klíče prvku

Stále platí, že podle klíče neumíme vyhledávat, takže operace DECREASE, DELETE a INCREASE musí dostat ukazatel na prvek v haldě, nikoliv jeho klíč.

Nyní binomiální haldu definujeme. Na rozdíl od binární haldy nebude mít tvar stromu, nýbrž souboru více tzv. binomiálních stromů.

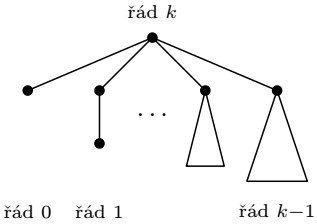
Binomiální stromy

Definice: Řekneme, že zakořeněný strom T je *binomiálním stromem řádu k*, pokud splňuje následující pravidla:

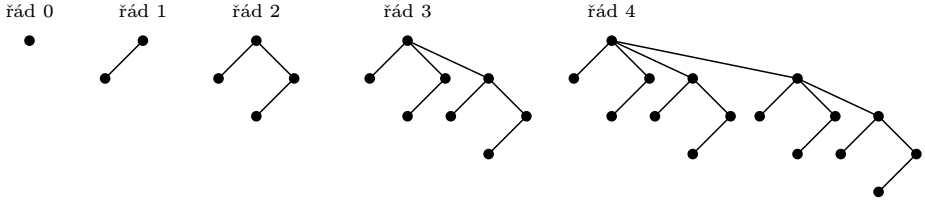
1. Pokud je řád k roven nule, pak T obsahuje pouze kořen.
2. Pokud je řád k nenulový, pak T má kořen s právě k syny. Tito synové jsou kořeny podstromů, které jsou po řadě binomiálními stromy řádů $0, \ldots, k-1$.

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

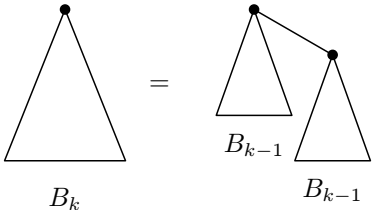
Podáme nyní tzv. rekurzivní definici binomiálních stromů, pro níž následně ukážeme ekvivalenci s předchozí definicí.



Obr. 1.1: Binomiální strom řádu k



Obr. 1.2: Příklady binomiálních stromů



Obr. 1.3: Rekurzivní definice binomiálního stromu

Definice: Zakořeněné stromy B_k jsou definovány takto: B_0 obsahuje pouze kořen, B_k pro $k > 0$ se skládá ze stromu B_{k-1}, pod jehož kořenem je napojený další strom B_{k-1}.

Lemma R (o rekurzivní definici): Strom B_k je binomiální strom řádu k. Každý strom T, který je binomiální strom řádu k, je izomorfní stromu B_k.

Důkaz: Postupujme matematickou indukcí. Pro $k = 0$ tvrzení zjevně platí. Zvolme $k > 0$. Pod kořenem stromu T jsou dle definice zavěšeny stromy binomiální stromy řádů $0, \ldots, k-1$. Odtržením posledního binomiálního podstromu S řádu $k-1$ a použitím indukčního předpokladu dostáváme z T binomiální strom řádu $k-1$, tedy B_{k-1}. Opětovným připojením S zpět dostáváme přesně strom B_k.

Naopak, uvážíme-li strom B_k, z indukce vyplývá, že B_{k-1} je binomiální strom řádu $k-1$, pod jehož kořen jsou dle definice napojeny binomiální stromy řádů $0, \ldots, k-2$. Pod kořen B_k jsou tudíž napojeny binomiální stromy řádů $0, \ldots, k-1$, tedy strom B_k je binomiální strom řádu k. □

Lemma V (vlastnosti binomiálních stromů): Binomiální strom T řádu k má 2^k vrcholů, které jsou rozděleny do $k+1$ hladin. Kořen stromu má právě k synů.

Důkaz: Počet synů kořene plyne přímo z definice binomiálního stromu, zbytek dokážeme matematickou indukcí. Pro $k = 0$ má T jistě 1 hladinu a $2^0 = 1$ vrchol. Uvažme $k > 0$. Z indukčního předpokladu vyplývá, že binomiální strom řádu $k-1$ má k hladin a 2^{k-1} vrcholů. Užitím lemmatu **R** dostáváme, že strom T je složený ze dvou stromů B_{k-1}, z nichž jeden je o hladinu níže než druhý, což dává hloubku $k+1$ stromu T. Složením dvou stromů 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.

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

1. Každý strom T_i je binomiální strom.
2. Uchovávané prvky jsou uloženy ve vrcholech stromů T_i. Klíč uložený ve vrcholu $v \in V(T_i)$ značíme $k(v)$.
3. Pro každý strom $T_i \in \mathcal{T}$ platí *haldové uspořádání,* neboli pro každý vrchol $v \in V(T_i)$ a libovolného jeho syna s je $k(v) \leq k(s)$.
4. V souboru $\mathcal{T}$ se žádný řád stromu nevyskytuje více než jednou.
5. Soubor stromů $\mathcal{T}$ je uspořádán vzestupně podle řádu stromu.

Jako vhodný způsob uložení souboru stromů $\mathcal{T}$ tedy poslouží například spojový seznam. Každý vrchol stromu si též bude pamatovat spojový seznam svých synů a řád podstromu, jehož je kořenem.

Lemma D (o dvojkovém zápisu): Binomiální strom řádu k se vyskytuje v souboru stromů n-prvkové binomiální haldy právě tehdy, když je ve dvojkovém zápisu čísla n k-tý nejnižší bit roven 1.

Důkaz: Z definice binomiální haldy vyplývá, že binomiální stromy dohromady obsahují $\sum_{i=0}^{k} b_i 2^i = n$ vrcholů, kde k je maximální řád stromu v $\mathcal{T}$ a $b_i \in \{0, 1\}$. Číslo $b_k b_{k-1} \ldots b_0$ tedy tvoří zápis čísla n v dvojkové soustavě. Z vlastností zápisu čísla ve dvojkové soustavě plyne, že pro dané n jsou čísla b_i (a tím i řády binomiálních stromů v $\mathcal{T}$) určena jednoznačně. □

Z lemmatu **V** nebo **D** plyne následující důležitá vlastnost:

Důsledek: Binomiální halda s n prvky sestává z nejvýše $\lceil \log_2 n \rceil$ binomiálních stromů.

Cvičení

1. Binomiální stromy vděčí za svůj název následující vlastnosti: Počet prvků na i-té hladině (číslujeme od 0) binomiálního stromu řádu k je roven kombinačnímu číslu neboli binomiálnímu koeficientu $\binom{k}{i}$. Dokažte.

2. Kolik má n-prvková binomiální halda listů?
3. Dokažte, že libovolné přirozené číslo x lze zapsat jako konečný součet mocnin dvojky $2^{k_1} + 2^{k_2} + \ldots$ tak, že $k_i \neq k_j$ pro různá i, j. Ukažte, že v tomto součtu figuruje nejvýše $\lceil \log_2 x \rceil$ sčítanců.
4. Na binomiální strom řádu k lze pohlížet jako na speciální kostru grafu k-rozměrné hyperkrychle. A to ne ledajakou, ale dokonce je to strom nejkratších cest. Dokažte například užitím cvičení 1.

1.2. Operace s binomiální haldou

Nalezení minima

Jak jsme již ukázali u binární haldy, pokud strom splňuje haldovou podmínku, prvek s nejmenším klíčem se nachází v kořeni stromu. Minimum celé binomiální haldy se proto musí nacházet v kořeni jednoho ze stromů v $\mathcal{T}$. Operaci MIN tedy postačí projít seznam $\mathcal{T}$, což potrvá čas $\Theta(\log n)$. Pokud bychom tuto operaci chtěli volat často, můžeme ji urychlit na $\Theta(1)$ tím, že budeme během všech operací udržovat ukazatel na globální minimum. Tuto údržbu v ostatních algoritmech explicitně nepopisujeme, nýbrž ji přenecháváme čtenáři do cvičení 3.

Slévání

Operaci MERGE poněkud netypicky popíšeme jako jednu z prvních, protože ji budeme nadále používat jako podproceduru ostatních operací. Algoritmus slévání vezme dvě binomiální haldy H_1 a H_2 a vytvoří z nich jedinou binomiální haldu H, jež obsahuje prvky obou hald.

Nejprve popíšeme spojení dvou binomiálních stromů stejného řádu. Při něm je potřeba napojit kořen jednoho stromu jako posledního syna kořene druhého stromu. Přitom musíme dát pozor, aby zůstalo zachováno haldové uspořádání, takže vždy zapojujeme kořen s větším prvkem pod kořen s menším prvkem. Vznikne binomiální strom o jedna vyššího řádu.

Procedura MERGEBINOMTREES (spojení binomiálních stromů)
Vstup: Stromy B_1, B_2 z binomiální haldy ($\check{r}\acute{a}d(B_1) = \check{r}\acute{a}d(B_2)$)

1. Pokud $k(ko\check{r}en(B_1)) \leq k(ko\check{r}en(B_2))$:
2. Připojíme $ko\check{r}en(B_2)$ jako posledního syna pod $ko\check{r}en(B_1)$.
3. $B \leftarrow B_1$
4. Jinak:
5. Připojíme $ko\check{r}en(B_1)$ jako posledního syna pod $ko\check{r}en(B_2)$.
6. $B \leftarrow B_2$
7. $\check{r}\acute{a}d(B) \leftarrow \check{r}\acute{a}d(B) + 1$

Výstup: Výsledný strom B

Algoritmus BHMERGE svým průběhem bude připomínat algoritmus „školního“ sčítání čísel pod sebou, byť ve dvojkové soustavě.

Při sčítání dvojkových čísel procházíme obě čísla současně od nejnižšího řádu k nejvyššímu. Pokud je v daném řádu v obou číslech 0, píšeme do výsledku 0. Pokud je v jednom z čísel 0 a ve druhém 1, píšeme 1. A pokud se setkají dvě 1, píšeme 0 a přenášíme 1 do vyššího řádu. Díky přenosu se pak ve vyšším řádu mohou setkat až tři 1, a tehdy píšeme 1 a posíláme přenos 1.

Podobně pracuje slévání binomiálních hald: binomiální strom řádu i se chová jako číslice 1 na i-tém řádu čísla. Procházíme tedy oběma haldami od nejnižšího řádu k nejvyššímu a kdykoliv se setkají dva stromy téhož řádu, sloučíme je pomocí MergeBinomTrees, což vytvoří strom o jedna vyššího řádu, čili přenos.

Protože udržujeme soubory stromů hald uspořádané dle řádu binomiálních stromů, lze algoritmus realizovat průchodem dvěma ukazateli po těchto seznamech, jako když sléváme setříděné posloupnosti. Řády stromů nepřítomné v haldě při tom přirozeně přeskakujeme.

Procedura BhMerge (slévání binomiálních hald)
Vstup: Binomiální haldy A, B

1. Založíme prázdnou haldu C.
2. $p \leftarrow$ *nedefinováno* ◁ *přenos do vyššího řádu*
3. Dokud A i B jsou neprázdné nebo je p definováno:
4. $\quad r_a \leftarrow$ nejnižší řád stromu v A, nebo $+\infty$ pro $A = \emptyset$
5. $\quad r_b \leftarrow$ nejnižší řád stromu v B, nebo $+\infty$ pro $B = \emptyset$
6. $\quad r_p \leftarrow$ řád stromu p (nebo ∞, pokud p není definováno)
7. $\quad r \leftarrow \min(r_a, r_b, r_p)$
8. $\quad$ Pokud $r = +\infty$, skončíme. ◁ *už není co slévat*
9. $\quad S \leftarrow \emptyset$ ◁ *sem uložíme stromy, které budeme spojovat*
10. $\quad$ Pokud $r_a = r$, přesuneme strom nejnižšího řádu z A do S.
11. $\quad$ Pokud $r_b = r$, přesuneme strom nejnižšího řádu z B do S.
12. $\quad$ Pokud p je definováno, přidáme p do S. ◁ *musí být* $r_p = r$
13. $\quad$ Pokud $|S| \geq 2$:
14. $\quad\quad$ Odebereme dva stromy z S a označíme je x a y.
15. $\quad\quad p \leftarrow$ MergeBinomTrees(x, y)
16. $\quad$ Pokud v S zbývá nějaký strom, přesuneme ho na konec haldy C.
17. Pokud je A nebo B neprázdná, připojíme ji na konec haldy C.

Výstup: Binomiální halda C poskládaná z prvků A a B

Pozorování: Algoritmus BhMerge je korektní a jeho časová složitost je $\Theta(\log n)$.

Vkládání prvků a postavení haldy

Operaci Insert 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 $\Theta(\log n)$.

Procedura BhInsert (vkládání do binomiální haldy)
Vstup: Binomiální halda H, vkládaný prvek x

1. Vytvoříme binomiální haldu H' s jediným prvkem x.
2. $H \leftarrow \text{BhMerge}(H, H')$

Výstup: Binomiální halda H s vloženým prvkem x

Tvrzení: Operace Insert má časovou složitost $\Theta(\log n)$. Pro zpočátku prázdnou binomiální haldu trvá libovolná posloupnost k volání operace Insert čas $\Theta(k)$.

Důkaz: Jednoprvkovou haldu lze 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.

Zde je však jedna ze slévaných hald jednoprvková. V nejhorším případě se samozřejmě může stát, že původní halda obsahuje 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. Ukážeme ovšem, že se to nemůže dít často.

Využijeme skutečností dokázaných pro binární počítadlo v kapitole **??**. Připomeňme, že provedeme-li posloupnost k inkrementů na počítadle, které bylo zpočátku nulové, strávíme celkově čas $\mathcal{O}(k)$. Opakované volání BhInsert je ekvivalentní opakovanému inkrementu binárního počítadla. Operace součtu dvou jedničkových bitů potom odpovídá operaci slití dvou binomiálních stromů. Při použití procedury BhMerge stačí odhadnout pouze maximální počet volání MergeBinomTrees, z čehož vyplývá celková časová složitost $\Theta(k)$ pro k volání procedury BhInsert. □

Analýza operace Insert dává návod na realizaci rychlé operace Build pro postavení binomiální haldy: opakovaně voláme na zpočátku prázdnou binomiální haldy operaci Insert.

Důsledek: Časová složitost operace Build pro n prvků je $\Theta(n)$.

Na rozdíl od binární haldy, jejíž rychlá stavba vyžadovala speciální postup, zde pro rychlé postavení binomiální haldy stačilo pouze lépe analyzovat časovou složitost Insertu.

Odstranění minima

Při odstraňování minima z binomiální haldy H opět využijeme operaci Merge. Nejprve průchodem souboru stromů najdeme binomiální strom M, jehož kořen je minimem haldy H, a tento strom z H odpojíme. Následně ze stromu M odtrhneme kořen a všechny jeho syny (včetně jejich podstromů) vložíme do nové binomiální haldy H'. Tato operace je poměrně jednoduchá, neboť se mezi syny dle definice binomiálního stromu nevyskytují dva stromy stejného řádu. Nakonec slijeme H s H', čímž se odtržené prvky začlení zpět.

Procedura BhExtractMin (odebrání minima z binomiální haldy)
Vstup: Binomiální halda H

1. $M \leftarrow$ strom s nejmenším kořenem v haldě H

2. $m \leftarrow k(\textit{kořen}(M))$
3. Odebereme M z H.
4. Vytvoříme prázdnou binomiální haldu H'.
5. Pro každého syna s kořene stromu M:
6. Odtrhneme podstrom s kořenem s a vložíme jej do H'.
7. Zrušíme M. ◁ *zbude jen kořen, který zrušíme*
8. $H \leftarrow \textsc{BhMerge}(H, H')$

Výstup: Binomiální halda H s odstraněným minimem m

Tvrzení: Časová složitost operace BhExtractMin v n-prvkové binomiální haldě je $\Theta(\log n)$. Libovolná korektní implementace operace BhExtractMin má časovou složitost $\Omega(\log n)$.

Důkaz: Nalezení minima trvá čas $\mathcal{O}(\log n)$, protože v souboru stromů je jich jen logaritmicky mnoho. Vytvoření haldy H' 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 $\Theta(\log n)$.

Dolní odhad složitosti odstraňování minima získáme z dolního odhadu složitosti třídění velmi podobně, jako jsme podobnou skutečnost dokázali u binární haldy (viz kapitola **??**). Kdyby existoval algoritmus na odstranění minima s časovou složitostí lepší než $\Theta(\log n)$, zkonstruovali bychom třídicí algoritmus takto: Vložili bychom n tříděných prvků operací Build do haldy a následně n-násobným odstraněním minima vypsali uspořádanou posloupnost. Tento algoritmus by však měl časovou složitost lepší než $\Theta(n \log n)$, což je spor s dolním odhadem složitosti třídění. □

V úvodu této kapitoly jsme zmínili ještě operace Decrease, Delete a Increase, které dostanou ukazatel na binomiální haldu a ukazatel na prvek v ní a provedou po řadě snížení klíče prvku, smazání prvku a zvýšení klíče. Tyto operace přenecháme čtenáři jako cvičení 4, 5 a 6.

Cvičení

1. Přeformulujte všechny definice a operace pro maximovou binomiální haldu.
2. Rozmyslete detaily reprezentace binomiální haldy ve vašem oblíbeném programovacím jazyce tak, aby byla zachována časová složitost všech operací.
3. U binomiální haldy lze minimum (přesněji referenci na kořen obsahující minimum) udržovat stranou, abychom k němu mohli přistupovat v konstantním čase. Upravte všechny operace tak, aby zároveň udržovaly odkaz na minimum a nezhoršily se jejich časové složitosti.
4. Navrhněte operaci Decrease s časovou složitostí $\Theta(\log n)$. Nezapomeňte, že bude třeba do reprezentace haldy v paměti doplnit další ukazatele a ty udržovat.
5. Navrhněte operaci Delete s časovou složitostí $\Theta(\log n)$.
6. Navrhněte operaci Increase s časovou složitostí $\Theta(\log n)$.

7.* Pokuste se definovat *trinomiální haldu,* jejíž stromy budou mít velikost mocnin trojky. Od každého řádu se přitom v haldě budou smět vyskytovat až dva stromy. Domyslete operace s touto haldou a srovnejte jejich složitosti s haldou binomiální.

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 odstraňování minima, dokud nejsou opravdu potřeba. Ukážeme, že tento postup sice zhorší časovou složitost v nejhorším případě, ale amortizovaně bude odebírání minima nadále logaritmické, a některé další operace dokonce konstantní.

Definice líné binomiální halda se téměř neliší od pilné. Pouze povolíme, že se v souboru stromů může vyskytovat více stromů stejného řádu. Reprezentace struktury v paměti bude stejná jako u pilné haldy, tedy pomocí spojových seznamů. Navíc se bude hodit, aby seznamy byly obousměrné a kruhové.

Operace slití dvou hald, kterou využívá vkládání prvku i vypuštění minima, se značně zjednoduší. Vzhledem k tomu, že se řády stromů mohou v haldě opakovat, slití realizujeme spojením seznamů stromů obou hald, což jistě zvládneme v konstantním čase.

Procedura LazyBhMerge (slévání líných binomiálních hald)
Vstup: Líné binomiální haldy H_1, H_2

1. Založíme novou haldu H.
2. Seznam stromů v $H \leftarrow$ spojení seznamů stromů v H_1 a H_2.

Výstup: Líná binomiální halda H poskládaná z prvků H_1 a H_2

Operace Insert pro vložení nového prvku do haldy je opět realizována jako slití haldy s novou, jednoprvkovou haldou.

Aby halda nezdegenerovala v obyčejný spojový seznam, musíme čas od času provést tzv. *konsolidaci* a stromy sloučit tak, aby jich bylo v seznamu co nejméně. Nejvhodnější čas na tento úklid je při mazání minima, které provedeme podobně jako u pilné binomiální haldy: Odtrhneme minimální kořen, z jeho synů uděláme haldu a tu následně výše popsaným způsobem slijeme s původní haldou. Na závěr provedeme následující konsolidaci.

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. Již však nemůžeme činit žádné předpoklady o počtech stromů v jednotlivých přihrádkách.

Proto budeme procházet přihrádky od nejnižšího řádu k nejvyššímu a kdykoliv v některé objevíme více stromů, budeme je odebírat po dvojicích a slučovat. Sloučené stromy budou mít o jedna vyšší řád, takže je přehodíme o přihrádku výše a vyřešíme v následujícím kroku. Nakonec tedy v každé přihrádce zbude nejvýše jeden strom.

Procedura LazyBhConsolidation (konsolidace líné binomiální haldy)
Vstup: Líná binomiální halda H o n prvcích

1. Připravíme pole $P[0 \dots \lceil \log n \rceil]$ spojových seznamů.
2. Pro každý strom T v H:
3. Odtrhneme T z H.
4. Vložíme T do $P[\check{r}\acute{a}d(T)]$.
5. Pro všechna $i = 0, \dots, \lceil \log n \rceil$:
6. Opakujeme, dokud jsou v $P[i]$ alespoň dva stromy:
7. Odtrhneme dva stromy B_1, B_2 z $P[i]$.
8. $B \leftarrow$ MergeBinomTrees(B_1, B_2)
9. Vložíme B do $P[i+1]$.
10. Pokud v $P[i]$ zbyl strom T:
11. Přesuneme T z $P[i]$ do H.

Výstup: Zkonsolidovaná halda H

Nyní můžeme přesněji popsat mazání minima.

Procedura LazyBhExtractMin (odebrání minima z líné binomiální haldy)
Vstup: Líná binomiální halda H

1. $M \leftarrow$ strom s nejmenším kořenem v haldě H
2. $m \leftarrow k(kořen(M))$
3. Odebereme M z H.
4. Odtrhneme seznam S podstromů kořene stromu M.
5. Připojíme S do seznamu stromů H.
6. Zrušíme M. ◁ *zbude jen kořen, který zrušíme*
7. $H \leftarrow$ LazyBhConsolidation(H)

Výstup: Líná binomiální halda H s odstraněným minimem m

Analýza líné binomiální haldy

Pro účely amortizované analýzy zvolíme potenciál $\Phi = c_\Phi \cdot t$, kde t je celkový počet stromů ve všech haldách, kterých se naše operace týkají, a c_Φ je vhodná konstanta, kterou určíme později. Pro ilustraci a analýzu penízkovou metodou si můžeme představit, že na každém stromu leží položeno c_Φ mincí. Označíme Φ_i hodnotu potenciálu po provedení i-té operace. Zjevně platí, že $\Phi_0 \leq \Phi_k$, kde k je počet provedených operací, neboť na počátku jsou všechny struktury prázdné a na konci je ve strukturách nezáporný počet stromů.

Lemma: Uvažme volání procedury LazyBhConsolidation(H) pro haldu H s nejvýše n prvky. Jeho skutečná cena je $\Theta(\log n + \textit{počet stromů } H)$ a její amortizovaná cena je $\mathcal{O}(\log n)$ vzhledem k potenciálu Φ.

Důkaz: Označme t počet stromů H před provedením konsolidace, A amortizovanou cenu konsolidace a C skutečnou cenu konsolidace. Inicializace pole P zabere čas $\Theta(\log n)$, stejně tak průchod přihrádkami. Vkládání stromů do P trvá lineárně

s počtem stromů v H. Stejně tak jejich následné spojování, neboť každým spojením ubude jeden strom. Je tedy $C \leq c_1(\log n + t)$ pro jistou konstantu c_1.

Abychom ukázali, že amortizovaná cena A konsolidace je logaritmická, stačí ověřit $A = C + \Delta\Phi = \mathcal{O}(\log n)$, kde $\Delta\Phi$ je změna potenciálu. Označme t' počet stromů v H po provedení konsolidace; zřejmě $t' \leq \lceil \log_2 n \rceil$. Tedy platí $A \leq c_1(\log n + t) + c_\Phi(t' - t) \leq c_\Phi(\log n + t) + c_\Phi(t' - t) = \mathcal{O}(\log n)$, za předpokladu že $c_\Phi \geq c_1$. □

Lemma: Uvažme volání procedur LazyBhMerge(H_1, H_2) a LazyBhInsert(H, x). Jejich skutečná cena je $\Theta(1)$ a jejich amortizovaná cena je $\Theta(1)$ vzhledem k potenciálu Φ.

Důkaz: Během slévání dvou hald se provede pouze konstantně mnoho operací, tedy skutečná cena je konstantní. Změna potenciálu je nulová, protože celkový počet stromů se nezmění, a tedy amortizovaná cena je též konstantní.

Vkládání do haldy nejprve založí jednoprvkový strom, což má konstantní skutečnou cenu a zvýší potenciál o c_Φ, takže amortizovaná cena je též konstantní. Poté provedeme slévání, čímž skutečnou ani amortizovanou cenu nezhoršíme. □

Lemma: Uvažme volání procedury LazyBhExtractMin(H), kde H má nejvýše n prvků. Jeho skutečná cena je $\Theta(\log n + \textit{počet stromů } H)$ a jeho amortizovaná cena je $\mathcal{O}(\log n)$ vzhledem k potenciálu Φ.

Důkaz: Označme nejprve s počet synů kořene stromu M, které procedura přepojuje. Jelikož řády stromů jsou nejvýše logaritmické, je s také $\mathcal{O}(\log n)$.

Volbu minimálního stromu M si necháme na konec důkazu. Odtržení M z H, připojení synů kořene M do H a zrušení M lze realizovat za konstantní skutečnou cenu. Potenciál se při tom zvýší o $s - 1$. Amortizovaná cena tedy vyjde $\mathcal{O}(\log n)$.

Označme t počet stromů H a t' počet stromů po provedení konsolidace; je tedy $t' = \mathcal{O}(\log n)$. Následná konsolidace má podle předchozích lemmat skutečnou cenu $\Theta(\log n + t + s) = \Theta(\log n + t) \leq c_3(\log n + t)$ pro nějakou konstantu c_3. Konečně započítáme také volbu minimálního stromu. Tu lze realizovat ve skutečném čase $\Theta(t)$, její skutečná cena je tedy nejvýše $c_2 t$ pro nějakou konstantu c_2. Stejně jako u konsolidace se tato cena sečte s poklesem potenciálu a vyjde amortizovaná cena $\mathcal{O}(\log n)$. Přesněji, pro amortizovanou cenu A platí $A \leq c_2 t + c_3(\log n + t) + c_\Phi(t' - t) \leq c_\Phi(\log n + t) + c_\Phi(t' - t) = \mathcal{O}(\log n)$, za předpokladu $c_\Phi \geq c_2 + c_3$. □

Nyní zbývá stanovit, jaká bude konstanta c_Φ v definici potenciálu Φ: stačí zvolit $c_\Phi = \max(c_1, c_2 + c_3)$ z předchozích důkazů. Poznamenejme ještě, že počet stromů líné binomiální haldy může být $\Theta(n)$ v nejhorším případě, a proto je složitost operace LazyBhExtractMin v nejhorším případě taktéž lineární.

Shrňme na závěr složitosti zmiňovaných operací líné binomiální haldy. Z nich zbývá ukázat ještě dolní odhad amortizované složitosti ExtractMin, který přenecháme do cvičení 3.

operace	*worst-case čas*	*amortizovaný čas*
INSERT	$\Theta(1)$	$\Theta(1)$
EXTRACTMIN	$\Theta(n)$	$\Theta(\log n)$
MERGE	$\Theta(1)$	$\Theta(1)$

Význam líné binomiální haldy vzroste především v dalším oddílu, 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 a udržovali zároveň ukazatel na poslední prvek seznamu.
2. Navrhněte operace MIN, DECREASE, DELETE a INCREASE pro línou binomiální haldu. Jaká bude jejich skutečná a amortizovaná cena?
3. Ukažte, že složitost EXTRACTMIN musí nutně být $\Omega(\log n)$, a to jak worst-case, tak amortizovaně.
4. *Konsolidace párováním:* Konsolidaci se můžeme pokusit zjednodušit tak, že po spojení dvou stromů stejného řádu výsledný strom rovnou umístíme do výsledné haldy. Může se tedy stát, že ve zkonsolidované haldě se budou řády stromů opakovat. Dokažte, že to neuškodí amortizované složitosti operací vůči potenciálu Φ.

1.4. Fibonacciho haldy

Budeme pokračovat v myšlence dalšího „zlíňování“ již tak dost líné binomiální haldy. Hlavním prostředkem bude zvolnění požadavku na strukturu stromů, ze kterých halda sestává.

Definice: *Fibonacciho halda* pro danou množinu prvků se skládá ze souboru stromů $\mathcal{T} = T_1, \ldots, T_\ell$, kde:

1. Uchovávané prvky jsou uloženy ve vrcholech stromů T_i. Klíč uložený ve vrcholu $v \in V(T_i)$ značíme $k(v)$.
2. Pro každý strom $T_i \in \mathcal{T}$ platí *haldové uspořádání,* neboli pro každý vrchol $v \in V(T_i)$ a libovolného jeho syna s je $k(v) \leq k(s)$.
3. Pro práci se strukturou se používají výhradně operace popsané níže.

Omezení na tvar stromů tentokrát neplynou přímo z definice, nýbrž z chování jednotlivých operací.

Soubor stromů a interní reprezentaci stromů budeme opět udržovat v kruhových spojových seznamech. Kromě toho si každý vrchol bude pamatovat svůj *řád,* tentokrát definovaný přímo jako počet synů vrcholu. *Řádem stromu* se rozumí řád jeho kořene.

Operace

Fibonacciho halda podporuje následující operace. Uvádíme u nich časové složitosti v nejhorším případě i amortizované, vše vzhledem k aktuálnímu počtu prvků n.

operace	*nejhůře*	*amortizovaně*	*činnost*
INSERT	$\Theta(1)$	$\Theta(1)$	vloží nový prvek
MIN	$\Theta(1)$	$\Theta(1)$	vrátí minimum množiny
EXTRACTMIN	$\Theta(n)$	$\Theta(\log n)$	vrátí a odstraní minimum
MERGE	$\Theta(1)$	$\Theta(1)$	sloučí dvě haldy do jedné
BUILD	$\Theta(n)$	$\Theta(n)$	postaví z n prvků haldu
DECREASE	$\Theta(n)$	$\Theta(1)$	sníží hodnotu klíče prvku
DELETE	$\Theta(n)$	$\Theta(\log n)$	smaže prvek

Základní operace Fibonacciho haldy se téměř neliší od haldy binomiální.

- MIN realizujeme tak, že budeme udržovat ukazatel na minimální kořen stromu ze souborů stromů haldy a v ostatních operacích budeme tento ukazatel přepočítávat. V operacích tento přepočet nezmiňujeme, nýbrž ho přenecháváme čtenáři k rozmyšlení do cvičení 2.
- MERGE pouze zřetězí spojové seznamy obou hald.
- INSERT vyrobí jednovrcholovou haldu s novým prvkem a tuto novou haldu spojí voláním MERGE s haldou původní. Všechny předchozí operace mají zjevně worst-case časovou složitost $\Theta(1)$.
- Operaci BUILD provedeme jako n-násobné volání INSERT s celkovou složitostí $\Theta(n)$.

Operaci EXTRACTMIN provedeme takřka stejně jako u líné binomiální haldy. Najdeme a odtrhneme kořen s minimálním klíčem, seznam jeho podstromů prohlásíme za novou Fibonacciho haldu a tu připojíme k haldě původní. Potom provedeme konsolidaci: setřídíme přihrádkově stromy dle jejich řádů a následně pospojujeme stromy stejných řádů tak, aby od každého řádu zbyl nejvýše jeden strom. Spojení stromů funguje analogicky ke spojování binomiálních stromů, tedy pod kořen s menším prvkem přivěsíme druhý strom, čímž řád vzroste o 1.

Hlavním rozdílem oproti binomiální haldě a zároveň motivací pro použití Fibonacciho haldy je zcela jinak fungující operace DECREASE pro snížení hodnoty klíče. Pokud dojde ke snížení klíče, může vzniknout porucha v uspořádání haldy mezi modifikovaným prvkem a jeho otcem. U binární nebo binomiální haldy bychom tuto poruchu vyřešili vybubláním sníženého klíče nahoru. Zde však podstrom zakořeněný v prvku se sníženým klíčem odtrhneme a vložíme do spojového seznamu mezi ostatní stromy haldy. Tomuto odtržení a přesunutí budeme říkat operace CUT.

Všimněme si, že pokud bychom používali pouze operace vkládání prvku a odstraňování minima, vznikaly by v souboru stromů haldy pouze binomiální stromy. Opakované snižování klíčů v jednom stromu by však mohlo způsobit, že by strom degeneroval na strom s nevhodnými vlastnostmi, protože by vrcholy mohly mít odtrženo příliš mnoho svých synů. Každého otce, jemuž byl odtržen jeden syn, proto

označíme. Pokud byl odtržen syn vrcholu v, který byl již předtím označený, zavoláme CUT i na v. To může vyústit v kaskádovité odtrhávání podstromů, dokud nenarazíme na neoznačený vrchol, případně kořen.

Algoritmus udržuje invariant, že kořeny stromů nikdy nejsou označené. Operace EXTRACTMIN s tímto invariantem musí počítat a musíme ji tedy ještě upravit: při zařazení odtržených podstromů do haldy z jejich kořenů odstraníme případné označení. Podobně u INSERTu je nově vytvořený prvek neoznačený.

Procedura FHDECREASE (snížení klíče prvku ve Fibonacciho haldě)
Vstup: Fibonacciho halda H, vrchol x, nový klíč t

1. $k(x) \leftarrow t$
2. Pokud je x kořen nebo $k(otec(x)) \leq k(x)$, skončíme.
3. Zavoláme FHCUT(x).

Výstup: Upravená halda H

Procedura FHCUT (odseknutí podstromu Fibonacciho haldy)
Vstup: Fibonacciho halda H, kořen x podstromu k odseknutí

1. $o \leftarrow otec(x)$
2. Odtrhneme x i s podstromem, odstraníme případné označení x a vložíme x do H.
3. Pokud o je označený, zavoláme FHCUT(o). ◁ *o jistě není kořen*
4. Jinak není-li o kořen, označíme ho.

Výstup: Upravená halda H

Konečně, operaci DELETE realizujeme snížením klíče mazaného prvku na $-\infty$ voláním DECREASE a následným voláním EXTRACTMIN.

Analýza Fibonacciho haldy

V analýze haldy budeme využívat Fibonacciho čísla F_n zavedená v oddílu **??**. Bude se nám hodit jejich následující vlastnost, jejíž důkaz přenecháme do cvičení 1.

Lemma: Pro Fibonacciho čísla platí:

$$1 + \sum_{i=0}^{d} F_i = F_{d+2}.$$

Značení: Označme T_v podstrom zakořeněný ve vrcholu v a $|T_v|$ počet jeho vrcholů.

Tvrzení: Po každé provedené operaci splňuje halda $\mathcal{T}$ definici Fibonacciho haldy a platí, že je-li strom $T \in \mathcal{T}$ řádu k, potom $|T| \geq F_{k+2}$.

Důkaz: Výsledná halda po operacích INSERT, MIN, MERGE splňuje definici Fibonacciho haldy, protože tyto operace nijak nemění strukturu stromů. Ze stejného důvodu je po předchozích operacích zachován vztah pro velikost stromu. Uvažme nyní operace EXTRACTMIN a DECREASE.

Zvolme vrchol v z libovolného stromu haldy. Indukcí podle hloubky T_v ukážeme, že $|T_v| \geq F_{k+2}$, kde k je řád v. Jestliže T_v má hloubku 0, je $|T_v| = 1 = F_2$. Předpokládejme dále, že T_v má kladnou hloubku a řád $k > 0$. Označme $x_1, \ldots, x_k$ syny vrcholu v v pořadí, v jakém byly vrcholy připojeny pod v (x_1 první, x_k poslední), a označme $r_1, \ldots, r_k$ jejich řády. Dokážeme, že $r_i \geq i - 2$ pro každé $2 \leq i \leq k$. Než byl x_i připojen pod v, byly $x_1, \ldots, x_{i-1}$ už syny vrcholu v a tedy v měl řád alespoň $i - 1$. Stromy jsou spojovány jen tehdy, mají-li stejný řád, musel tedy také x_i mít v okamžiku připojení pod v řád alespoň $i - 1$. Od té doby mohl x_i ztratit pouze jednoho syna, což zaručuje mechanismus označování vrcholů, a tedy $r_i \geq i-2$.

Jelikož hloubky všech T_{x_i} jsou ostře menší než hloubka T_v, z indukčního předpokladu plyne $|T_{x_i}| \geq F_{r_i+2} \geq F_{(i-2)+2} = F_i$. Vrcholy v a x_1 přispívají do $|T_v|$ každý alespoň 1. Dostáváme tak

$$|T_v| \geq 2 + \sum_{i=2}^{k} |T_{x_i}| \geq 2 + \sum_{i=2}^{k} F_i = 1 + \sum_{i=0}^{k} F_i = F_{k+2}.$$

Korektnost operace DELETE plyne jednoduše z korektnosti předchozích operací. □

Z předchozího tvrzení a toho, že Fibonacciho čísla rostou alespoň tak rychle jako funkce 1.618^n (viz cvičení **??**), plyne následující důsledek.

Důsledek: Řád každého stromu ve Fibonacciho haldě je nejvýše $\lceil \log_{1.618} n \rceil$.

Nyní přistoupíme k amortizované analýze. Jako potenciál zvolíme $\Phi = c_\Phi(t + 2m)$, kde t je celkový počet stromů ve všech haldách, kterých se operace týkají, m je celkový počet označených vrcholů v nich a c_Φ je konstanta, kterou určíme později. Pro představu analýzou pomocí penízkové metody: na každém stromu bude položeno c_Φ mincí a na každém označeném vrcholu $2c_\Phi$ mincí.

Lemma: Uvažme volání procedur FHMIN, FHMERGE a FHINSERT. Jejich skutečná cena i amortizovaná cena jsou $\mathcal{O}(1)$ vzhledem k potenciálu Φ.

Důkaz: FHMIN pouze vrací zapamatované minimum a potenciál nemění. Během slévání hald se provede jen konstantně mnoho operací, takže skutečná cena je konstantní. Amortizovaná cena také, neboť potenciál se nemění (počet stromů se nemění, nic se neoznačuje). Konečně FHINSERT kromě slévání vytvoří jednoprvkovou haldu, což stojí konstantní skutečný čas a zvýší potenciál o konstantu. Amortizovaná cena je proto opět konstantní. □

Z předchozí analýzy FHINSERT snadno vyplyne analýza FHBUILD.

Důsledek: Uvažme volání procedury FHBUILD pro n prvků. Jeho skutečná i amortizovaná cena je $\mathcal{O}(n)$.

Lemma: Uvažme volání procedury FHEXTRACTMIN(H), kde H má n prvků. Potom jeho skutečná cena je $\Theta(\log n + \textit{počet stromů } H)$ a jeho amortizovaná cena je $\mathcal{O}(\log n)$ vzhledem k potenciálu Φ.

Důkaz: Označme t počet stromů H před odstraněním minima, A amortizovanou cenu a C skutečnou cenu. Volbu stromu M s nejmenším kořenem lze realizovat v čase

nejvýše $c_1 t$ pro vhodnou konstantu c_1. Odtržení M z H, připojení seznamu synů M do H a zrušení M lze realizovat za konstantní skutečnou cenu. Odznačení synů M trvá čas lineární k jejich počtu, což je $\mathcal{O}(\log n)$. Při konsolidaci zabere inicializace pole přihrádek čas $\Theta(\log n)$, stejně tak průchod přihrádkami, jelikož maximální řád stromu je $\mathcal{O}(\log n)$. Zbývajících operací, tedy vkládání stromů do přihrádek a jejich následné spojování, je $\Theta(\log n + t)$. Je tedy $C \leq c_2(\log n + t)$ pro nějakou konstantu c_2.

Abychom ukázali, že amortizovaná cena A je logaritmická, stačí ověřit $A = C + \Delta\Phi = \mathcal{O}(\log n)$, kde $\Delta\Phi$ je změna potenciálu. Označme t' počet stromů v H a m' počet označených vrcholů po provedení konsolidace. Nový počet stromů $t' \leq c_3 \log n$ pro vhodnou konstantu c_3 a $m' \leq m$, jelikož se neoznačují žádné další vrcholy. Tedy za předpokladu $c_1 + c_2 + c_3 \leq c_\Phi$ platí $A \leq c_1 t + c_2(\log n + t) + c_3 t' + 2m' - c_\Phi(t + 2m) \leq c_\Phi(\log n + t) + c_\Phi(t' + 2m') - c_\Phi(t + 2m) = \mathcal{O}(\log n)$. □

Z předchozího důkazu vidíme, že v definici potenciálu Φ stačí zvolit konstantu $c_\Phi \geq c_1 + c_2 + c_3$. Zřejmě v nejhorším případě může v n-prvkové haldě FhExtractMin trvat čas $\Theta(n)$.

Lemma: Uvažme volání procedury FhCut(v), kde v je odsekávaný vrchol. Označme ℓ počet nových stromů, které operace při svém běhu vytvoří. Potom skutečná cena operace je $\mathcal{O}(\ell)$ a její amortizovaná cena je $\mathcal{O}(1)$ vzhledem k potenciálu Φ.

Důkaz: Odtržení jednoho podstromu a jeho vložení do haldy trvá konstantní čas. Skutečná cena C je tedy lineární s počtem nových stromů ℓ. FhCut udržuje invariant, že kořen stromu nikdy není označený. Každý z ℓ nových stromů možná s výjimkou prvního byl už před zahájením operace označený. Poté, co z kořenů podstromů vznikly kořeny stromů v haldě, byly odznačeny a nově označen mohl být jen jeden vrchol. Počet označených vrcholů m' po konci FhCut tedy bude $m' = m - (\ell - 1) + 1 = m - \ell + 2$. Změna potenciálu je tudíž $\Delta\Phi = c_\Phi(\ell + 2(m' - m)) = c_\Phi(\ell - 2\ell + 4) = c_\Phi(-\ell + 4)$. Skutečná cena je $C = \mathcal{O}(\ell)$, a tedy amortizovaná cena $A = \mathcal{O}(\ell) + \Delta\Phi = \mathcal{O}(1)$ za předpokladu dostatečně velké konstanty c_Φ v definici potenciálu Φ. □

Důsledek: Složitost procedury FhDecrease je stejná jako u procedury FhCut.

Důkaz: Snížení klíče obnáší kromě případného odseknutí podstromu pouze konstantní počet operací, aniž by se změnil potenciál. □

Poznámka: Po provedení FhDecrease může být v n-prvkové haldě až $\Theta(n)$ nových stromů. Tuto skutečnost přenecháváme k ověření čtenáři do cvičení 4.

Z analýzy FhDecrease a FhExtractMin také plyne analýza složitosti operace FhDelete.

Důsledek: Uvažme volání procedury FhDelete. Jeho skutečná cena je $\Theta(\log n + t + \ell)$, kde t je počet stromů haldy a ℓ počet stromů vzniklých při mazání vrcholu. Amortizovaná cena činí $\mathcal{O}(\log n)$ vzhledem k potenciálu Φ.

Na složitost FhDelete v nejhorším případě se vztahují stejné odhady jako u FhDecrease. Pro složitost některých operací jsme zatím ukázali pouze horní asymptotický odhad. Dolní odhady worst-case složitosti FhDecrease, a tedy

také FhDelete, přenecháme do cvičení 4, dolní odhady amortizované složitosti FhExtractMin, a tedy také FhDelete, přenecháme do cvičení 5. Tím bude analýza všech operací zmíněných v úvodu oddílu 1.4 kompletní.

Cvičení

1. Dokažte, že $1 + \sum_{i=0}^{d} F_i = F_{d+2}$.
2. Navrhněte operaci FhMin s konstantní časovou složitostí pomocí udržování ukazatele na nejmenší prvek v ostatních operacích. Bude třeba též znovu provést jejich (amortizovanou) časovou analýzu.
3. O zakořeněném stromu T řekneme, že má Fibonacciho vlastnost, pokud pro každý $v \in V(T)$ řádu k je $|T_v| \geq F_{k+2}$. Dokažte, že pro každé n existuje strom na n vrcholech, který má Fibonacciho vlastnost.
4. Ukažte, že existuje posloupnost $\mathcal{O}(n)$ operací na Fibonacciho haldě taková, že všech n prvků v haldě je uloženo v jediném stromu, všechny vrcholy až na jeden mají řád 1 a všechny vrcholy až na kořen jsou označené.
5. Dokažte, že při stávajících ostatních operacích nemůže existovat implementace FhExtractMin s lepším než logaritmickým amortizovaným časem.
6. Bylo by možné ve Fibonacciho haldě také realizovat operaci zvýšení klíče (pro minimovou haldu) tak, aby fungovala v lepším než logaritmickém čase?