1. Binární halda

Uvažme následující problém ze života: jsme velmi vytíženým manažerem, kterému se plánovací diář stále plní obrovským množstvím úkolů. Každý úkol má předepsanou prioritu a vždy je třeba začít pracovat na úkolu, který ma v daný okamžik nejvyšší prioritu. Úkoly tedy průběžně přibývají a jsou zpracovávány a odstraňovány ze seznamu. Pro řádově desítky úkolů je asi zvládnutelné je prostě nějak zapsat a při odebírání vyhledat ten s nejvyšší prioritou. Pro tisíce nebo biliony úkolů je takový postup těžko představitelný.

V následujícím textu popíšeme datovou strukturu, která nejen že bude schopná efektivně vyřešit manažerský problém, nýbrž bude mít podstatně širší využití.

1.1. Definice binární haldy

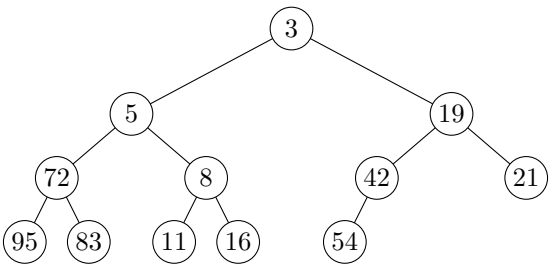
Halda je datová struktura uchovávající množinu N porovnatelných prvků (v našem případě pro názornost celých čísel) ve vrcholech binárního stromu splňujícího jisté speciální vlastnosti. Vrcholy odpovídají různým prvkům, pojmem *hodnota vrcholu* v (značeno $h(v)$) rozumíme prvek ve vrcholu v uložený. Haldu přesně zavedeme v následující definici.

Definice: *Minimová halda* je datová struktura uchovávající množinu N porovnatelných prvků ve vrcholech binárního stromu T, který:

1 má všechny hladiny plně obsazené, s výjimkou poslední,

2 v poslední hladině jsou všechny listy nalevo,

3 pro každý vrchol $v \in T$ a jeho (až) dva syny ℓ a r platí $h(v) \leq h(\ell)$ a $h(v) \leq h(r)$; tomuto axiomu říkáme *haldová podmínka*.



Obr. 1.1: Příklad haldy pro $N = 12$.

Podotýkáme, že podmínka 3 se vztahuje pouze na relaci mezi otcem a syny a nic neříká a relaci mezi levým a pravým synem.

Má smysl zavést též *maximovou haldu*, jejíž definice je naprosto stejná, pouze s opačnými relacemi v axiomu 3. Všechno, co v této kapitole ukážeme pro minimovou haldu platí i pro maximovou haldu, avšak s opačnými relacemi.

Shrňme jednoduché vlastnosti haldy.

Pozorování: Nechť T je minimová halda s N prvky. Potom

1. má $k = \lfloor \log_2 N \rfloor + 1 = \mathcal{O}(\log N)$ hladin,
2. každá cesta v T z kořene do listu obsahuje neklesající posloupnost prvků reprezentované množiny,
3. globální minimum reprezentované množiny je uloženo v kořeni T a je tedy přístupné v konstantním čase.

Důkaz: Vlastnost 1 plyne z vlastností vyvážených binárních stromů, které mají $\mathcal{O}(\log N)$ hladin. Vlastnost 2 je důsledkem haldové podmínky, která platí pro každý vrchol T; indukcí snadno nahlédneme platnost pro libovolně dlouhou cestu z kořene T. Vlastnost 3 je důsledkem vlastnosti 2. □

Implementace haldy v poli

Pozorný čtenář jistě postřehl, že jsme zatím k ničemu nevyužili axiom 2 definice haldy. Tento axiom skutečně pro všechny algoritmy nad haldou není nutný, vynikne však při jednoduché implementaci haldy v poli, jak nyní ukážeme.

Prvky haldy očíslujeme od 1 do N postupně po jednotlivých hladinách shora dolů a v hladině zleva doprava. Prvek číslo i umístíme do pole na index i. Díky axiomu 2 je pole souvisle zaplněné.

Všimněme si, že s tímto očíslováním je pro prvek na indexu i

- jeho otec na indexu $\lfloor i/2 \rfloor$,
- jeho levý syn na indexu $2i$,
- a jeho pravý syn na indexu $2i + 1$.

Kořen haldy tedy patří na index 1, znamená to tedy, že index 0 je v programovacích jazycích číslujících pole od 0 nevyužit. Můžeme ho však využít k zjednodušení některých algoritmů pro techniku zarážky. Poznamenejme také, že pokud dopředu neznáme počet prvků v haldě uložených, můžeme použít „nafukovací pole“, které umí přidávat nové prvky v amortizovaném čase $\mathcal{O}(1)$.

Implementace haldy v poli je jednoduchá, kompaktní, velice šikovná a rychlá. Algoritmy však přesto budeme popisovat pro uložení haldy ve stromu, protože je to o něco názornější.

1.2. Operace s haldou

Jak již bylo zmíněno, halda nabízí pohodlný přístup k nejmenšímu prvku v konstantním čase. Oproti tomu časová složitost vkládání a mazání prvků bude horší (dokonce to je nevyhnutelné, jak časem dokážeme).

Uvedeme nejdříve přehledovou tabulku podporovaných operací spolu s jejich časy.

Operace	*Čas*	*Komentář*
HeapInsert	$\Theta(\log N)$	Vloží nový prvek.
HeapGetMin	$\Theta(1)$	Vrátí minimum reprezentované množiny.
HeapExtractMin	$\Theta(\log N)$	Vrátí a odstraní minimum množiny.
HeapBuild	$\Theta(N)$	Postaví z N prvků haldu.
HeapChangeKey	$\Theta(\log N)$	Změní hodnotu klíče prvku.
HeapDelete	$\Theta(\log N)$	Smaže prvek.

Operace HeapInsert, HeapGetMin, HeapExtractMin a HeapBuild popisujeme v tomto textu. Operace HeapChangeKey a HeapDelete ponecháváme čtenáři jako cvičení 1.5.2 a 1.5.3.

Nejjednodušší operací je HeapGetMin, který také nevyžaduje hlubšího vysvětlování – prostě vrátíme hodnotu uloženou v kořeni stromu haldy, což zvládneme za čas $\mathcal{O}(1)$.

Vkládání do haldy

Při vkládání v poslední hladině haldy vytvoříme nový list, a to samozřejmě dle definice haldy na první volné pozici zleva. Do tohoto listu uložíme přidávaný prvek. Používáme značení v pro vrchol obsahující nově přidávaný prvek a $p(v)$ pro otce vrcholu v.

Nyní hrozí, že některý z axiomů haldy neplatí, konkrétně haldová podmínka, protože by mohl otec $p(v)$ mít vyšší klíč než v. V takovém případě mezi sebou prohodíme obsah v a $p(v)$. Pokud stále není splněna haldová podmínka, provádíme opět prohození v s $p(v)$, a to tak dlouho, až buďto haldová podmínka začala platit nebo s nový prvek dostal až do kořene stromu. Všimněme si, že prohazováním v a $p(v)$ se nemůže porušit haldová podmínka mezi druhým synem r vrcholu $p(v)$ a mezi v: již před prohozením byl klíč r větší než klíč jeho otce, výměnou za ještě menší prvek jsme tudíž nemohli poškodit haldovou podmínku.

Z vlastností zmíněného prohazovacího postupu vyplývá, že se vykoná nejvýše tolik operací, kolik je výška stromu haldy. Z toho plyne i odhad $\Theta(\log N)$ časové složitosti pro HeapInsert do N-prvkové haldy.

Algoritmus HeapInsert

Vstup: Halda H, vkládaný prvek x.

Výstup: Upravená halda H.

1. Pokud je poslední hladina H plně zaplněná, založ novou hladinu.
2. Do poslední hladiny přidej na první volnou pozici zleva nový list ℓ obsahující x.
3. $v \leftarrow \ell$
4. Dokud v není kořen a je $h(p(v)) > h(v)$:
5. Prohoď obsah v a $p(v)$.
6. $v \leftarrow p(v)$

Odstranění minima

Při odstraňování minima nejprve přesuneme prvek uložený v nejpravějším listu ℓ poslední hladiny do kořene a ℓ zrušíme. Tím jsme sice zrušili minimální prvek, nicméně v kořeni teď nejspíš bude porušená haldová podmínka. To vyřešíme podobným prohazovacím způsobem jako v Insertu, tentokrát však směrem shora dolů.

Při zabublání dolů musíme dát pozor na jednu věc. Zatím co směrem vzhůru jsme prvek porovnávali jen s otcem, opačným směrem se můžou nacházet až dva synové. Pravidlo, které nám vybere správného syna na porovnání, je jednoduché. Prvek vždy porovnáváme a poté případně i prohazujeme s menším z obou synů. Kdybychom ho prohodili s větším synem, došlo by k další poruše haldové podmínky, protože by se větší syn stal otcem menšího.

Algoritmus HeapExtractMin
Vstup: Halda H.
Výstup: Upravená halda H.

1. Do kořene H přesuň obsah nejpravějšího listu ℓ poslední hladiny.
2. Zruš ℓ.
3. $v \leftarrow$ kořen H
4. Dokud má v syny:
5. $\quad s \leftarrow$ menší z obou synů *(Při pouze jednom synovi je to snazší.)*
6. $\quad$ Pokud $h(s) \geq h(v)$, skončí cyklus.
7. $\quad$ Prohoď obsah v a s.
8. $\quad v \leftarrow s$

V nejhorším případě se může stát, že náhradní prvek, který jsme si půjčili z konce haldy, bude zabublávat až zpátky na poslední hladinu. Počet operací je tedy opět přímo úměrný výšce stromu a celková složitost bude $\Theta(\log N)$.

Postavení haldy

Nyní popíšeme, jak z neuspořádané množiny N prvků postavit korektní haldu. Máme na to dokonce už dostatek nástrojů: zavoláme N-krát operaci HeapInsert postupně na každý prvek, což tedy celkem za čas $\Theta(N \log N)$ haldu vyrobí. Ukážeme nyní, že existuje, poněkud překvapivě, rychlejší algoritmus na stavbu haldy, který pracuje pouze v čase $\Theta(N)$.

Nejprve vyrobíme N-vrcholový strom T dle podmínek tvaru haldy (axiomy 1 a 2) a do jeho vrcholů jednoduše bez dalšího uspořádání nějak nakopírujeme vstupní prvky.

Všimněme si, že každý izolovaný vrchol v nejspodnější hladině T je vlastně korektní halda – pro jednoprvkové stromy skutečně musí platit haldová podmínka. Zadíváme se nyní na předposlední hladinu T a bereme v úvahu izolované stromy, jejichž kořeny jsou vrcholy této hladiny – ty mají 2 hladiny. Zde již nebude triviálně platná haldová podmínka, budeme ji muset ručně obnovit. To však je stejný problém, jaký jsme již řešili v operaci HeapExtractMin, kdy jsme do kořene haldy

přesunuli jiný prvek, který jsme potom museli prohazováním zabublávat dolů. Stejný zabublávací postup nyní tedy spustíme na každý vrchol předposlední hladiny T, což ve výsledku vyrobí korektní dvouhladinové „haldičky“ v předposledním hladině.

Naprosto stejný postup můžeme nyní aplikovat i pro třetí hladinu stromu T, pak na čtvrtou nejnižší, a tak dále, až ve výsledku na kořen stromu T, což ve výsledku ze stromu T postaví korektní haldu. (Matematicky založený čtenář si může korektnost algoritmu dokázat formálně matematickou indukcí dle výšky stromu T.) Této operaci stavění haldy budeme říkat HeapBuild.

Zanalyzujeme nyní časovou složitost operace HeapBuild. K tomu si uvědomíme, že čas potřebný na korekci haldové podmínky porušené v kořeni haldy výšky h je $\Theta(h)$. Dále, počet prvků poslední hladiny stromu T odhadneme shora N, počet prvků předposlední hladiny je nejvýše $N/2$, předpředposlední $N/4$, a tak dále, obecně k-tá hladina odspodu obsahuje jistě nejvýše $N/2^{k-1}$ vrcholů. Protože stavbu haldy dle předchozího odstavce stačí zahájit od předposlední hladiny, dostáváme řádový horní odhad na počet operací

$$\frac{N}{2}\cdot 1+\frac{N}{4}\cdot 2+\frac{N}{8}\cdot 3+\ldots+\frac{N}{N}\cdot\log_2 N\leq N\sum_{i=1}^{\lceil\log_2 N\rceil}\frac{i}{2^i}\leq N\sum_{i=1}^{\infty}\frac{i}{2^i}.$$

Pro analýzu poslední sumy nyní můžeme využít poznatků z matematické analýzy a použitím například podílového kritéria pro konvergenci řad ukázat, že suma $\sum_{i=1}^{\infty}\frac{i}{2^i}$ konverguje, což tedy dává odhad $\mathcal{O}(N)$ na počet operací. (Čtenáře odkážeme na cvičení 1.5.7, aby součet řady spočetl přesně.) Výsledná analýza časové složitosti $\Theta(N)$ plyne z toho, že vstupních N prvků musíme alespoň přečíst.

Algoritmus HeapBuild
Vstup: Prvky $p_1,\ldots,p_N$.
Výstup: Halda H.

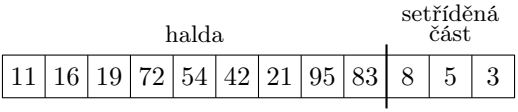
1. Postav N-vrcholový strom T dle axiomů haldy 1 a 2.
2. Do vrcholů T libovolně zapiš prvky $p_1,\ldots,p_N$.
3. Pro hladinu h postupně od druhé nejnižší až ke kořenové:
4. Pro každý vrchol v hladiny h:
5. Dokud má v syny opakuj:
6. $s\leftarrow$ menší z obou synů *(Při pouze jednom synovi je to snazší.)*
7. Pokud $h(s)\geq h(v)$, skončí vnitřní cyklus.
8. Prohoď obsah v a s.
9. $v\leftarrow s$
10. Vrať strom T jako korektní haldu H.

Poznamenejme, že zejména u operace HeapBuild vynikne jednoduchost implementace haldy pomocí pole (což čtenáři doporučujeme vyzkoušet ve cvičení 1.5.1), protože není třeba složitě rozlišovat, do které hladiny určitý vrchol T patří.

1.3. Třídíme haldou

Třídění haldou (HeapSort) patří mezi klasická vylepšení přímých třídících metod, konkrétně Selectsortu popsaného v kapitole o třídění. Selectsort postupně budoval setříděné pole tak, že v každém kroku vybral z dosud nepoužitých prvků minimum a připojil ho na konec setříděné posloupnosti. A právě na tento výběr by se nám mohla velice hodit halda.

Je poněkud nešikovné, když třídicí algoritmus potřebuje větší množství pomocné paměti, a proto se pokusíme haldu postavit a udržovat přímo v tříděném poli. V každém kroku pak odebereme z haldy minimum. Tím se nám halda přirozeně zmenší a na svém konci uvolní jednu pozici v poli, kam můžeme odebrané minimum umístit. Na konci, když se halda úplně vyprázdní, nám zůstanou jen sestupně setříděné prvky.



Obr. 1.2: Halda v poli při třídění

Algoritmus HeapSort
Vstup: Pole $P[1 \dots N]$.
Výstup: Setříděné pole P.

1. V poli P postavíme maximovou haldu.
2. Pro i jdoucí od N do 2 opakujeme:
3. $x \leftarrow P[1]$ *(Dočasně podržíme maximum.)*
4. *ExtractMax*$(P[1 \dots i])$ *(Smažeme maximum z i-prvkové haldy.)*
5. $P[i] \leftarrow x$

Pokud bychom chtěli prvky třídit vzestupně, máme několik možností. Samozřejmě bychom mohli reprezentaci haldy v poli otočit, takže kořen by byl na pozici N (místo 1), ale existuje i jednodušší způsob – místo minimové haldy pracovat s haldou maximovou. Použitím maximové haldy zajistíme, že se prvky budou ukládat od největších a výsledné pole bude tedy setříděné sestupně.

Nyní se podívejme na časovou složitost. Sestavení haldy nám zabere $\Theta(N)$, ale to provádíme jen jednou na začátku. Algoritmus má $N-1$ kroků a v každém z nich provádíme operaci odebrání ExtractMax, které trvá čas $\Theta(\log N)$, takže celková složitost bude $\Theta(N + N \log N) = \Theta(N \log N)$.

1.4. Dolní odhad složitosti HeapExtractMin

Dosud jsme nepoložili otázku, zda náhodou výše popsané operace nejdou ještě nějak podstatně urychlit. Ukážeme, že pokud opravdu trváme na zachování axiomů

haldy po skončení každé operace, námi uvedený HeapExtractMin je nejrychlejší možný.

Tvrzení: Nechť H je N-prvková binární halda. Potom každá korektní operace odebrání minima z haldy musí mít časovou složitost $\Omega(\log N)$.

Důkaz: Předpokládejme pro účely sporu, že existuje nějaký asymptoticky rychlejší algoritmus M na odstranění minima, který má časovou složitost $\mathcal{O}(f(N))$, kde $f(N) \ll \log N$.

Zkonstruujeme nyní následující třídící algoritmus na třídění libovolných prvků $p_1, \ldots, p_N$: pomocí operace HeapBuild sestavíme ze vstupních prvků korektní minimovou haldu a poté z ní algoritmem M postupně odebíráme minima, která vypisujeme na výstup. Tento algoritmus zjevně korektně třídí. Jeho složitost se skládá z $\mathcal{O}(N)$ za operaci HeapBuild a $\mathcal{O}(N \cdot f(N)$ za N volání algoritmu M.

To však znamená, že jsme právě sestrojili třídící algoritmus s lepší časovou složitostí než $\mathcal{O}(N \log N)$, neboť $N f(N) \ll N \log N$. To je spor s dolním odhadem složitosti třídění, takový třídící algoritmus nemůže existovat a tedy nemůže existovat ani takový algoritmus M na odebírání minima z haldy. □

1.5. Obecné d-regulární haldy

Halda, kterou jsme představili, není pochopitelně jedinou strukturou, která umí rychle mazat minimum. Kromě komplikovanějších struktur jako jsou Binomiální a Fibonacciho haldy, se kterými se seznámíme v následujících kapitolách, můžeme lehce rozšířit i definici binární haldy na něco obecnějšího.

Definice: *d-regulární halda* je datová struktura uchovávající množinu N porovnatelných prvků ve vrcholech d-árního ($d > 1$ přirozené) stromu T, který:

1. splňuje, že každý vrchol $v \in T$ má maximálně d následníku,
2. má všechny hladiny plně obsazené, s výjimkou poslední,
3. v poslední hladině jsou všechny listy nalevo,
4. pro každý vrchol $v \in T$ a jeho syny $s_1, \ldots, s_d$ platí $h(v) \leq h(s_i)$ pro $i = 1, \ldots, d$.

Binární strom je tedy vlastně speciálním případem (2-ární strom) a halda, o které jsme mluvili doposud, se celým jménem nazývá 2-regulární halda.

Všechny vlastnosti obecných d-regulárních hald jsou analogické vlastnostem klasické binární haldy. Výrazně se liší pouze svou výškou, protože úplný d-ární strom má $\lfloor \log_d N \rfloor + 1$ hladin.

Časová složitost vkládání závisí pouze na výšce této haldy a je tedy rovna $\Theta(\log_d N)$. Operace vypouštění minima zároveň potřebuje v každém kroku vybrat nejmenšího syna. Složitost bude tedy $\Theta(d \log_d N)$. Parametr d nám poskytuje určitou možnost ladění a zvýhodnění složitosti vkládání oproti odebírání minima, pokud to aplikace žádá. V extrémním případě, pro $d = \mathcal{O}(N)$, bude vkládání probíhat v konstantním čase, ale odebrání minima nám zabere $\mathcal{O}(N)$.

Toto rozšíření definice zde uvádíme pouze pro úplnost. Budeme-li dále mluvit pouze o haldě, budeme mít na mysli klasickou 2-regulární haldu, pokud nebude výslovně uvedeno jinak.

Cvičení:

1. Napište všechny výše uvedené algoritmy pro haldu reprezentovanou v poli.
2. Navrhněte operaci HeapChangeKey(H, x, Δ), kde H je halda, x ukazatel na prvek v haldě a Δ změna hodnoty klíče prvku x, která změní hodnotu klíče x od Δ a pochopitelně zanechá haldu H v korektním stavu. Jakou bude mít časovou složitost?
3. Navrhněte operaci HeapDelete(H, x), kde H je halda a x ukazatel na prvek v haldě, která z haldy smaže prvek x. Jakou bude mít časovou složitost?
4. Prvky klasické haldy umíme uspořádat do pole tak, že první prvek bude kořen a prvek na pozici i bude mít syny na pozicích $2i$ a $2i+1$. Jak byste vhodně poskládali ternární (3-regulární) a obecnou d-regulární haldu do pole indexovaného od 1?
5. Rozmyslete, jaké vlastnosti by měla 1-regulární halda.
6. Navrhněte, jak d-regulární haldu reprezentovat v poli.
7. Dokažte, že $\sum_{i=1}^{\infty} i/2^i = 2$.