Halda

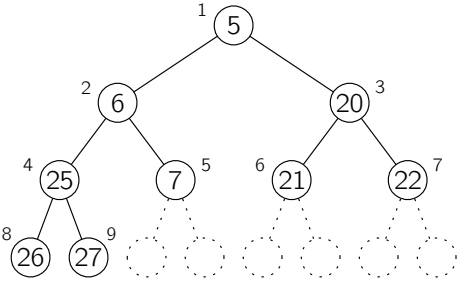
Halda je datová struktura pro uchovávání množiny čísel (či jakýchkoliv jiných prvků, na kterých máme definováno uspořádání, tj. umíme pro každou dvojici prvků říci, který z nich je menší). Tato datová struktura obvykle podporuje následující operace: přidání nového prvku, nalezení nejmenšího prvku a odebrání nejmenšího prvku. My si ukážeme jednoduchou implementaci haldy, která bude při uložení N prvků potřebovat čas $\mathcal{O}(\log N)$ na přidání či odebrání jednoho prvku a $\mathcal{O}(1)$ (tj. konstantní) na zjištění hodnoty nejmenšího prvku.

Naše implementace bude vypadat následovně: Pokud halda obsahuje N prvků, uložíme její prvky do pole na pozice 1 až N. Prvek na pozici k bude mít dva *následníky,* a to prvky na pozicích $2k$ a $2k + 1$; samozřejmě, pokud je k velké, a tedy např. $2k + 1 > N$, má takový prvek jen jednoho či dokonce žádného následníka. Naopak prvek na pozici $\lfloor k/2 \rfloor$ nazveme *předchůdcem* prvku na pozici k. Ti z vás, kteří znají binární stromy, v tomto jistě rozpoznali způsob, jak v poli uchovávat úplné binární stromy (následníci jsou synové a předchůdci otcové v obvyklé stromové terminologii, prvek č. 1 je kořen stromu). O stromech se více dozvíte v následujících kuchařkách.

Prvky haldy však v poli neuchováváme v úplně libovolném pořadí. Chceme, aby platilo, že každý prvek je menší nebo roven všem svým následníkům. Naše halda (uložená v poli `h`) tedy může vypadat např. takto:

`i`	1	2	3	4	5	6	7	8	9
`h[i]`	5	6	20	25	7	21	22	26	27

Tomu odpovídá tento strom:



Z toho, co jsme si právě popsali, je jasné, že nejmenší prvek je uložen na pozici s indexem 1, a tedy můžeme snadno v konstantním čase zjistit jeho hodnotu. Ještě prozradíme, jak lze prvky do haldy rychle přidávat a odebírat:

Jestliže halda obsahuje N prvků, pak nový prvek, říkejme mu třeba x, přidáme na konec pole, tj. na pozici s indexem $N + 1$. Nyní x porovnáme s jeho předchůdcem. Pokud je jeho předchůdce menší, je vše v pořádku a jsme hotovi. V opačném případě x s jeho předchůdcem prohodíme.

Tím jsme problém napravili, ale nyní může být x menší než jeho nový předchůdce. To lze napravit dalším prohozením a tak budeme pokračovat dále, než se buďto dostaneme do situace, kdy už je x větší nebo rovno svému předchůdci, nebo „vybublá“

až do kořene haldy, kde už žádného předchůdce nemá. Protože se v každém kroku pozice, na níž se prvek x právě nachází, zmenší alespoň na polovinu, provedeme dohromady nejvýše $\mathcal{O}(\log N)$ výměn, a tedy spotřebujeme čas $\mathcal{O}(\log N)$.

Odebírání nejmenšího prvku probíhá podobně: Prvek z poslední pozice (tj. z pozice N) přesuneme na pozici 1, tedy místo minima. Místo s předchůdci jej však porovnáme s jeho následníky a v případě, že je větší než některý z jeho následníků, opět je prohodíme (pokud je větší než oba následníci, prohodíme ho s menším z nich). A protože se nám v každém kroku index „bublajícího“ prvku v poli alespoň zdvojnásobí, opět spotřebujeme čas $\mathcal{O}(\log N)$.

Poznámky

- V čase $\mathcal{O}(\log N)$ lze z haldy smazat dokonce libovolný prvek, pokud si ovšem pamatujeme, kde se v haldě nachází. Také můžeme prvek ponechat a jen změnit jeho hodnotu.

Zdrojový kód

```
var halda: array[1..MAX] of integer;
    N: integer;            { počet prvků v haldě }

function nejmensi: integer;
begin
  nejmensi:=halda[1]
end;

procedure vloz(prvek: integer);
var i, x: integer;
begin
  i:=N; N:=N+1;
  halda[i]:=prvek;
  while (i>1) and (halda[i div 2]>halda[i]) do begin
    x:=halda[i div 2];
    halda[i div 2]:=halda[i];
    halda[i]:=x;
    i:=i div 2
  end
end;

procedure smaz_nejmensi;
var i, j, x: integer;
begin
  halda[1]:=halda[N];
  N:=N-1;
  i:=1;
  while 2*i<=N do begin
    j:=i;
    if halda[j]>halda[2*i] then j:=2*i;
    if (2*i+1<=N) and (halda[j]>halda[2*i+1]) then j:=2*i+1;
```

```
    if i=j then break;
    x:=halda[i]; halda[i]:=halda[j]; halda[j]:=x;
    i:=j
  end
end;
```

HeapSort

Když už máme k dispozici haldu, můžeme pomocí ní například snadno třídit čísla. Máme-li N čísel, která chceme setřídit, vytvoříme si z nich nejprve haldu o N prvcích (například postupným vkládáním do prázdné haldy), načež z ní budeme postupně N-krát odebírat nejmenší prvek. Tím získáme prvky původního pole v rostoucím pořadí. Celkově provedeme N vložení, N nalezení minima a N smazání. To vše dohromady stihneme v čase $\mathcal{O}(N \log N)$.

Než si ukážeme program, přidáme ještě dva triky, které nám implementaci značně usnadní. Předně si vše uložíme do jednoho pole – to bude při plnění haldy obsahovat na svém začátku haldu a na konci zbytek vstupního pole, přitom zbytek pole se bude postupně zmenšovat a uvolňovat tak místo haldě; naopak v druhé polovině algoritmu budeme zmenšovat haldu a do volného prostoru ukládat setříděné prvky. K tomu se nám bude hodit získávat prvky v opačném pořadí, proto si upravíme haldu tak, aby udržovala nikoliv minimum, nýbrž maximum.

Druhý trik spočívá v tom, že nebudeme haldu vytvářet postupným vkládáním, nýbrž naopak zabubláváním prvků (podobným, jako děláme při mazání minima) od konce. Všimněte si, že takto také získáme správné nerovnosti mezi prvky a jejich následníky, a dokonce tak zvládneme celou haldu vytvořit v lineárním čase – proč to tak je, si zkuste dokázat sami (stačí si uvědomit, kolikrát zabubláváme které prvky). Zbytek třídění bohužel nadále zůstává $\mathcal{O}(N \log N)$.

Tomuto algoritmu se obvykle říká *HeapSort* (čili třídění haldou) a je jedním z mála známých rychlých třídicích algoritmů, které nepotřebují pomocnou paměť.

```
type Pole = array[1..MAXN] of Integer;

procedure HeapSort(var A: Pole);
var i, x: integer;
   procedure bublej(m, i: integer); { "zabublání" prvku }
   { m je velikost haldy, i je index zabublávaného prvku }
   var j, x: integer;
   begin
      while 2*i<=m do begin
         j:=2*i;
         if (j<m) and (A[j+1]>A[j]) then j:=j+1;
         if A[i]>=A[j] then break;
         x:=A[i]; A[i]:=A[j]; A[j]:=x;
         i:=j;
      end;
   end;
```

```
begin
   for i:=N div 2 downto 1 do bublej(N,i); { bublej }
   for i:=N downto 2 do begin { vybírej maximum }
      x:=A[1]; A[1]:=A[i]; A[i]:=x;
      bublej(i-1, 1);
   end;
end;
```

Dan Kráľ, Martin Mareš a Petr Škoda

Úloha 19-2-3: Moneymaker

Na vstupu dostane váš program číslo N, což je počet úkolů ke zpracování. Zpracování každé úlohy zabere jednotkový čas. Dále pak N řádků, každý se dvěma čísly. První číslo znamená, do kdy je třeba úkol vykonat, a druhé číslo je odměna, kterou za splněný úkol dostaneme. V jednom čase mohu pracovat právě na jednom úkolu. Výstupem programu by pak mělo být takové pořadí úkolů, aby zisk byl maximální. Pokud je takových pořadí více, stačí libovolné z nich.

Příklad: Pro $N = 4$ a záznamy

3	1
1	3
2	5
2	4

je optimální pořadí 3, 4 a 1.

Úloha 20-4-4: Skupinky pro chytré

Budeme pracovat se záznamy lidí. Každý člověk má jméno a IQ (přičemž IQ je celé kladné číslo). Z lidí budeme vytvářet skupinky. Každá skupinka má unikátní *ID* (identifikační číslo), které je opět celé a kladné. Na počátku máme pouze jedinou prázdnou skupinku s *ID*=1.

Nad skupinkami chceme provozovat operace:

- `INSERT` – Vloží nového člověka.
- `FIND_BEST` – Nalezne člověka s nejvyšším IQ.
- `DELETE_BEST` – Člověk s nejvyšším IQ odchází za kariérou do zahraničí (odstraníme jej).

Výše uvedené operace dostanou vždy *ID* skupinky, nad kterou mají být provedeny. Žádná z operací nemodifikuje skupinku, ale místo toho vytvoří skupinku novou, ve které budou uloženy výsledky operace. *ID* nové skupinky bude nejmenší dosud nepoužité číslo. Pochopitelně operace `FIND_BEST` pouze vrací nalezeného člověka, takže nevytvoří novou skupinku. Skupinky nikdy nezanikají, takže je potřeba si je nějakým způsobem udržovat všechny.

Výsledek každé operace musíte oznámit ještě před tím, než začnete zpracovávat operaci další – tj. nesmíte si operace bufferovat a pak jich provést víc najednou.

Vaším úkolem je navrhnout a popsat vhodnou datovou reprezentaci a jak na ní budou probíhat požadované operace.

Příklad: Jak bylo řečeno, na začátku máme jen jednu prázdnou třídu s *ID*=1. Budeme provádět operace:

- `INSERT("Aleš", IQ=130)` do *ID*=1 vytvoří skupinku *ID*=2.
- `INSERT("Petr", IQ=110)` do *ID*=2 vytvoří skupinku *ID*=3.
- `INSERT("Jana", IQ=140)` do *ID*=1 vytvoří skupinku *ID*=4.
- `FIND_BEST` v *ID*=2 vrátí `"Aleš"`.
- `FIND_BEST` v *ID*=3 vrátí také `"Aleš"`.
- `FIND_BEST` v *ID*=4 vrátí ovšem `"Jana"`.
- `DELETE_BEST` z *ID*=3 vytvoří skupinku *ID*=5.
- `FIND_BEST` v *ID*=5 vrátí `"Petr"`, protože Aleše odstranila předchozí operace.