

Příklady otázek – Algoritmy a datové struktury

Reálné otázky ze SZZ 2022–2025 (I2)

Jakub Klesa

Pokryté předměty: NTIN060 (ADS 1), NTIN061 (ADS 2)

Obsah

1	Binární vyhledávací stromy	1
2	Třídění	2
3	Třídění sléváním	3
4	Algoritmy rozděl a panuj	4
5	Třídy složitosti	5

Poznámka. Tento dokument obsahuje reálné otázky z minulých termínů SZZ (2022–2025) zařazené do okruhu I2. U otázek s obrázkem je grafika oříznutá z originálního PDF (viz odkaz na zdroj). Text, číselná data a tabulky jsou přepsané věrně. Zadání i oficiální náčrty řešení pocházejí ze sad zveřejněných na webu MFF UK:

<https://www.mff.cuni.cz/cs/studenti/bakalarske-studium/statni-zaverecne-zkousky/bakalarske-statni-zkousky-studijniho-programu-informatika>

Náčrty označené jako doplněné jsou vlastní, oficiální sada je k dané otázce neobsahuje. Pojmy a značení odpovídají učebnici M. Mareše a T. Vally *Průvodce labyrintem algoritmů* (<https://pruvodce.ucw.cz/>).

1 Binární vyhledávací stromy

(Reálná otázka: léto 2022, č. 1 – Binární vyhledávací stromy)

Zadání.

1. Definujte binární vyhledávací strom (BVS).
2. Implementujte vložení prvku do BVS (bez vyvažování) a rozeberte časovou složitost v nejlepším a nejhorším případě.
3. Implementujte nalezení následníka, tj. pro daný klíč x vrchol s nejmenším klíčem ostře větším než x , a rozeberte časovou složitost v nejlepším a nejhorším případě.

Implementaci popište pseudokódem.

Náčrt řešení.

(1) *Binární vyhledávací strom* je binární strom, v němž každý vrchol nese klíč a pro každý vrchol platí *uspořádací vlastnost*: všechny klíče v levém podstromu jsou menší než klíč vrcholu a všechny klíče v pravém podstromu jsou větší než klíč vrcholu.

(2) Vložení. Sestupujeme od kořene a podle porovnání klíče jdeme doleva, nebo doprava; nový prvek vložíme jako list na místo, kde sestup narazí na prázdný podstrom.

```
Insert(strom T, klic k):
    if T.koren == null:
        T.koren = new Uzel(k); return
    v = T.koren
    loop:
        if k == v.klic: return // klic uz ve strome je
        if k < v.klic:
            if v.levy == null: v.levy = new Uzel(k); return
            else: v = v.levy
        else: // k > v.klic
            if v.pravy == null: v.pravy = new Uzel(k); return
            else: v = v.pravy
```

Vložení projde jednou cestou od kořene nejvýše k listu, stojí tedy $O(h)$, kde h je hloubka stromu. Nejhorší případ: degenerovaný strom (liána, např. vzniklý vkládáním setříděných klíčů) má $h = n - 1$ a nový klíč se připojí pod jeho dolní konec, tj. $\Theta(n)$. Nejlepší případ: sestup skončí hned pod kořenem, tj. $\Theta(1)$ (např. do liány $1, 2, \dots, n$ s kořenem 1 vkládáme 0). Uvažujeme-li nejlepší tvar stromu při nejhorší volbě klíče, je to vyvážený strom s $h = \Theta(\log n)$, tedy $\Theta(\log n)$.

(3) Následník. Pokud má vrchol pravý podstrom, následník je jeho nejlevější vrchol. Jinak stoupáme nahoru, dokud nejsme levým synem; tento rodič je následník (neexistuje-li, x byl maximum a následník není).

```
Successor(uzel x):
    if x.pravy != null: // nejmensi v pravem podstromu
        v = x.pravy
        while v.levy != null: v = v.levy
        return v
    v = x; r = x.rodic // vystup nahoru, dokud nejsem levy syn
    while r != null and v == r.pravy:
        v = r; r = r.rodic
    return r
```

Obě větve projdou jednu cestu stromem, takže složitost je $O(h)$. Nejhorší případ $\Theta(n)$: ve stromu vzniklém vkládáním $1, n, n - 1, \dots, 2$ je následník kořene 1 klíč 2 na dně levé větve pod n , cesta má $n - 1$ hran. Nejlepší případ $\Theta(1)$: pravý syn x je list, nebo x je levý syn bez pravého podstromu. Ve vyváženém stromu je i nejhorší případ jen $\Theta(\log n)$.

(V PDF bez náčrtu řešení; náčrt doplněn k výkladu okruhu.)

2 Třídění

(Reálná otázka: jaro 2023, č. 1 – Třídění (BST/AVL, složitost))

Zadání.

1. Popište, jak pomocí binárního vyhledávacího stromu setřídít (uspořádat vzestupně) posloupnost $x_1, \dots, x_n$ navzájem různých prvků.
2. Co se změní, pokud se prvky mohou opakovat?
3. Co z toho plyne pro minimální možnou složitost operací s vyhledávacím stromem? Předpokládejme přitom, že klíče ve stromu je možné pouze porovnávat.
4. Jakou bude mít váš algoritmus z bodu 1 časovou složitost, pokud jako strom použijeme AVL strom a prvky posloupnosti budou řetězce délky L ?

Náčrt řešení.

1) Prvky $x_1, \dots, x_n$ postupně vložíme do zprvu prázdného binárního vyhledávacího stromu a pak strom projdeme *in-order* (levý podstrom, vrchol, pravý podstrom). Z uspořádací vlastnosti plyne, že *in-order* průchod vypíše klíče vzestupně. Vložení stojí $O(h)$, průchod $\Theta(n)$. S nevyváženým stromem může vzniknout liána (např. pro už seřazený vstup) a čas je až $\Theta(n^2)$; s vyváženým stromem (AVL) je $h = \Theta(\log n)$ a celé třídění trvá $\Theta(n \log n)$.

2) Uspořádací vlastnost s ostrými nerovnostmi ($<$ vlevo, $>$ vpravo) stejné klíče nepřipouští. Možnosti: ve vrcholu držet počet výskytů klíče a při průchodu ho vypsát tolikrát; nebo povolit rovnost na jedné straně ($\leq$); nebo klíče zjednotřit dvojicí (x_i, i) porovnávanou lexikograficky (třídění je pak navíc stabilní). Složitost se nemění.

3) Třídění stromem je n operací INSERT a jeden *in-order* průchod za $O(n)$, který nic neporovnává. Porovnávací třídění ale potřebuje v nejhorším případě $\Omega(n \log n)$ porovnání (rozhodovací strom má aspoň $n!$ listů, jeho hloubka je aspoň $\log_3 n! = \Omega(n \log n)$). Kdyby šlo n vložení provést v čase $o(n \log n)$, porušili bychom tento dolní odhad. Operace INSERT proto v porovnávacím modelu nemůže mít (ani amortizovaně) složitost $o(\log n)$; logaritmická složitost operací AVL stromu je asymptoticky optimální.

4) Porovnání dvou řetězců délky L stojí $O(L)$ (znak po znaku). Vložení do AVL stromu udělá $O(\log n)$ porovnání, tedy $O(L \log n)$, a n vložení $O(nL \log n)$. *In-order* průchod vypíše n řetězců celkové délky nL za $O(nL)$. Celkem $O(nL \log n)$.

(V PDF bez náčrtu řešení; náčrt doplněn k výkladu okruhu.)

3 Třídění sléváním

(Reálná otázka: léto 2023, č. 1 – Třídění sléváním (Mergesort, slévání k posloupností))

Zadání.

1. Formulujte algoritmus třídění posloupnosti n celých čísel sléváním (Mergesort) a запиšte ho v pseudokódu.
2. Analyzujte časovou a prostorovou složitost tohoto algoritmu.
3. Navrhněte efektivní algoritmus pro slévání k seřazených posloupností o celkem n prvcích. Upravte Mergesort, aby toto slévání používal. Jak se změní jeho časová složitost? Zapište ji jako funkci proměnných n a k .

Náčrt řešení.

1) Mergesort (třídění sléváním):

```
MergeSort(a[0..n-1]):
  if n <= 1: return a
  m = n div 2
  L = MergeSort(a[0..m-1]); R = MergeSort(a[m..n-1])
  return Merge(L, R)

Merge(L, R): // obe posloupnosti setridene
  i = 0; j = 0; out = []
  while i < |L| and j < |R|:
    if L[i] <= R[j]: out.append(L[i]); i++
    else: out.append(R[j]); j++
  pripoj zbytek L[i..] a R[j..] na konec out
  return out
```

2) MERGE přesune každý prvek jednou, stojí $\Theta(n)$. Rekurence $T(n) = 2T(n/2) + \Theta(n)$ má podle kuchařkové věty ($a = 2$, $b = 2$, $c = 1$, $a/b^c = 1$) řešení $T(n) = \Theta(n \log n)$, a to v každém

případě (nezávisí na datech). Paměť: pomocná pole pro slévání $\Theta(n)$ a zásobník rekurze $\Theta(\log n)$, celkem $\Theta(n)$.

3) Slévání k posloupností: do minimové haldy vložíme první prvek každé posloupnosti (s informací, odkud pochází). Opakovaně vyjme minimum (EXTRACTMIN), zapíšeme ho na výstup a vložíme do haldy další prvek téže posloupnosti, pokud existuje. Halda má nejvýše k prvků, každá operace stojí $O(\log k)$, každý z n prvků se jednou vloží a jednou vyjme: celkem $O(n \log k)$ (plus $O(k)$ na postavení haldy).

k-cestný Mergesort: vstup rozdělíme na k částí velikosti n/k , každou setřídíme rekurzivně a výsledky slijeme haldou. Rekurence $T(n) = kT(n/k) + \Theta(n \log k)$. Strom rekurze má $\log_k n$ hladin a na každé je slévání celkem $\Theta(n \log k)$, takže

$$T(n) = \Theta(n \log k \cdot \log_k n) = \Theta\left(n \log k \cdot \frac{\log n}{\log k}\right) = \Theta(n \log n).$$

Asymptotická složitost se tedy nezmění, faktor $\log k$ se vykrátí (pro $k \geq 2$; pro $k = n$ jde vlastně o třídění haldou, opět $\Theta(n \log n)$).

(V PDF bez náčrtu řešení; náčrt doplněn k výkladu okruhu.)

4 Algoritmy rozděl a panuj

(Reálná otázka: léto 2024, č. 1 – Algoritmy rozděl a panuj (Master theorem, medián $2n$ čísel))

Zadání.

1. Mějme algoritmus A , který na datech velikosti n udělá $T(n)$ elementárních operací. Algoritmus A je rekurzivní a pracuje takto (a, c, x, y jsou přirozená čísla, $a > 0$, $c > 1$):
 - (a) Udělá $\Theta(n^x)$ elementárních operací, aby ze vstupních dat vybral a podmnožin velikosti n/c .
 - (b) Rekurzivně pustí sám sebe na každou z vybraných podmnožin dat (pokud má velikost alespoň c , pro data menší velikosti vyřeší úlohu v konstantním čase).
 - (c) Udělá $\Theta(n^y)$ elementárních operací, aby všechna řešení z bodu b) spojil do řešení původní úlohy na datech velikosti n .

Určete (bez důkazu) asymptoticky těsný odhad funkce $T(n)$ v závislosti na parametrech a, c, x, y .

2. Necht X a Y jsou pole délky n , každé obsahující setříděnou posloupnost n přirozených čísel. Navrhněte a popište algoritmus s časovou složitostí $O(\log n)$, který najde medián (jeden z mediánů) všech $2n$ čísel obsažených v polích X a Y . Dokažte metodou z bodu 1), že složitost Vašeho algoritmu je opravdu $O(\log n)$.

Náčrt řešení.

1) Úloha je řešitelná pomocí Master Theoremu, tj. Kuchařky. V algoritmu A je společná rezie $\Theta(n^x + n^y) = \Theta(n^{\max\{x,y\}})$ při každém rekurzivním volání. Z toho vychází rekurzivní rovnice $T(n) = a \cdot T(n/c) + \Theta(n^{\max\{x,y\}})$ pro časovou složitost řešitelná pomocí Master Theoremu. Zavedeme $d = \max\{x, y\}$. Řešení rovnice se dělí na 3 případy:

1. $T(n) = \Theta(n^d)$ pro $a/c^d < 1$;
2. $T(n) = \Theta(n^d \log n)$ pro $a/c^d = 1$;
3. $T(n) = \Theta(n^{\log_c a})$ pro $a/c^d > 1$.

2) V jednotlivých krocích algoritmu porovnáváme mediány obou polí X a Y , například prvky $x = X[\lfloor (n-1)/2 \rfloor]$ a $y = Y[\lfloor (n-1)/2 \rfloor]$ při indexování od 0.

Pokud $x < y$, můžeme odstranit levou polovinu X (prvky vlevo od x) a pravou polovinu Y (prvky vpravo od y) a na ponechané prvky voláme rekurzi. Je velmi podstatné, aby počet odstraněných prvků z X (které jsou všechny garantovaně menší než hledaný medián) byl stejný jako počet odstraněných prvků z Y (které jsou všechny garantovaně větší než hledaný medián). Tato vlastnost zajistí, že medián prvků které vstupují do rekurzivního volání je také mediánem původní množiny prvků.

Pokud $x > y$, pak odstraňujeme symetricky levou polovinu Y a pravou polovinu X a voláme rekurzi.

Pokud $x = y$, pak je x medián (jeden z mediánů).

Přístup k prvkům x a y , jejich porovnání a posunutí indexů (vymezujičích aktuální části X a Y) zvládneme v konstantním čase $\Theta(1) = \Theta(n^0)$. Pro $n < 3$ dopočítáme medián v konstantním čase.

Vychází rovnice $T(n) = 1 \cdot T(n/2) + \Theta(1)$, tj. $a = 1$, $c = 2$, $d = 0$. V Kuchařce vyjde $a/c^d = 1/2^0 = 1$, tj. případ 2 s řešením $\Theta(n^0 \log n) = \Theta(\log n)$.

Doplňující vysvětlení (nad rámec oficiálního náčrtu): proč odříznutí nemění medián. Nechť $x < y$, $k = \lfloor (n-1)/2 \rfloor$ a všech $2n$ čísel setříděných je $z_1 \leq \dots \leq z_{2n}$ (mediány jsou z_n a z_{n+1}). Pro odebraný prvek $X[i]$, $i < k$, jsou čísla $\geq X[i]$ aspoň všechna v $X[i..n-1]$ (aspoň $n-k+1$ čísel) a v $Y[k..n-1]$ ($n-k$ čísel, všechna $\geq y > x$), celkem aspoň $2n-2k+1 \geq n+2$; proto $X[i] \leq z_{n-1}$. Symetricky pro odebraný $Y[j]$, $j \geq n-k$, jsou čísla $\leq Y[j]$ aspoň všechna v $X[0..k]$ a $Y[0..j]$, celkem aspoň $n+2$, takže $Y[j] \geq z_{n+2}$. Odebereme tedy k čísel pod dolním mediánem a k čísel nad horním mediánem a prostřední dvojice z_n, z_{n+1} se nezmění. Obě pole mají po kroku délku $n-k = \lceil (n+1)/2 \rceil$, pro $n \geq 3$ tedy velikost klesne; pro $n = 2$ by krok nic neodebral, proto se malé případy dořeší přímo. Algoritmus vrací dolní nebo horní medián (ověřeno náhodnými testy včetně opakovaných hodnot).

5 Třídy složitosti

(Reálná otázka: podzim 2025, č. 1 – Třídy složitosti (P/NP, redukce, NP-úplnost))

Zadání.

1. Definujte třídy složitosti **P** a **NP**.
2. Definujte převod (redukci) mezi rozhodovacími problémy (jazyky).
3. Definujte **NP-úplnost** rozhodovacího problému.
4. Rozhodněte, zda následující problém leží v třídě **NP**, a odpověď zdůvodněte: Je dáno přirozené číslo x zapsané v desítkové soustavě. Existují přirozená čísla a a b taková, že $1 < a, b < x$, $x = a \cdot b$ a desítkové zápisy a i b bez počátečních nul jsou stejně dlouhé?

Náčrt řešení.

Definice viz Průvodce labyrintem algoritmů, kapitoly 19.1 a 19.3. Problém leží v třídě **NP**, jako certifikát stačí použít číslo a . Verifikátor pak vypočte $b = x/a$ (pokud by při dělení vyšel zbytek, zamítne) a zkontroluje, že $1 < a, b < x$ a desítkové zápisy a a b jsou stejně dlouhé. Délka certifikátu je lineární a časová složitost verifikátoru polynomiální, obojí vzhledem k délce vstupu (počtu číslic x).