Základy návrhu a optimalizace algoritmů

Vyhledávání

Pavel Majer

22. 5. 2016

1. Úvod

Existuje mnoho různých datových struktur a způsobů vyhledávání dat. Algoritmy mají různě velkou asymptotickou složitost. Úkolem této práce je vyzkoušet v praxi chování několika základních algoritmů a porovnat výsledky s očekáváním – s teoretickou složitostí. Výstupy z této práce mohou pak pomoct s výběrem vhodné datové struktury a vyhledávacího algoritmu.

2. Popis podmínek testování

Specifikace počítače

Pro účely testování byl použit notebook Lenovo T430, s CPU i5-3320, 4GB RAM, Windows 7-64bit s SSD diskem.

Testovací program

Pro účely testování byl vyroben program, který podle nastavení vyrobí různě velká pole, a různě velké množiny náhodných čísel. Následně pak měří, jak dlouho trvá několikanásobné vkládání, hledání a mazaní dat v těchto polích.
Byly testovány dodané metody, určené pro vkládání, hledání a mazání dat v polích typu unsorted array, sorted array a Binary search tree.
Pro pole typu sorted array se provedou dvě sady testování – první s binárním vyhledáváním a druhá s interpolačním vyhledáváním.
Velký důraz byl také kladen na porovnání chování interpolačního vyhledávání a binárního vyhledávání. Pro účely tohoto zkoumání byla do testovacího programu dodána nová vlastnost – možnost výroby datových polí, kde budou čísla seřazena a budou bez mezer.

Nastavení parametrů pro testování

Délka pole (m)	10	50	100	500	1000	5000
Počet opakování (n)	50000	10000	5000	1000	100	10

Počet kol testování: 10
Náhodná čísla: ano (pro první běh ano, pro pozdější zkoumání chování interpolačního hledání: ne)

Záznam výsledků

Výsledky (průměrná doba a mediány) jsou programem zapsány od výstupních souborů a na obrazovku. Maximálnímu množství prvků v množinách odpovídají řádky ve výstupu, a naměřené hodnoty pro jednotlivé algoritmy/datové struktury jsou ve sloupcích.

Soubor times_avg.txt a times_med.txt

Nadpis sloupce	*Popis*
#m	*Maximální počet prvků ve skupině*
UA_insert	*Průměrné trvání vložení 1 prvku do unsorted array*
SA_insert	*Průměrné trvání vložení 1 prvku do sorted array*
BST_insert	*Průměrné trvání vložení 1 prvku do Binary search tree*
UA_search	*Průměrné trvání nalezení 1 prvku v unsorted array*
SA_binarys	*Průměrné trvání nalezení 1 prvku v sorted array pomoci binárního hledání*
SA_interps	*Průměrné trvání nalezení 1 prvku v sorted array pomoci interpolačního hledání*
UA_delete	*Průměrné trvání smazání 1 prvku z unsorted array*
SA_delete	*Průměrné trvání smazání 1 prvku z sorted array*
BST_delete	*Průměrné trvání smazání 1 prvku z Binary search tree*

Testování testovacího programu

Samotný program na testování algoritmů byl testován také. Byl zkoumán především z pohledu průkaznosti změřených časů. V jednom z testů bylo dočasně přidáno volání metody sleep a zkoumalo se, zda výsledné hodnoty z měření odpovídají očekávanému navýšení.

3. Očekávané chování

Důležité předpoklady

Dodané datové struktury se chovají jako množiny, proto při vkládání náhodných dat dochází k zahození takových čísel, která se opakují. Tato vlastnost má za následek mezery v datech a tyto mezery mohou ovlivnit chování interpolačního vyhledávání.

Očekávané chování

Popis očekávaných výsledků měření spolu s logickým odůvodněním těchto očekávání, především na základě teoretických poznatků o asymptotických složitostech jednotlivých algoritmů.

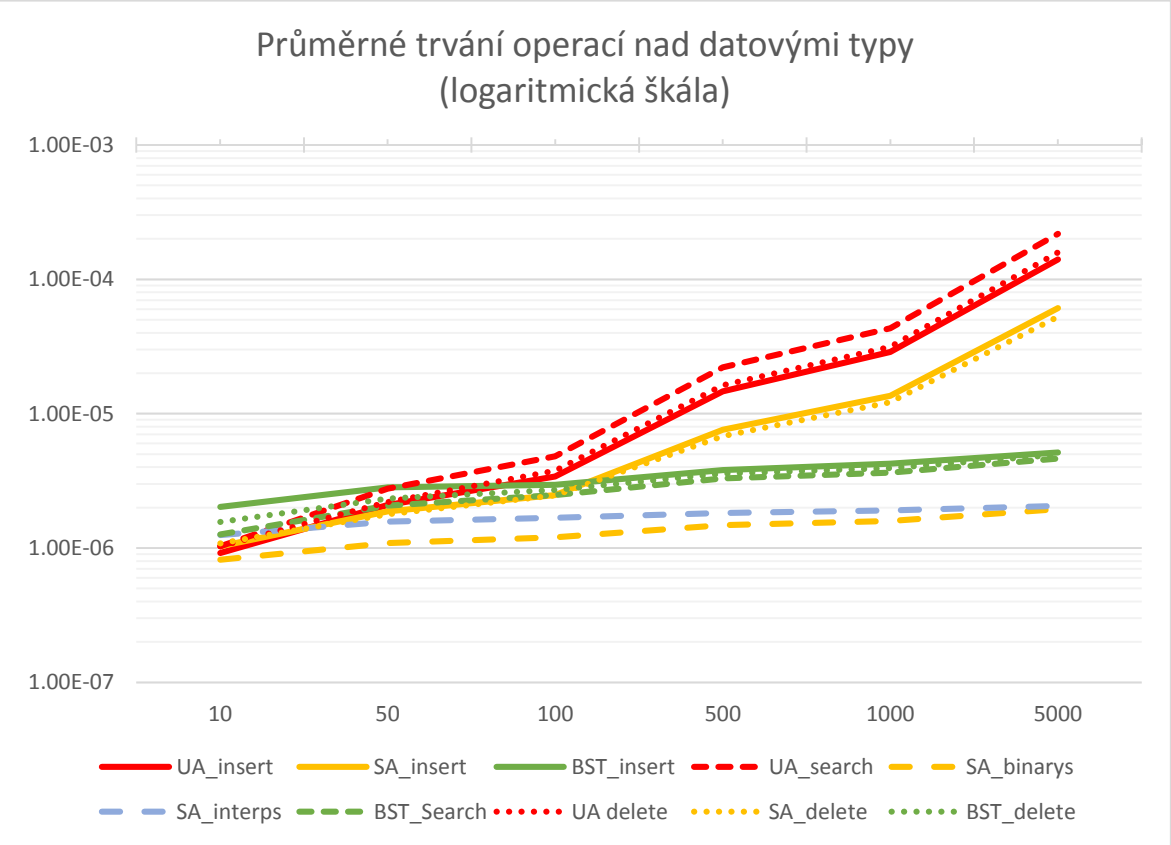
Algoritmus/pole	***Očekávaná složitost***	***Odůvodnění*** (1) (2) (3) (4)
vkládání do UA	O(n)	Vkládání do netříděného pole (typu množina) se skládá ze dvou kroků: zjištění, jestli prvek v poli již neexistuje [O (n)] a pak samotné vložení nakonec. Hledání v nejhorším i průměrném případě bude O (n), vkládání pak bude konstantní složitost O (1)
vkládání do SA	O(n)	Vkládání se také skládá ze dvou kroků – nalezení a případné vložení nového prvku na určité místo. Tento mechanismus bude lépe rychleji hledat, ale pomaleji vkládat.
vkládání do BST	O(log(n))	Počet kroků odpovídá výšce stromu. Při vyváženém rozložení stromu je složitost log (n), při velice nevyváženém rozložení stromu může být složitost až O(n)
Vyhledávání v UA	O(n)	Sekvenční vyhledávání porovnává každý prvek s hledanou hodnotou. Proto pak musí provést až n operací. Hledání by mělo být pomalejší než u SA
Binární vyhledávání v SA	O(log(n))	Pracuje na bázi dělení intervalů, na seřazených polích. Díky tomu není potřeba projít každý prvek v poli. Díky půlení intervalů se zkrátí množství kroků a složitost hledání.
Interpolační vyhledávání v SA	O(log(logn))	Jedná se o vylepšený mechanismus binárního vyhledávání dat. Odhaduje se index v poli podle velikosti hledaného čísla. Pokud jsou data seřazena a nejsou mezi hodnotami mezery, pak by tento mechanismus měl být rychlejší než binární
Vyhldávání v BST	O(log(n))	Počet kroků by měl být roven výšce stromu. Výška stromu je u rovnoměrně rozloženého stromu log(n). v Nejhorším případě rozložení dat bude strom nerovnoměrně rozložen a náročnost pak může být až O (n)
Mazání v UA	O(n)	Nejdříve se prvek musí najít, pak se smaže a všechny následující se přesunou o jedno místo doleva
Mazání v SA	O(n)	Mazání prvků z tříděného pole způsobí přesun následujících prvků, jak v nejhorším případě tak i v nejlepším.
Mazání v BST	O(log(n))	Počet kroků by měl být roven výšce stromu. Výška stromu je u rovnoměrně rozloženého stromu log(n). v Nejhorším případě rozložení dat bude strom nerovnoměrně rozložen a náročnost pak může být až O(n)

4. Výsledky měření

Po proběhnutí testů byly výsledky zaznamenány do následujících tabulek a grafů:

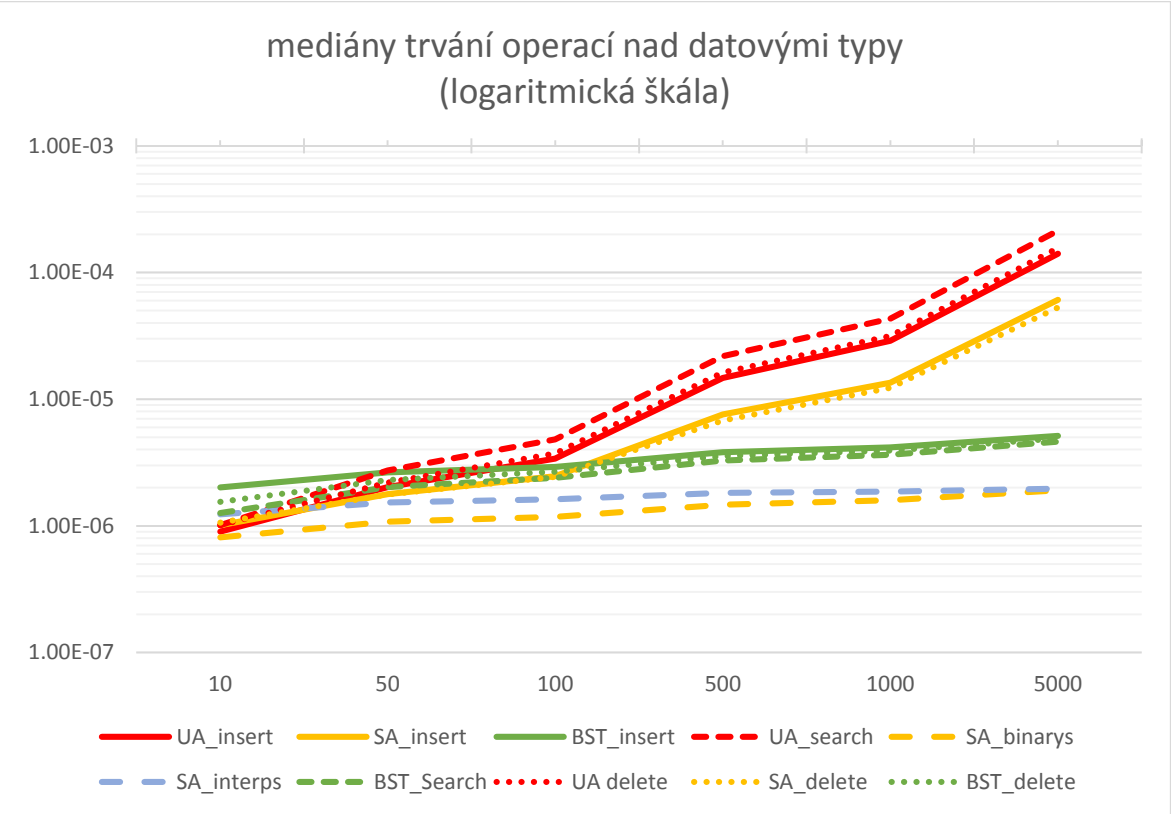
1. Průměrný čas potřebný na vložení, nalezení, smazání prvku (times_avg.txt)

#M	UA insert	SA in-sert	BST insert	UA search	SA binarys	SA interps	BST Search	UA delete	SA delete	BST delete
10	9.20E-07	1.03E-06	2.02E-06	1.03E-06	8.18E-07	1.25E-06	1.26E-06	1.03E-06	1.08E-06	1.56E-06
50	2.09E-06	1.87E-06	2.82E-06	2.77E-06	1.09E-06	1.57E-06	2.07E-06	2.20E-06	1.78E-06	2.33E-06
100	3.40E-06	2.48E-06	2.95E-06	4.82E-06	1.20E-06	1.68E-06	2.48E-06	3.77E-06	2.46E-06	2.70E-06
500	1.47E-05	7.58E-06	3.81E-06	2.21E-05	1.48E-06	1.82E-06	3.29E-06	1.63E-05	6.82E-06	3.57E-06
1000	2.88E-05	1.36E-05	4.25E-06	4.33E-05	1.59E-06	1.90E-06	3.64E-06	3.15E-05	1.22E-05	3.93E-06
5000	1.41E-04	6.10E-05	5.13E-06	2.18E-04	1.95E-06	2.06E-06	4.63E-06	1.59E-04	5.28E-05	4.94E-06



2. Medián času potřebného na vložení, nalezení, smazání prvku (times_med.txt)

#m	UA insert	SA insert	BST insert	UA search	SA binarys	SA interps	BST Search	UA delete	SA delete	BST delete
10	9.03E-07	1.01E-06	2.01E-06	1.01E-06	8.12E-07	1.23E-06	1.26E-06	1.01E-06	1.06E-06	1.55E-06
50	2.02E-06	1.78E-06	2.64E-06	2.74E-06	1.08E-06	1.53E-06	2.02E-06	2.19E-06	1.77E-06	2.30E-06
100	3.40E-06	2.47E-06	2.92E-06	4.82E-06	1.18E-06	1.62E-06	2.40E-06	3.73E-06	2.44E-06	2.67E-06
500	1.47E-05	7.58E-06	3.82E-06	2.19E-05	1.47E-06	1.82E-06	3.27E-06	1.62E-05	6.78E-06	3.55E-06
1000	2.89E-05	1.35E-05	4.16E-06	4.32E-05	1.59E-06	1.86E-06	3.63E-06	3.15E-05	1.23E-05	3.93E-06
5000	1.41E-04	6.10E-05	5.13E-06	2.14E-04	1.91E-06	1.96E-06	4.61E-06	1.56E-04	5.29E-05	4.89E-06



3. Průměrné počty prvků v poli po vložení a smazání (counts.txt)

# m	UA_insert	SA_insert	BST_insert	UA delete	SA_delete	BST_delete
10	7	7	7	2	2	2
50	32	32	32	12	12	12
100	63	63	63	23	23	23
500	316	316	316	116	116	116
1000	633	633	633	232	232	232
5000	3160.5	3160.5	3160.5	1163.5	1163.5	1163.5

Při vkládání dat do zmíněných datových struktur se zhruba třetina dat v průměru zahodila kvůli duplicitám. Tento fakt měl zároveň vliv na to, že v datech vznikaly mezery, které mohly ovlivňovat další měření (například interpolační vyhledávání)

Z těchto výše zmíněných výstupů lze vyvodit několik závěrů:

1. **Unsorted array**
    a. Podle očekávání byly operace nad unsorted array nejpomalejší. Vložení, vyhledání a smazání jednoho elementu u této struktury podle očekávání mělo trvat nejdéle O(n), a tento předpoklad byl tímto ověřen. (OK)
    b. Insert, delete a search mají velice podobný průběh. Je to díky tomu, že největší část z algoritmu zabere hledání (složitost O(n)). Očekávání bylo ověřeno testem (OK)
2. **Sorted array**
    a. V sorted array se má v průměru lépe vyhledávat, ale pomaleji vkládat než unsorted array. Hledání je buď půlením intervalů, nebo interpolační – proto je mnohem rychlejší než u UA, kde se musí prohledat každý prvek. Vkládání do SA s sebou přináší nutnost přesouvat prvky tak, aby zůstala data seřazena. Všechny tyto předpoklady jsou potvrzeny měřením. Křivka mazání a vkládání mají podobný průběh, zatímco křivka znázorňující náročnost hledání znázorňuje daleko lepší výsledky. (OK)
    b. Křivka časové náročnosti mazání v podstatě kopíruje křivku vkládání. Je to hlavně díky tomu, že se musí prvky přesouvat, aby pole bylo seřazené a nemělo uvnitř mezery
    c. V testu byly zkoumány dva druhy hledání: interpolační a binární, u interpolačního je očekávaná průměrná složitost O(log(logn)) a u binárního O(log(n)), oba vyhledávací algoritmy by měly být rychlejší než je u unsorted array. Očekávání bylo ověřeno testem. (OK)
    d. Interpolační hledání (log(logn)) by mělo být rychlejší než binární (logn), za předpokladu, že jsou rovnoměrně rozložená data. V tomto testu bylo binární hledání rychlejší, což svědčí o tom, že při náhodném vkládání dat vznikají nerovnoměrné mezery mezi hledanými čísly. Pro potvrzení této teorie bylo provedeno ještě jedno měření, s nastavením programu nahodna_cisla="N", kde pak byla hledána data v polích bez mezer. V tomto dodatečném testu nebyly mezi čísly v poli mezery a interpolační hledání bylo vždy o něco rychlejší než binární hledání. (OK)

3. **Binary search tree**
    a. Algoritmus byl zkoumán na náhodných datech, kde pravděpodobně nebylo ani úplně ideální rozložení dat, ale ani úplně nejhorší rozložení dat. Tento fakt nejspíše způsobil to, že hledání v této struktuře bylo o něco pomalejší než u sorted array.

b. Algoritmus byl na polích větších než 100 prvků nejrychlejší z pohledu vkládání dat a velice rychlý z pohledu vyhledávání dat.

Tento test umožnil zajímavé porovnání chování algoritmů v závislosti na velikosti dat. Z grafu se dá například vyčíst, následující:

1. Do 50 prvků je rychlejší vkládání do UA a SA, a od 100 prvků výš je rychlejší práce s BST.
2. UA je od 50 prvků nejpomalejší ze všech. Od této velikost se nejspíše začíná projevovat neefektivita hledání, kdy se musí vždy projít celé pole prvek po prvku, aby se zjistilo, jestli se prvek může vložit, případně smazat.
3. Rychlost interpolačního hledání a binárního hledání na náhodných datech s náhodnými mezerami se při dostatečně velkém vzorku k sobě blíží. Tento fakt se nejspíše bude týkat samotného rozložení dat, v našem případě u pole s 5000 prvky.
4. Sorted Array měl sice pomalejší zápis dat, ale zároveň měl mezi všemi zkoumanými algoritmy nejlepší časy potřebné na vyhledávání. Tato výhoda se začíná objevovat již u práce s 10 prvky, a s narůstající velikostí polí byl rozdíl ještě více zřetelný.
5. BST se chová na náhodných datech velice „slušně“. Má sice o něco pomalejší vyhledávání dat než má SA, ale samotný zápis a mazání dat stojí daleko méně času, než vyžaduje SA na stejných náhodných datech.

5. Závěr

Většina původních předpokladů byla testy potvrzena. Našlo se ale pár vyjímek, u kterých by se dalo očekávat, že budou v průměru rychlejší/pomalejší než jiné algoritmy, ale nakonec byla realita jiná:

1. **Interpolační vyhledávání bylo o něco pomalejší než binární** – Bylo způsobeno tím, že náhodná data, ač setříděná do pole způsobila, že čísla mezi sebou obsahovala různě velké mezery.
2. **Vyhledávání v Sorted array bylo o něco rychlejší než u BST** – obě metody měly očekávanou náročnost O(log(n), u binárního stromu by se dala očekávat menší složitost, ale v našem případě zřejmě strom nebyl ideálně rozložen.

Dále se z výsledků testů dá odvodit, že vyšší časová náročnost u jednotlivých datových struktur a algoritmů se projevuje až od určitého množství prvků v poli.

- Nejpomalejší algoritmy ze všech zkoumaných byly všechny, které se týkaly unsorted array (find, insert, delete). Pomalost prováděných algoritmů nad touto strukturou se začíná více projevovat u polí, která jsou větší než 100 prvků.

- Algoritmus s nejrychlejším vyhledáváním ze všech srovnávaných byl binary_search a interpolation_search ze Sorted Array. U tohoto je ale časově náročnější plnění a mazání dat do této struktury. Zvýšená časová náročnost se začíná projevovat u polí, která jsou větší než 100 prvků.
- Datová struktura s nejrychlejším zápisem a mazáním u polí s více jak 100 prvky byla BST. Tato struktura má i relativně rychlé hledání prvků, je stále však pomalejší než hledání v SA. (u 500 prvků trvá dvakrát tak dlouho, u 5000 prvků trvá desetkrát tak dlouho)

6. Reference

1. **https://www.algoritmy.net/article/160/Interpolacni-vyhledavani. *www.algoritmy.net.* [Online] https://www.algoritmy.net/article/160/Interpolacni-vyhledavani.**
2. https://www.algoritmy.net/. [Online] https://www.algoritmy.net/.
3. http://bigocheatsheet.com/. *bigocheatsheet.* [Online] http://bigocheatsheet.com/.
4. https://en.wikipedia.org/wiki/Sorted_array.
https://en.wikipedia.org/wiki/Sorted_array. *wiki.* [Online]
https://en.wikipedia.org/wiki/Sorted_array.