

Porównanie drzew binarnych

Bartosz Rajczyk

19 maja 2019

1 Wstęp

Drzewo binarne T jest strukturą zdefiniowaną na skończonym zbiorze węzłów, która:

- nie zawiera żadnych węzłów, albo
- składa się z trzech rozłącznych zbiorów węzłów: korzenia, drzewa binarnego zwanego lewym poddrzewem i drzewa binarnego zwanego prawym poddrzewem.¹

W tym dokumencie zaprezentujemy przeprowadzone badania na trzech rodzajach drzew binarnych: binarnym drzewie przeszukiwań, drzewie czerwono-czarnym oraz drzewie splay. Algorytmy zostały zaimplementowane w języku Java.

2 Opisy algorytmów

2.1 BST

Drzewem wyszukiwań binarnych nazywamy drzewo binarne spełniające własność drzewa BST:

Niech x będzie węzłem drzewa BST. Jeśli y jest węzłem znajdującym się w lewym poddrzewie węzła, to $y.key \leq x.key$. Jeśli y jest węzłem znajdującym się w prawym poddrzewie węzła x , to $y.key \geq x.key$.²

gdzie przez $N.key$ rozumiemy wartość w danym węźle.

2.2 RBT

Drzewo czerwono czarne jest drzewem wyszukiwań, w którym na każdy węzeł przypada jeden dodatkowy bit informacji - kolor - który może przyjmować wartości *RED* lub *BLACK* (stąd nazwa). Spełnia ono następujące własności:

- każdy węzeł jest czarny lub czerwony
- korzeń jest czarny
- każdy liść (NIL) jest czarny
- jeżeli węzeł jest czerwony, to obaj jego synowie są czarni
- każda ścieżka prosta z ustalonego węzła do liścia ma tyle samo czarnych węzłów

Własności te sprawiają, że drzewo to zawsze będzie w przybliżeniu zrównoważone.³

2.3 Splay Tree

Drzewa splay są samodostosowującymi się drzewami wyszukiwań binarnych o zamortyzowanych kosztach wstawiania, usuwania i wyszukiwania $O(\log(n))$, gdzie n jest liczbą elementów w drzewie.⁴

¹Cormen, „Wprowadzenie do algorytmów”, ISBN 978-83-09-16911-4, strona 1201

²Cormen, „Wprowadzenie do algorytmów”, ISBN 978-83-09-16911-4, strona 288

³Cormen, „Wprowadzenie do algorytmów”, ISBN 978-83-09-16911-4, strony 309, 310

⁴<http://www.cs.cmu.edu/~sleator/papers/self-adjusting.pdf>

3 Metodologia

Testowane są trzy parametry:

- czas pracy algorytmu,
- liczba porównań,
- liczba przepięć węzłów

dla każdego z trzech typów drzew. Do drzew wstawiane są ciągi znaków rozpoczynające się zawsze literą, порядkiem na nich określonym jest porządek alfabetyczny. Ciągi znaków pochodzą z czterech plików testowych:

- `aspell_wordlist` - lista wszystkich słów w języku angielskim w porządku alfabetycznym,
- `kjb` - King James Bible,
- `lotr` - pierwsza część Władcy Pierścieni,
- `sample` - tekst instruktażowy obsługi Świętego Granatu Ręcznego z filmu "Monty Python i Święty Graal".

Najdłuższym plikiem jest *kjb*, najkrótszym - *sample*. Plik *aspell_wordlist* jest jedynym w kolejności alfabetycznej. Pozostałe można traktować w przybliżeniu jako będące w losowej kolejności. W przeprowadzonych testach słowa najpierw ładowane są do listy w kodzie, następnie dodawane do drzewa po kolei, wyszukiwane w drzewie po kolei oraz usuwane również po kolei.

4 Wyniki

Uzyskane wyniki prezentują poniższe wykresy:

- czasy działania:
 - `lotr`,
 - `sample`,
 - `kjb`,
 - `aspell_wordlist`
- liczba porównań,
- liczba przepięć

oraz tabelki z danymi:

- czasy działania:
 - `lotr`,
 - `sample`,
 - `kjb`,
 - `aspell_wordlist`
- liczba porównań,
- liczba przepięć

	insert	search	delete
rbt	89.56	38.48	61.36
splay	33.54	116.17	66169.80
bst	34332.69	44888.70	16.84

Tabela 1: czasy operacji w ms na pliku aspell

	insert	search	delete
rbt	682.99	173.57	366.41
splay	544.89	329.76	815.435
bst	405795.41	406.75	202.77

Tabela 2: czasy operacji w ms na pliku kjb

	insert	search	delete
rbt	178.23	62.03	94.67
splay	163.50	113.25	174.44
bst	7397.92	71.90	53.67

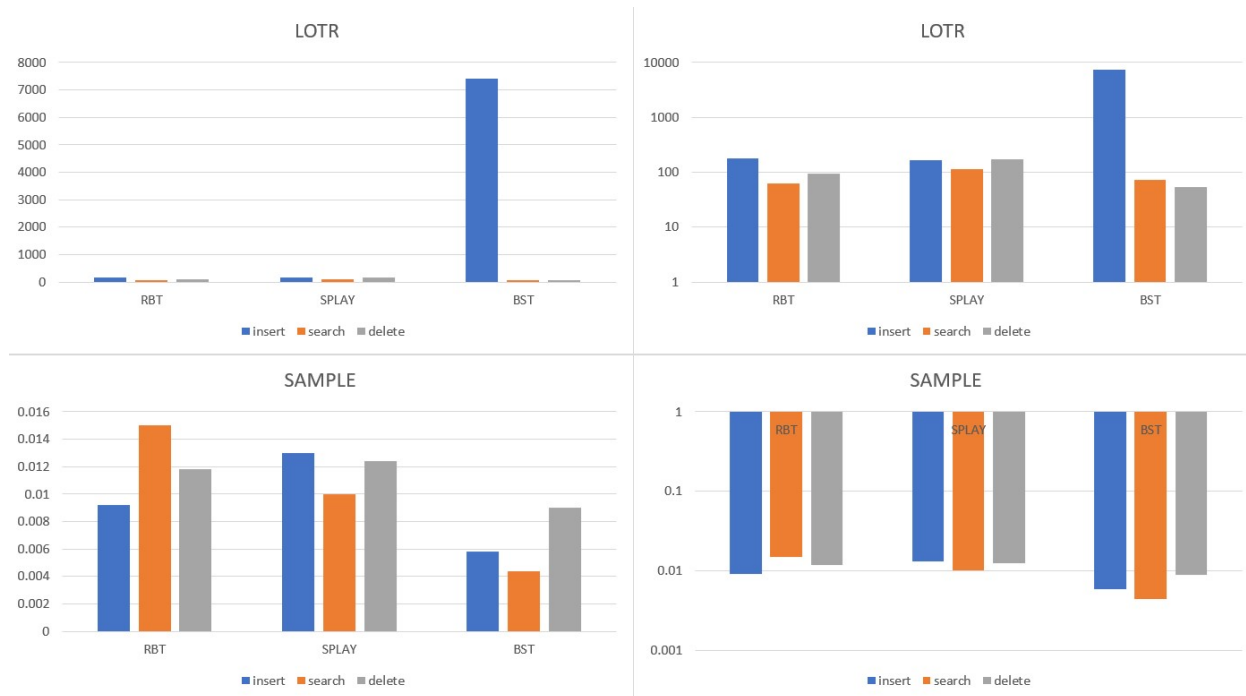
Tabela 3: czasy operacji w ms na pliku lotr

	insert	search	delete
rbt	0.0092	0.015	0.0118
splay	0.013	0.01	0.0124
bst	0.0058	0.0044	0.009

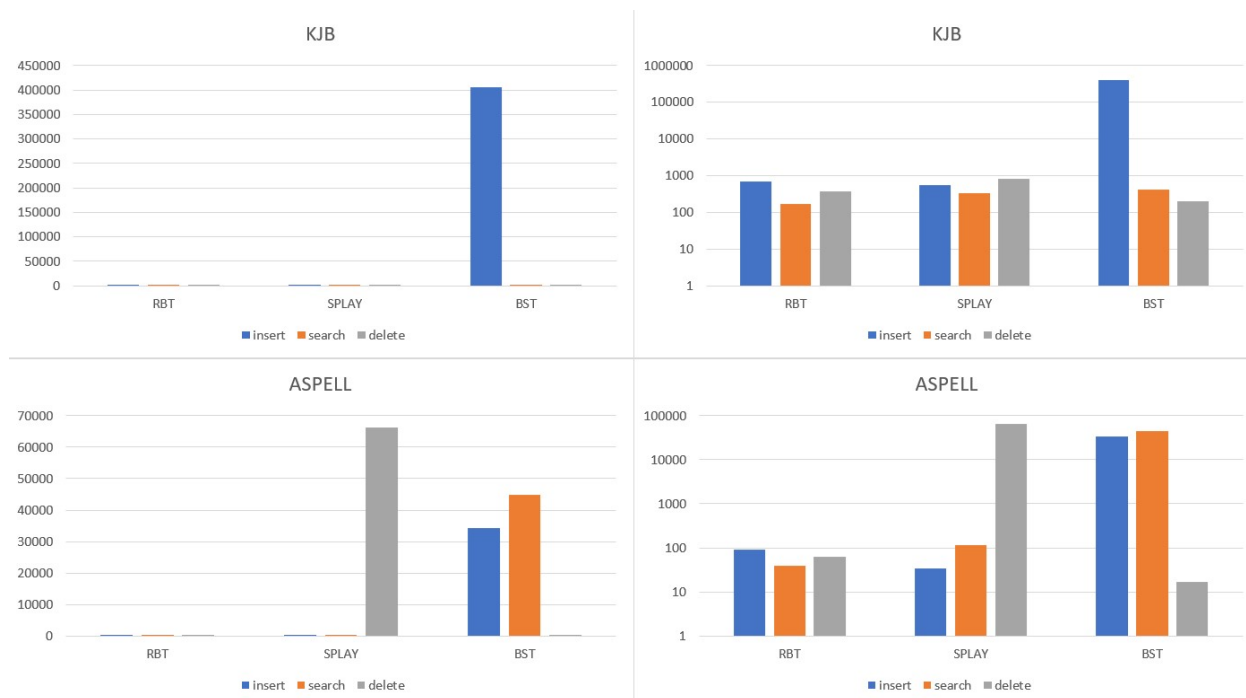
Tabela 4: czasy operacji w ms na pliku sample

plik	algo.	porównania	przepięcia
aspell	rbt	5403973	19910958
	splay	1742009	23811685818
	bst	7935165248	39674692474
kjb	rbt	37478562	106015737
	splay	81738426	263444125
	bst	4239212077	8546201352
lotr	rbt	7911101	22982699
	splay	21511318	69340916
	bst	152641493	323964616
sample	rbt	1003	2096
	splay	1960	6768
	bst	658	2418

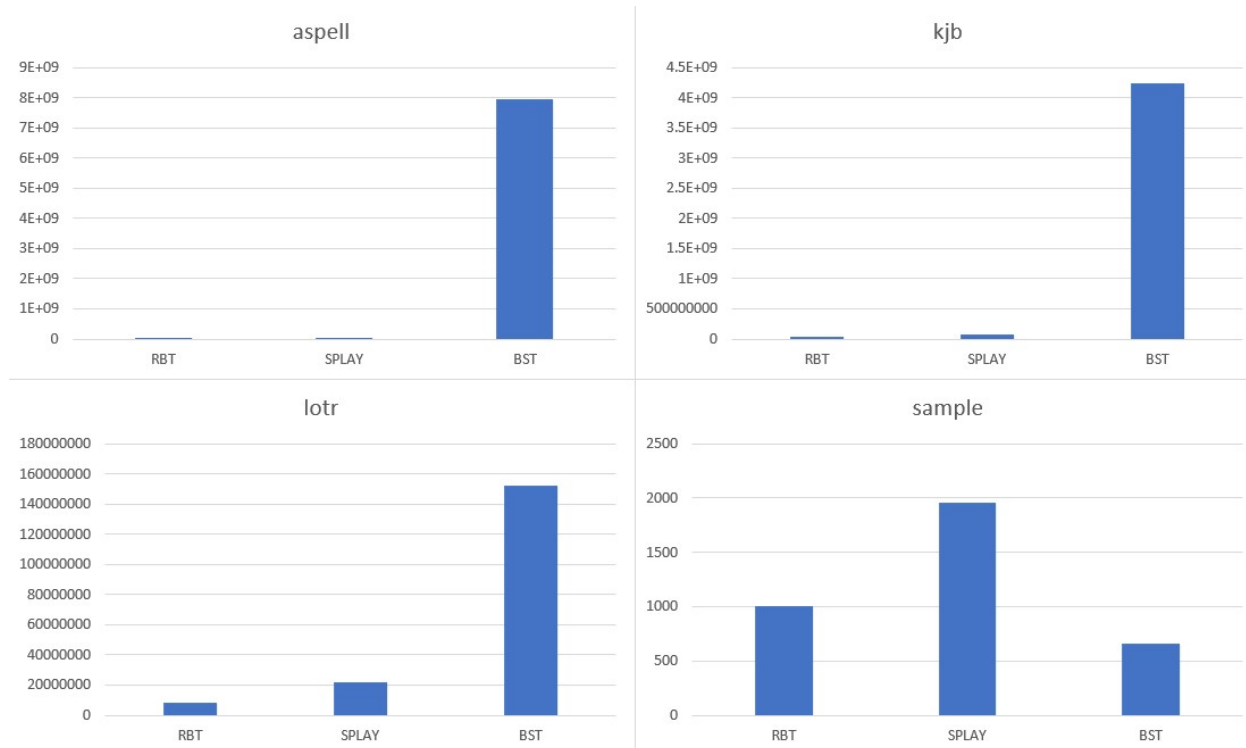
Tabela 5: porównania i przepięcia



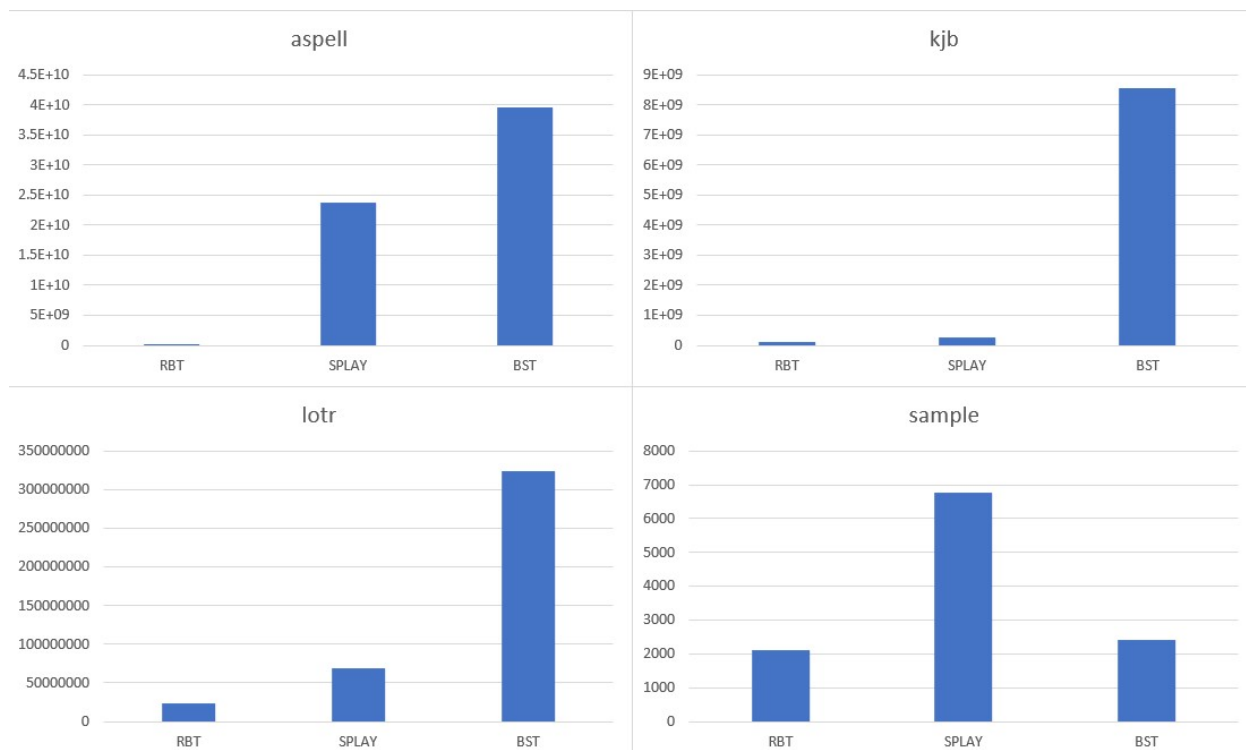
Rysunek 1: czas działania - lotr, sample



Rysunek 2: czas działania - kjb, aspell



Rysunek 3: liczba przebiegów węzłów



Rysunek 4: liczba porównań węzłów

5 Dane idealne

Spróbujemy wyznaczyć, dla jakich danych każde drzewo zachowuje się najlepiej.

5.1 BST

Aby uzyskać możliwie zbalansowane drzewo przeszukiwań binarnych możemy przyjąć następującą taktykę dla danych wejściowych A :

1. Posortować dane A
2. Wybrać środkowy element danych A i dodać do drzewa
3. Podzielić dane A na dwie połowy, wybrać ich środkowe elementy i dodać do drzewa
4. Kontynuować dzielenie na połowy i wybieranie ich środkowych elementów aż do wybrania wszystkich

W ten sposób zawsze wypełniamy po kolei kolejne rzędy drzewa aż do ich zapelnienia, a następnie przechodzimy do wypełniania rzędu niżej (środkowy element niższej części połowy tablicy idzie na lewo, wyższej - na prawo; rekurencyjnie). Dane te można wygenerować postępując wedle wymienionych wyżej kroków lub np. czytając je rzędami z drzewa RBT (bowiem to jest drzewem zbalansowanym). Dane testowe znajdują się tutaj.

5.2 RBT

Z przeprowadzonych testów wynika, że drzewo RBT zachowuje się najlepiej dla danych posortowanych, a te zostały sprawdzone już w przypadku pliku *aspell_wordlist*.

5.3 Splay

Splay najlepiej zachowuje się w przypadku częstego pytania o te same dane; jest to wynikiem sposobu jego działania, czyli przesuwaniu węzłów z kluczami, o które pytaliśmy, w górę drzewa. Można więc stwierdzić, że najlepiej będzie zachowywał się on dla danych losowych zyskując nad innymi drzewami w przypadku często powtarzanych zapytań.

algo.	insert	search	delete
bst	3.80586	2.80981	2.130020
splay	2.56725	3.46802	3.398935
rbt	3.35222	2.81433	2.422735

Tabela 6: średnia z 20 wywołań "idealnych" danych dla BST w ms

6 Wnioski

Z przeprowadzonych doświadczeń można wyciągnąć kilka wniosków:

- Zwykle drzewo BST nie jest najlepszym rozwiązaniem, jeżeli nasze dane nie są już odpowiednio uporządkowane. Jediną przewagę przy danych "losowych" w tym typie drzewa widzieliśmy przy bardzo małej ilości danych w pliku *sample*, co jest rezultatem prostoty wykorzystywanego algorytmu.
- Drzewa RBT są dosyć uniwersalnym typem drzewa dobrze zachowującym swoje własności dla wszystkich przetestowanych danych.
- Splay w niektórych testach potwierdza fakt, że jest drzewem o kosztach amortyzowanych $\log(n)$, a nie - tak jak RBT - rzeczywistych. Przy pracy z posortowanymi danymi jego wydajność znacząco spada.