Algorytmy i struktury danych - Złożone struktury danych

Dariusz Max Adamski

Wstęp

W tym sprawozdaniu porównywane będą czasy wykonywania czynności dodawania, wyszukiwania oraz usuwania elementów, w liście jednokierunkowej, drzewie poszukiwań binarnych BST i w dokładnie wyważonym BST.

Metodologia

Pomiary wykonywane były na tablicach o wielkości n od 1 000 elementów do 20 000 elementów, z krokiem 1 000 (20 punktów pomiarowych).

Uprzednio wygenerowano losowo listę 50 000 studentów (elementy uporządkowane rosnąco według indeksu). Przed rozpoczęciem pomiarów lista została załadowana do pamięci, i zapisana jako tablica S wskaźników $Student*$. W każdej iteracji wykonywane są poniższe kroki:

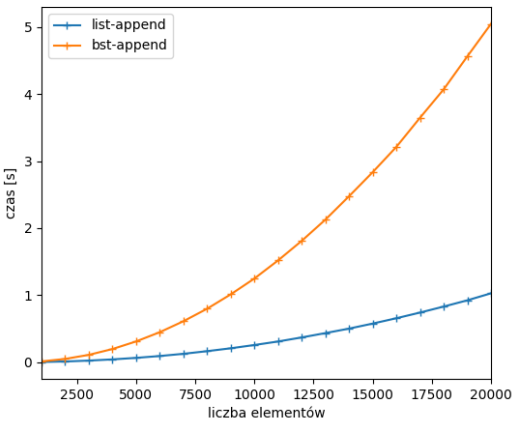
1. Tworzone jest $P := permutacja(0..n-1)$.
2. Tworzona jest struktura danych D.
3. Dla każdego i w $[0, n)$ student S_i jest dodawany do D.
4. Dla każdego i w P student S_i jest wyszukiwany w D.
5. Dla każdego i w $[0, n)$ student S_i jest usuwany z D.
6. Kroki 2 do 5 są powtarzane dla każdej testowanej struktury danych.

Czasy wykonywania kroku dodawania (append) i usuwania (remove) są przedstawione w sekundach, a czasy wykonywania kroku znajdywania (find) zostały podzielone przez n, i są przedstawione w milisekundach.

Aby utworzyć drzewo BBST w kroku 3 dodawana jest najpierw mediana $D_{0..n-1}$, a później rekurencyjnie mediany lewej i prawej podlist D, wydzielonych przez medianę.

Optymalizacja kompilatora została wyłączona flagą „-O0”. Czas wykonywania był mierzony w nanosekundach.

1 Dodawanie



Rysunek 1: Czas dodawania n elementów

Czynność dodawania n elementów ma złożoność $O(n^2)$ dla obu struktur.

Operacja dodawania elementu do BST ma złożoności $O(n)$, ponieważ elementy dodawane są według indeksów rosnąco (przypadek zdegenerowany). Gdyby dodawać elementy losowo, w średnim przypadku operacja ma złożoność $O(\log n)$.

Dla listy dodawanie (na koniec) także ma złożoność $O(n)$ ponieważ jest przechowywany wskaźnik do ostatniego elementu listy, więc trzeba przejść przez elementy w liście. Przy zachowanym wskaźniku dodawanie na koniec ma złożoność $O(1)$

BST w każdym kroku wykonuje dwa porównania, więc czas dodawania jest około 2 razy większy od dodawania do listy.

n	list	bst
1000	0.003289	0.011810
2000	0.010366	0.048701
3000	0.023749	0.109475
4000	0.041711	0.196439
5000	0.064503	0.309720
6000	0.092126	0.447733
7000	0.125115	0.611278
8000	0.164445	0.797717
9000	0.207542	1.012307
10000	0.256388	1.248592
11000	0.308848	1.518601
12000	0.369406	1.810808
13000	0.434191	2.127213
14000	0.501251	2.476297
15000	0.576299	2.835428
16000	0.655959	3.212760
17000	0.740366	3.647645
18000	0.829946	4.071280
19000	0.923338	4.563527
20000	1.027893	5.048294

Tablica 1: Czas dodawania n elementów (s)

n	list	bst
1000	0.006513	0.013936
2000	0.024612	0.055026
3000	0.055153	0.124716
4000	0.096888	0.221752
5000	0.150517	0.347778
6000	0.216285	0.501824
7000	0.294928	0.690220
8000	0.384989	0.903653
9000	0.485047	1.134099
10000	0.597422	1.402916
11000	0.724543	1.707715
12000	0.861291	2.022971
13000	1.012601	2.393878
14000	1.174417	2.760050
15000	1.345964	3.168627
16000	1.534628	3.623250
17000	1.730977	4.090354
18000	1.935672	4.569544
19000	2.153723	5.119397
20000	2.388142	5.651870

Tablica 2: Czas usuwania n elementów (s)

2 Usuwanie

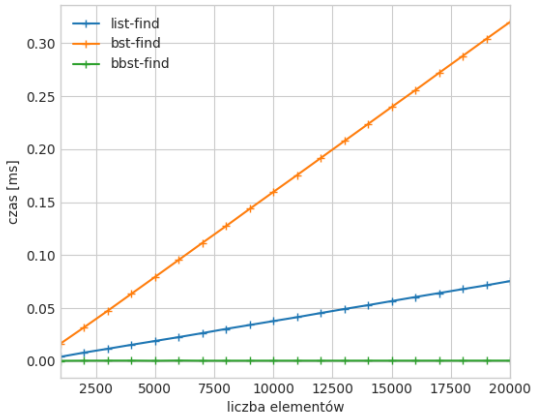
Usuwanie n elementów ma złożoność $O(n^2)$ dla obu struktur.

Ponieważ drzewo BST nie jest zbalansowane, usuwanie elementu ma złożoność $O(n)$, w średnim przypadku ta operacja ma złożoność $O(\log n)$.

Usuwanie z listy także ma złożoność $O(n)$. Tym razem nie musimy jednak przechodzić całej listy (zazwyczaj).

Czasy usuwania są większe od dodawania prawdopodobnie przez potrzebę uwalniania pamięci.

3 Znajdywanie



Rysunek 2: Średni czas znajdywania elementu

Znajdywanie elementu ma złożoność $O(n)$ w liście i zdegenerowanym BST, a w BBST $O(\log n)$. BST musi wykonywać więcej operacji porównań, więc jest wolniejsze od listy.

BBST bardzo szybko znajduje studentów. Dla 20 000 elementów w drzewie potrzebne jest maksymalnie 15 kroków aby dotrzeć do jakiegokolwiek z elementów, potrzeba do tego tylo paru mikrosekund.

Nawet dla 10 000 000 studentów, czyli największej ilości dozwolonej przez format indesku, wystarczą 24 kroki na znalezienie dowolnego studenta!

4 Wnioski

Podsumowując, opłaca się używać drzewa BST, jeżeli możemy sobie pozwolić na balansowanie drzewa, oraz bardziej skomplikowany algorytm.

n	list	bst	bbst
1000	0.004231	0.016690	0.000430
2000	0.008215	0.032072	0.000549
3000	0.011814	0.047713	0.000595
4000	0.015548	0.063703	0.000616
5000	0.019168	0.079625	0.000468
6000	0.022800	0.095685	0.000678
7000	0.026581	0.111733	0.000516
8000	0.030546	0.127593	0.000521
9000	0.034237	0.143810	0.000535
10000	0.037932	0.159774	0.000542
11000	0.041589	0.175649	0.000549
12000	0.045514	0.191723	0.000537
13000	0.049323	0.207780	0.000546
14000	0.052950	0.223813	0.000564
15000	0.056799	0.239908	0.000567
16000	0.060622	0.255765	0.000563
17000	0.064335	0.272016	0.000582
18000	0.068062	0.288011	0.000572
19000	0.071693	0.304130	0.000582
20000	0.075641	0.320039	0.000597

Tablica 3: Średni czas znajdywania elementu (ms)

W przypadku danych posortowanych, niezbalansowane drzewo BST jest wolniejsze w każdej kategorii od listy jednokierunkowej.

Ponadto gdy wiemy jaki jest ostatni element listy, dodanie kolejnego elementu do listy jest mniej kosztowne niż dodanie go do BST, nawet gdy jest ono zbalansowane (ale $\log n$ to nie jest tak dużo).

Alternatywnie, lista jednokierunkowa stanowi bardzo prosty zamiennik, który jest wystarczający dla większości zastosowań.