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 od 0 do $n-1$, student S_i jest dodawany do D.
4. Dla każdego i w P student S_i jest wyszukiwany w D.
5. Dla każdego i od $n-1$ do 0 student S_i jest usuwany z D.
6. Kroki 2 do 5 są powtarzane dla każdej testowanej struktury danych.

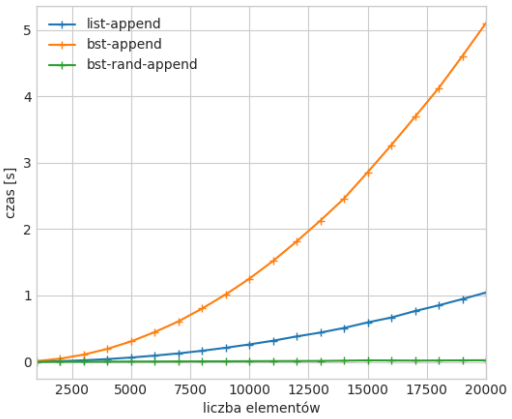
Czasy wykonywania kroku dodawania (append) i usuwania (remove) są przedstawione w sekundach, a czasy wykonywania kroku znajdywania (find) zostały podzielone przez n, i są przedstawione w milisekundach.

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

Aby zbadać przypadek średni BST, do struktury elementy były dodawane w losowej kolejności. Ten przypadek został oznaczony „bst-rand".

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 listy i BST. W losowym BST (bst-rand) złożoność to $O(\log n)$.

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

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

n	list	bst	bst-rand
1000	0.002661	0.011889	0.000589
2000	0.010666	0.047994	0.001286
3000	0.023610	0.109383	0.002030
4000	0.041885	0.195986	0.002962
5000	0.065355	0.308594	0.004163
6000	0.094211	0.448754	0.005042
7000	0.127729	0.609810	0.006246
8000	0.166954	0.802585	0.007339
9000	0.212617	1.015997	0.008508
10000	0.262925	1.252901	0.009542
11000	0.318462	1.520143	0.011066
12000	0.382884	1.814662	0.011968
13000	0.440970	2.128248	0.013442
14000	0.510665	2.460187	0.017374
15000	0.594712	2.856746	0.022724
16000	0.668024	3.265268	0.021312
17000	0.765135	3.690369	0.019138
18000	0.850640	4.122500	0.020796
19000	0.945453	4.604765	0.022472
20000	1.043897	5.102193	0.023676

Tablica 1: Czas dodawania n elementów (s)

n	list	bst	bst-rand
1000	0.006609	0.013653	0.000632
2000	0.024495	0.055007	0.001416
3000	0.054919	0.124721	0.001891
4000	0.096243	0.221804	0.002753
5000	0.149129	0.348087	0.003342
6000	0.214895	0.502943	0.004039
7000	0.293032	0.688971	0.004995
8000	0.391136	0.898184	0.005716
9000	0.482249	1.138674	0.006727
10000	0.605697	1.430359	0.007591
11000	0.724709	1.712778	0.007894
12000	0.857279	2.032565	0.009671
13000	1.005381	2.377777	0.009933
14000	1.165851	2.805722	0.010517
15000	1.349882	3.175463	0.013170
16000	1.524639	3.634742	0.013761
17000	1.724029	4.116298	0.013361
18000	1.929486	4.611285	0.014566
19000	2.142746	5.126889	0.014650
20000	2.374588	5.794622	0.015774

Tablica 2: Czas usuwania n elementów (s)

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

2 Usuwanie

Usuwanie n elementów, dla testowanych struktur danych, ma taką samą złożoność jak dodawanie.

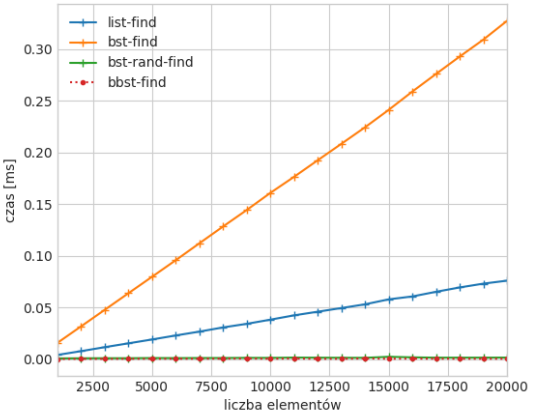
Usuwanie z listy także ma złożoność $O(n)$. Tym razem operacja może zakończyć się przed przejściem całej listy.

Podobnie jak w dodawaniu możemy zastosować trik. Usuwanie pierwszego elementu ma złożoność $O(1)$, więc jeżeli chcemy usunąć wszystkie elementy, złożoność tej operacji będzie $O(n)$.

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

3 Znajdywanie

Znajdywanie elementu ma złożoność $O(n)$ w liście i zdegenerowanym BST, a w losowym BST oraz BBST $O(\log n)$. Zdegenerowane



Rysunek 2: Średni czas znajdywania elementu

drzewo BST musi wykonywać więcej operacji porównań, więc jest wolniejsze od listy.

Losowe BST jest wolniejsze od drzewa BBST średnio o parę mikrosekund. Czas dostępu jest nadal bardzo dobry.

BBST bardzo szybko znajduje studentów. Dla 20 000 elementów w drzewie potrzebne jest maksymalnie 15 kroków aby dotrzeć do jakiegokolwiek z elementów, więc zajmuje to tylko kilka mikrosekund.

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

n	list	bst	bst-rand	bbst
1000	0.004244	0.015962	0.000864	0.000322
2000	0.007895	0.031924	0.000966	0.000471
3000	0.011752	0.047928	0.001034	0.000460
4000	0.015461	0.063974	0.001033	0.000522
5000	0.019198	0.079915	0.001209	0.000574
6000	0.023065	0.095926	0.001140	0.000574
7000	0.026805	0.112298	0.001223	0.000611
8000	0.030859	0.128535	0.001253	0.000677
9000	0.034333	0.144497	0.001376	0.000785
10000	0.038369	0.161070	0.001343	0.000589
11000	0.042525	0.176576	0.001664	0.000779
12000	0.046058	0.192620	0.001536	0.000496
13000	0.049603	0.208537	0.001460	0.000513
14000	0.053212	0.224537	0.001424	0.000837
15000	0.058018	0.241379	0.002411	0.000555
16000	0.060834	0.259103	0.001888	0.000702
17000	0.065323	0.276133	0.001554	0.000580
18000	0.069529	0.292969	0.001574	0.000584
19000	0.073227	0.309345	0.001579	0.000556
20000	0.076098	0.327500	0.001643	0.000558

Tablica 3: Średni czas znajdywania elementu (ms)

4 Wnioski

Podsumowując, opłaca się używać drzewa BST, jeżeli możemy sobie pozwolić na bardziej skomplikowany algorytm i jesteśmy gotowi je balansować.

W przypadku danych posortowanych, 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.

Jeżeli jednak mamy wszystkie dane dostępne (możemy je zapisać w losowej kolejności) i chcemy je zapisać w strukturze danych, drzewo BST jest dobrym wyborem.