Sprawozdanie z Laboratorium 5 - Pomiar czasu sortowaniu przy użyciu dwóch algorytmów - QuickSort oraz MergeSort.

Kamil Kuczaj

10 kwietnia 2016

1 Wstęp

Podanym zadaniem był pomiar czasu znajdywania losowego elementu listy typu*string*. Należało wykonać pomiary zapisu: 10^1, 10^3, 10^5, 10^6 oraz 10^9. Niestety przy próbie załadowania do tablicy 10^9 elementów komputer zatrzymywał się na wielkości 6 GB zużytej pamięci RAM i nie chciał alokować dalszej pamięci. Pomimo włączonego pliku stronicowania, który wielkością odpowiada ilości pamięci fizycznej w komputerze nie mogłem naprawić tego problemu. Podane dalej pomiary będą uwzględniać pomiary jedynie dochodzące do 10^6 elementów.

Nie byłem w stanie określić czasu szybkiego sortowania w przypadku pesymistycznym, gdyż gdzieś pojawił mi się nieoczekiwany błąd. Otóż pamięć alokowała się w nieskończoność powodując zatrzymanie pracy komputera (*freeze)*.

2 Specyfikacja komputera

Wersja kompilatora *g++*	4.8.4
System	Ubuntu 14.04.4
Procesor	Intel Core i5 2510M 2.3 GHz
Pamięć RAM	8 GB DDR3 1600 MHz
Rozmiar zmiennej *int*	4 bajty

3 Pomiary oraz ich interpretacja

Tablica 1: Wyniki pomiarów i ich średnie arytmetyczne.

MergeSort					**QuickSort**			
100	**1000**	**100000**	**1000000**		**100**	**1000**	**100000**	**1000000**
0	0	1	337		0	0	1	189
0	0	1	482		0	0	0	540
0	0	1	691		0	0	1	473
0	0	1	947		0	0	1	628
0	1	2	1118		0	0	1	760
0	0	1	1274		0	0	1	1175
0	0	2	1466		0	0	1	1221
0	0	2	1676		0	0	1	1352
0	1	2	1877		0	0	1	1423
0	0	2	2166		0	0	1	1556
0	0	2	2288		0	0	1	1658
0	1	2	2492		0	0	2	1743
0	0	2	2691		0	0	2	2070
0	0	2	2897		0	0	2	2219
0	1	3	3105		0	0	2	2276
0	0	2	3299		0	0	1	2444
0	0	3	3501		0	0	2	2658
0	0	3	3721		0	1	2	2830
0	0	3	3932		0	0	2	2918
0	1	4	4156		0	0	2	3013
0	0	4	4348		0	0	2	3096
0	1	4	4589		0	0	1	3229
0	0	5	4870		0	0	2	3367
0,02	**0,44**	**4,44**	**5533,5**	**T [ms]**	**0**	**0,12**	**2,12**	**3791,64**

Tablica 2: Wyniki pomiarów i ich średnie arytmetyczne.

MergeSort					**QuickSort**			
100	**1000**	**100000**	**1000000**		**100**	**1000**	**100000**	**1000000**
0	1	4	5047		0	0	2	3436
0	0	4	5251		0	0	2	3734
0	1	5	5501		0	0	2	3792
0	1	4	5658		0	0	2	3958
0	0	4	5924		0	0	3	4245
0	1	4	6073		0	0	3	4347
0	0	5	6287		0	0	3	4458
0	0	6	6453		0	1	2	4545
0	1	6	6710		0	0	2	4618
0	0	6	6875		0	0	2	4760
0	0	6	7101		0	1	2	4842
0	1	6	7341		0	0	3	4967
0	1	6	7577		0	0	3	5018
0	0	7	7801		0	0	3	5144
0	0	6	8119		0	1	3	5254
0	1	6	8299		0	0	3	5375
0	1	7	8714		0	0	3	5913
0	0	7	10785		0	0	3	6348
0	0	7	9320		0	1	3	6788
0	0	7	9805		0	1	3	6652
0	1	8	10867		0	0	3	6712
0	1	8	10646		0	0	3	7261
0	1	8	9796		0	0	3	6715
0	1	7	9954		0	0	3	6804
1	1	8	10232		0	0	3	6716
0	1	7	11878		0	0	3	6835
0	1	9	10738		0	0	4	7507
0,02	**0,44**	**4,44**	**5533,5**	**T [ms]**	**0**	**0,12**	**2,12**	**3791,64**

Czasy pomiarów obu metod wydłużają się wraz ze wzrostem numeru próby. Być może jest to spowodowane tym jak Linux radzi sobie z przypisywaniem priorytetów zadaniom, które długo wykonują się na komputerze. Czasy są ponad dwukrotnie dłuższe niż te początkowe. Zaburza to wynik początkowych pomiarów.

4 Wnioski

Wg slajdów doktora Jelenia, sortowanie przez scalanie (*MergeSort* zyskuje przewagę nad sortowaniem szybkim. Zwykle bierzemy pod uwagę ilości elementów większe od jednego miliona. Niestety, wskutek błędu, którego nie mogłem dopatrzyć się w kodzie kolegi, nie byłem w stanie tego stwierdzić. Wg moich pomiarów, algorytm *QuickSort* jest dużo szybszy od algorytmu *MergeSort*. Jest to również niezależne od ilości danych. Wg literatury oba te algorytmy w przypadku średnim posiadają złożoność obliczeniową rzędu $O(nlogn)$. Algorytm QuickSort w przypadku pesymistycznym, tj. gdy obierzemy sobie za tzw. *pivot* element maksymalny lub minimalny zbioru ma wtedy złożoność obliczeniową rzędu $O(n^2)$. Niestety nie byłem w stanie tego sprawdzić, o czym wspomniałem we wstępie tego sprawozdania.