Wojciech Krzaczek 184035 31.03.2012r

**Projektowanie algorytmów i metody sztucznej inteligencji**

**Lista nr 2: algorytmy sortowania, analiza efektywności algorytmów**

1. Wprowadzenie

Celem ćwiczenia jest zbadanie złożoności obliczeniowej i pamięciowej trzech algorytmów sortowania: Buble sort, Quick sort, Merge sort.

Badania algorytmów sortowania zostały wykonane dla różnego typu danych wejściowych (posortowanych lub losowych). Wyniki badań dla poszczególnych algorytmów zostały przedstawione w dalszej części sprawozdania.

Pomiar czasu został zrealizowany przy pomocy biblioteki time.h. Przed startem samego algorytmu zapisano aktualny czas systemowy w jako zmienną typu clock\_t za pomocą funkcji clock(). Podobny czas zanotowano na końcu działania algorytmu. Zmierzony czas jest to różnica pomiędzy czasem początkowym a końcowym. Podany wynik jest wyrażony w milisekundach. Wspomniana metoda nie należy do zbyt dokładnych, więc podane wartości czasu są wartościami średnimi obliczonymi na podstawie kilku prób.

1. Badane algorytmy

* Sortowane bąbelkowe (buble sort)

Porównywane są dwa sąsiednie elementy tablicy, na początku ostatni i przedostatni. Jeśli ich

kolejność jest niewłaściwa, to zamieniane są one miejscami. Porównanie powtarzane jest dla

pozostałych elementów. Po pierwszym przebiegu element najmniejszy znajdzie się na

początku tablicy (w części uporządkowanej). W kolejnych krokach procedura jest powtarzana

ale tylko dla nieuporządkowanej części tablicy. W ulepszonej wersji algorytmu sortowanie

powinno zostać zakończone w sytuacji, gdy po wykonaniu całego przebiegu nie zostanie

wykonana ani jedna zamiana.

|  |  |
| --- | --- |
| Przypadki | Złożoność obliczeniowa |
| Średni |  |
| Pesymistyczny |  |

Ze względu na duża złożoność obliczeniową tego algorytmu, badanie zostały zawężone do dziesięciokrotnego sortowania tablic o rozmiarach 10 000, 50 000, 100 000. Badania zostały przeprowadzone dla tablic o wartościach 0-1000: wypełnionych losowo, posortowanych odwrotnie, a także posortowanych w 25%, 50%, 75%, 95%, 99% i 99,7%.

Wyniki prezentują się następująco:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| rodzaj tablicy | ilość elementów | średni czas [ms] | rodzaj tablicy | ilość elementów | średni czas [ms] |
| elementy losowe | 10000 | **704,3** | 0,75 | 10000 | **262,8** |
| 50000 | **17132,7** | 50000 | **6582,1** |
| 100000 | **68480,1** | 100000 | **26279,3** |
| odwrotne | 10000 | **873,3** | 0,95 | 10000 | **235,2** |
| 50000 | **20879,2** | 50000 | **5852,7** |
| 100000 | **85709,5** | 100000 | **23451,2** |
| 0,25 | 10000 | **516,9** | 0,99 | 10000 | **238,4** |
| 50000 | **12441,1** | 50000 | **5848,8** |
| 100000 | **49720,4** | 100000 | **23389,2** |
| 0,5 | 10000 | **349,6** | 0,997 | 10000 | **233** |
| 50000 | **8754,2** | 50000 | **5836,3** |
| 100000 | **34812,4** | 100000 | **23361,3** |

* Sortowanie przez scalanie (merge sort)

Sortowana tablica dzielona jest rekurencyjnie na dwie podtablice aż do uzyskania tablic

jednoelementowych. Następnie podtablice te są scalane w odpowiedni sposób, dający w

rezultacie tablicę posortowaną. Wykorzystana jest tu metoda podziału problemu na mniejsze,

łatwiejsze do rozwiązania zadania („dziel i rządź”).

|  |  |
| --- | --- |
| Przypadki | Złożoność obliczeniowa |
| Średni |  |
| Pesymistyczny |  |

Badania zostały przeprowadzone dla tablic o wartościach 0-1000: wypełnionych losowo, posortowanych odwrotnie, a także posortowanych w 25%, 50%, 75%, 95%, 99% i 99,7%.

Wyniki prezentują się następująco:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| rodzaj tablicy | ilość elementów | średni czas [ms] | rodzaj tablicy | ilość elementów | średni czas [ms] |
| elementy losowe | 10000 | **3,61** | 75% | 10000 | **2,96** |
| 50000 | **19,9** | 50000 | **15,33** |
| 100000 | **40,25** | 100000 | **32,65** |
| 500000 | **219,17** | 500000 | **181,65** |
| 1000000 | **454,94** | 1000000 | **380,21** |
| odwrotnie | 10000 | **2,69** | 95% | 10000 | **2,64** |
| 50000 | **14,65** | 50000 | **14,67** |
| 100000 | **30,27** | 100000 | **31,08** |
| 500000 | **171,37** | 500000 | **174,75** |
| 1000000 | **359,64** | 1000000 | **366,21** |
| 25% | 10000 | **3,36** | 99% | 10000 | **2,61** |
| 50000 | **17,68** | 50000 | **14,72** |
| 100000 | **37,1** | 100000 | **30,91** |
| 500000 | **206,01** | 500000 | **173,7** |
| 1000000 | **429,72** | 1000000 | **364,4** |
| 50% | 10000 | **3,1** | 99,7% | 10000 | **3,01** |
| 50000 | **16,42** | 50000 | **14,65** |
| 100000 | **34,58** | 100000 | **30,87** |
| 500000 | **192,9** | 500000 | **173,67** |
| 1000000 | **403,28** | 1000000 | **368,39** |

* Sortowanie szybkie (quick sort)

Na początku wybierany jest tzw. element osiowy. Następnie tablica dzielona jest na dwie

podtablice. Pierwsza z nich zawiera elementy mniejsze od elementu osiowego, druga

elementy większe lub równe, element osiowy znajdzie się między nimi. Proces dzielenia

powtarzany jest aż do uzyskania tablic jednoelementowych, nie wymagających sortowania.

Właściwe sortowanie jest tu jakby ukryte w procesie przygotowania do sortowania. Wybór

elementu osiowego wpływa na równomierność podziału na podtablice (najprostszy wariant –

wybór pierwszego elementu tablicy – nie sprawdza się w przypadku, gdy tablica jest już

prawie uporządkowana).

|  |  |
| --- | --- |
| Przypadki | Złożoność obliczeniowa |
| Średni |  |
| Pesymistyczny |  |

Badania zostały przeprowadzone dla tablic o wartościach 0-1000: wypełnionych losowo, posortowanych odwrotnie, a także posortowanych w 25%, 50%, 75%, 95%, 99% i 99,7%.

Wyniki prezentują się następująco:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| rodzaj tablicy | ilość elementów | średni czas [ms] | rodzaj tablicy | ilość elementów | średni czas [ms] |
| elementy losowe | 10000 | **1,89** | 75% | 10000 | **1,55** |
| 50000 | **10,36** | 50000 | **8,76** |
| 100000 | **20,22** | 100000 | **17,41** |
| 500000 | **106,21** | 500000 | **93,06** |
| 1000000 | **222,69** | 1000000 | **196,83** |
| odwrotnie | 10000 | **0,99** | 95% | 10000 | **1,52** |
| 50000 | **7,27** | 50000 | **7,96** |
| 100000 | **10,37** | 100000 | **16,01** |
| 500000 | **62,47** | 500000 | **92,05** |
| 1000000 | **133,97** | 1000000 | **195,45** |
| 25% | 10000 | **1,66** | 99% | 10000 | **1,39** |
| 50000 | **9,59** | 50000 | **7,95** |
| 100000 | **18,3** | 100000 | **16,45** |
| 500000 | **100,22** | 500000 | **91,72** |
| 1000000 | **209,58** | 1000000 | **194,87** |
| 50% | 10000 | **1,61** | 99,7% | 10000 | **1,44** |
| 50000 | **9,31** | 50000 | **8** |
| 100000 | **17,45** | 100000 | **16,49** |
| 500000 | **96,64** | 500000 | **91,41** |
| 1000000 | **201,84** | 1000000 | **194,95** |

1. Wnioski

Jak widać na wykresach, z trzech przetestowanych algorytmów najszybsze jest *sortowanie szybkie*. Dodatkowo bardzo skraca czas sortowania posortowane elementów tablicy odwrotnie. Niewiele od niego wolniejszy jest algorytm *sortowania przez scalanie*. Dodatkową jego cechą jest to iż niezależnie od tego jak elementy są ułożone w tablicy, zawsze ma taką samą złożoność obliczeniową. Jego wadą jest to, że ma większą złożoność pamięciową, gdyż do wykonania tego sortowania niezbędne jest utworzenie drugiej tablicy – pomocniczej. Ostatnim testowanym algorytmem było *sortowanie bąbelkowe*. Jest ono bardzo powolne przez dużą złożoność obliczeniową - , co widać na wykresie po kształcie linii trendu. Dlatego badania zostały przeprowadzone dla mniejszej ilości elementów, ponieważ szacowany czas wykonania dla 1 000 000 elementów wynosił ok. 1h.

1. Literatura

* http://pl.wikipedia.org/wiki/Sortowanie\_b%C4%85belkowe
* http://pl.wikipedia.org/wiki/Sortowanie\_przez\_scalanie
* http://pl.wikipedia.org/wiki/Sortowanie\_szybkie