**Projektowanie Algorytmów i Metody Sztucznej Inteligencji**

Projekt 1  
**Algorytmy sortujące**

|  |  |
| --- | --- |
| Data oddania sprawozdania | 03.04.2019 |
| Imię i nazwisko | Kinga Tokarska 241621 |
| Nr indeksu | 241621 |
| Termin zajęć | Środa, 11.15-13.00 |
| Prowadzący kurs | Dr inż. Łukasz Jeleń |
| Kod kursu | E02-47f |

1. **Wstęp teoretyczny**

Sortowanie danych jest jednym z podstawowych, najczęściej rozwiązywanych problemów informatyki. Polega on na uporządkowaniu zbioru danych pewnego kryterium. Jest ono także doskonałym przykładem problemu, który można rozwiązać na wiele sposobów, różniących się od siebie efektywnością.

Miarą służącą do porównywania efektywności algorytmów, a jednocześnie jednym  
z najważniejszych parametrów charakteryzujących algorytm jest złożoność obliczeniowa, czyli ilość zasobów potrzebnych do rozwiązania problemów obliczeniowych. Złożoność obliczeniowa badana jest w dwóch podstawowych aspektach: czasu (złożoność czasowa) i pamięci (złożoność pamięciowa).

Rozróżniamy kilka rodzajów złożoności: pesymistyczna (określa ilość zasobów potrzebnych do wykonania algorytmu przy założeniu wystąpienia złośliwych lub najgorszych danych), typowa (określa ilość zasobów potrzebnych do wykonania algorytmu przy założeniu wystąpienia typowych lub oczekiwanych danych) oraz optymistyczna (określa ilość zasobów potrzebnych do wykonania algorytmu przy założeniu wystąpienia najlepszych danych).

1. **Opis badanych algorytmów**
   1. **Sortowanie przez scalanie**

Sortowanie przez scalanie jest algorytmem typu "dziel i zwyciężaj". Stanowi przykład algorytmu rekurencyjnego. Ideą działania sortowania przez scalanie jest dzielenie zbioru danych na mniejsze zbiory.

Algorytm sortowania przez scalanie polega na dzieleniu porządkowanego zbioru danych na kolejne połowy, dopóki taki podział jest możliwy, to znaczy do momentu uzyskania tablic jednoelementowych. Proces dzielenia następuje bez sprawdzania warunków. Tablice jednoelementowe uznajemy za posortowane. Kolejnym procesem jest połączenie uzyskanych uporządkowanych zbiorów w jeden, również uporządkowany zbiór. Operacja ta realizowana jest z wykorzystaniem zbioru pomocniczego, w którym tymczasowo przechowywane będą scalane elementy dwóch zbiorów. Zawartość zbioru głównego zostaje skopiowana do struktury pomocniczej, a wskaźniki ustawione na początkach kolejnych zbiorów. Następnie należy porównywać ze sobą pierwsze elementy każdego ze scalanych zbiorów, za każdym razem umieszczając mniejszą wartość w zbiorze tymczasowym i przesuwając wskaźnik o jeden. Czynność powtarza się aż do momentu osiągnięcia końca jednego ze zbiorów. Należy uwzględnić fakt, że wielkość zbiorów może być różna. Jeżeli koniec osiągnął zbiór pierwszy, elementy w zbiorze głównym są posortowane prawidłowo. W przypadku gdy zbiór drugi wcześniej osiągnie koniec, pozostałą zawartość zbioru pierwszego należy umieścić w zbiorze tymczasowym. Ostatni krok stanowi przepisanie zawartości zbioru tymczasowego do zbioru wynikowego.

Tablica wejściowa dzielona jest na pół przy każdym wywołaniu rekurencji, więc wysokość drzewa wywołań funkcji dla tablicy o rozmiarze wynosi . Złożoność operacji scalania tablic jest liniowa, każde scalanie to koszt . Wynikowa złożoność czasowa algorytmu wynosi więc, zarówno  
w przypadku optymistycznym, typowym, jak i pesymistycznym, .

* 1. **Sortowanie Shella**

Sortowanie Shella to algorytm, będący uogólnieniem sortowania przez wstawianie. Opiera się ono na podziale sortowanego zbioru na podzbiory, których elementy są od siebie odległe o pewien ustalony odstęp. Często nazywane jest sortowaniem przez wstawianie z malejącym odstępem.

Pierwszym krokiem jest wybór pewnej przerwy między elementami, które mają zostać posortowane. Każdy z otrzymanych podzbiorów należy posortować algorytmem sortowania przez wstawianie, a następnie zmniejszyć wielkość początkowo wybranej przerwy między elementami według pewnej reguły. W ten sposób powstają nowe podzbiory, z których każdy należy posortować przez wstawianie, a następnie ponownie zmniejszyć odstęp pomiędzy elementami. Operację sortowania powtarza się, za każdym razem zmniejszając odstęp, aż do momentu kiedy przerwa będzie miała wartość równą jeden. Odstęp taki nie dzieli zbioru wejściowego na podzbiory. Wówczas należy posortować przez wstawianie cały pozostały zbiór. Operacja ta nie jest skomplikowana, gdyż dzięki poprzednim obiegom sortującym zbiór został częściowo uporządkowany. Elementy o małych wartościach zbliżyły się do początku zbioru, z kolei te o większych wartościach do jego końca.

Efektywność algorytmu sortowania Shella w dużym stopniu zależna jest od ciągu przyjętych odstępów. Istnieje wiele sposobów wyboru kolejnych wyrazów z sortowanej tablicy. Każdy z nich charakteryzuje się inną złożonością obliczeniową. Pierwotnie Shell zaproponował pierwszy odstęp równy połowie liczby elementów w sortowanym zbiorze. Kolejne odstępy otrzymywał, dzieląc odstęp przez dwa. Złożoność obliczeniowa optymistyczna wynosi , a złożoność obliczeniowa typowa i pesymistyczna .

* 1. **Quicksort**

Sortowanie szybkie również jest przykładem algorytmu rekurencyjnego. Podobnie jak omawiane wcześniej sortowanie przez scalanie wykorzystuje technikę „dziel i zwyciężaj”.

W każdym kroku sortowania szybkiego wybrany zostaje jeden element w sortowanej tablicy, określany jako piwot. Może nim być zarówno pierwszy, środkowy, ostatni, mediana, jak i dowolny element wybrany według innego dostosowanego do zbioru danych schematu. Następnie algorytm porównuje z wybranym elementem wszystkie pozostałe elementy tablicy. Na początku zbioru umieszczone zostają dwa wskaźniki. Pierwszy z nich przebiega przez zbiór, wyszukując wartości mniejsze od piwotu. Po odnalezieniu takich elementów są one wymieniane z elementami na pozycji wskazywanej przez drugi wskaźnik. Po tej operacji drugi wskaźnik jest przesuwany na następną pozycję. Wskaźnik ten zapamiętuje pozycję, na którą trafi następny element oraz na końcu wskazuje miejsce, gdzie znajdzie się piwot. W ten sposób powstają dwie, niekoniecznie tych samych rozmiarów, partycje. Elementy nie większe od piwotu ustawione są po jego lewej stronie (w lewej partycji), natomiast nie mniejsze po prawej (w prawej partycji). Porządek elementów w partycji jest nieustalony. Położenie elementów o wartości równej wartości piwotu nie wpływa na proces sortowania, zatem mogą one występować w obu partycjach. Element wybrany do podziału znajduje się już na swojej prawidłowej pozycji i nie bierze udziału w dalszym sortowaniu. Kroki te są powtarzane aż do uzyskania posortowanej tablicy.

Złożoność obliczeniowa zależna jest od wyboru elementu rozdzielającego. Operacja przenoszenia elementów odbywa się w czasie liniowym. O złożoności obliczeniowej algorytmu decyduje więc to, ile nastąpi rekurencyjnych wywołań funkcji. W przypadku optymistycznym tablica będzie dzielona na pół. Wówczas głębokość drzewa wywołań zależy logarytmicznie od liczby danych wejściowych, a złożoność czasowa algorytmu wynosi . Przypadek pesymistyczny występuje, gdy piwot za każdym razem znajduje się na brzegu tablicy. Wówczas przy każdym kolejnym wywołaniu liczba elementów do posortowania jest tylko o 1 mniejsza, a funkcja zostanie wywołana n razy. Pesymistyczna złożoność czasowa wynosi więc .

1. **Przebieg ćwiczenia**

Ćwiczenie polegało na zaimplementowaniu trzech algorytmów sortujących: sortowanie przez scalanie, sortowanie Shella oraz sortowanie quicksort. W celu przeprowadzenia analizy efektywności  
i porównania ze sobą powyższych algorytmów przeprowadzono pomiary czasu sortowania stu tablic składających się z elementów typu całkowitoliczbowego o rozmiarach: , , , , o różnych stopniach wstępnego posortowania: , , , , , , oraz tablicy posortowanej odwrotnie.

Przeprowadzona została także wstępna weryfikacja poprawności sortowania zaimplementowanych algorytmów. Dla tablic o małych rozmiarach dokonano weryfikacji wizualnej.  
W celu sprawdzenia tablic o większych rozmiarach wykorzystano pomocniczą procedurę sprawdzającą poprawność uporządkowania elementów w tablicy.

Eksperymenty przeprowadzano, korzystając z komputera wyposażonego w procesor Intel Core i5-7200U (2.50 GHz, 2712 MHz, 2 rdzenie) i 8 GB pamięci RAM SO-DIMM DDR4.

Wyniki przeprowadzanych eksperymentów (minimalne, średnie oraz maksymalne wartości czasów sortowania o różnych rozmiarach i różnych stopniach wstępnego posortowania) przedstawiono  
w poniższych tabelach oraz na poniższych wykresach.

1. **Wyniki pomiarów**
   1. **Sortowanie przez scalanie**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |  |
| Wszystkie elementy losowe | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |
| Posortowane początkowe elementów | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |
| Posortowane początkowe elementów | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |
| Posortowane początkowe elementów | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |
| Posortowane początkowe elementów | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |
| Posortowane początkowe elementów | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |
| Posortowane początkowe elementów | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |
| Posortowane w odwrotnej kolejności | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |

Tab.1. Wyniki pomiarów dla sortowania przez scalanie

Rys. 1. Wykres zależności czasu sortowania od ilości danych dla sortowania przez scalanie

* 1. **Sortowanie Shella**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |  |
| Wszystkie elementy losowe | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |
| Posortowane początkowe elementów | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |
| Posortowane początkowe elementów | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |
| Posortowane początkowe elementów | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |
| Posortowane początkowe elementów | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |
| Posortowane początkowe elementów | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |
| Posortowane początkowe elementów | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |
| Posortowane w odwrotnej kolejności | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |

Tab.2. Wyniki pomiarów dla sortowania metodą Shella

Rys. 2. Wykres zależności czasu sortowania od ilości danych dla sortowania metodą Shella

* 1. **Sortowanie Quicksort**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |  |
| Wszystkie elementy losowe | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |
| Posortowane początkowe elementów | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |
| Posortowane początkowe elementów | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |
| Posortowane początkowe elementów | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |
| Posortowane początkowe elementów | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |
| Posortowane początkowe elementów | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |
| Posortowane początkowe elementów | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |
| Posortowane w odwrotnej kolejności | min |  |  |  |  |  |
| śr |  |  |  |  |  |
| max |  |  |  |  |  |

Tab.3. Wyniki pomiarów dla sortowania metodą quicksort

Rys. 3. Wykres zależności czasu sortowania od ilości danych dla sortowania metodą quicksort

1. **Wnioski**

Dla sortowania przez scalanie wszystkie otrzymane czasy (Tab. 1) są proporcjonalne do iloczynu , co oznacza, że złożoność obliczeniowa algorytmu wynosi . Najdłużej trwa sortowanie zbioru elementów losowych. Należy jednak zwrócić uwagę na fakt, że różnice w otrzymanych czasach są niewielkie, algorytm nie jest specjalnie czuły na rozkład danych wejściowych (Rys. 1). Przypadek optymistyczny i pesymistyczny nie występują.

Dla sortowania metodą Shella (Tab. 2) czas sortowania zbioru posortowanego odwrotnie jest znacznie krótszy od czasu sortowania zbioru losowego. Przypadkiem pesymistycznym jest więc tablica  
z elementami losowymi. Czym większy jest stopień posortowania tablicy początkowej, tym czas wykonania algorytmu jest krótszy, ale nawet przy posortowanych 99,7% tablicy, czas sortowania nie jest krótszy niż przy sortowaniu tablicy uporządkowanej odwrotnie (Rys. 2).

Dla sortowania szybkiego wszystkie otrzymane czasy (Tab. 3) są proporcjonalne do iloczynu , co oznacza, że złożoność obliczeniowa algorytmu wynosi . Czasy sortowania dla poszczególnych przypadków tablic (tablic losowych, częściowo uporządkowanych i posortowanych odwrotnie) są tego samego rzędu, nie udaje się wyodrębnić przypadku optymistycznego  
i pesymistycznego (Rys. 3).

Porównanie czasów działania badanych algorytmów pozwala stwierdzić, że najszybszą metodą sortowania jest algorytm sortowania szybkiego. Zarówno przy wykorzystaniu tablic losowych, jak  
i częściowo uporządkowanych, okazuje się on znacznie szybszy od sortowania przez scalanie  
i sortowania metodą Shella. Ze względu na swoją rekurencyjność najlepiej sprawdza się w przypadku sortowania większych ilości danych.

1. **Literatura**

*https://www.tutorialspoint.com/data\_structures\_algorithms/shell\_sort\_algorithm.htm*[*https://pl.wikipedia.org/wiki/Sortowanie\_przez\_scalanie*](https://pl.wikipedia.org/wiki/Sortowanie_przez_scalanie)[*https://pl.wikipedia.org/wiki/Sortowanie\_szybkie*](https://pl.wikipedia.org/wiki/Sortowanie_szybkie) *https://pl.wikipedia.org/wiki/Sortowanie\_Shella*