Projektowanie algorytmów i metody sztucznej inteligencji
Projekt
29.03.2021

Projekt 1
Sortowania

Autor
Mikołaj Zapotoczny
(252939)

Prowadzący
Mgr inż. Marta Emirsajłow

1 Wprowadzenie

W tym sprawozdaniu są zawarte: opis i testy algorytmów: quicksort, sortowanie przez scalanie oraz sortowanie Shella. Algorytmy były testowane zgodnie z zaleceniami zawartymi w opisie projektu. W programie, dzięki któremu mogłem przeprowadzić testy, zostały zawarte implementacje powyższych algorytmów wykorzystujące tylko podstawowe operacje, bez użycia funkcji wbudowanych. Ponadto program zawiera możliwość generowania tablic odwrotnych, co umożliwiło przeprowadzenie wszystkich testów.

Link do GitHuba: <KOD PROGRAMU>

2 Opis badanych algorytmów

2.1 Quicksort (Sortowanie szybkie)

- Na początku wybierany jest tzw. element osiowy (pivot). Następnie tablica dzielona jest na dwie podtablice. Jedna z tablic zawiera elementy mniejsze od pivota, a druga większe. I każdą podtablicę traktujemy w ten sam sposób. Proces dzielenia powtarzany jest aż do uzyskania tablic jednoelementowych, nie wymagających sortowania. Właściwe sortowanie jest tu jakby ukryte w procesie przygotowaniado sortowania. Wybór elementu osiowego wpływa na równomierność podziału na podtablice (najprostszy wariant, tzn. wybór pierwszego elementu tablicy, nie sprawdza się w przypadku, gdy tablica jest już prawie uporządkowana).

-
  - Złożoność obliczeniowa dla przypadków optymistycznego i średniego jest równa: $O(n * log(n))$, co czyni ten algorytm bardzo szybkim. Jest to przypadek kiedy naszym elementem osiowym jest liczba, której wartość znajduje się bliżej środka wartości elementów w tablicy
  - Złożonośc obliczeniowa dla przypadku pesymistycznego jest równa: $O(n^2)$. Ten przypadek jest wtedy kiedy naszym elementem osiowym będzie element, którego wartość jest graniczna, gdyż wtedy podział na dwie części tablicy jest nieefektywny.

- Podsumowując, jest to algorytm bardzo szybki dla przypadku średniego i optymistycznego. Jest jednak nieefektywny dla przypadku pesymistycznego, co czyni go niestabilnym. Jest łatwy w implementacji.

2.2 Sortowanie przez scalanie

- Sortowana tablica dzielona jest rekurencyjnie na dwie podtablice aż do uzyskania tablic jednoelementowych. Następnie podtablice te są scalane jednocześnie będąc kolejno sortowane. Wykorzystana jest tu metoda podziału problemu na mniejsze, łatwiejsze do rozwiązania zadania („dziel i rządź").

- Ciekawą cechą tego algorytmu jest to, że dla wszystkich przypadków ustawienia liczb wykonuje on tyle samo operacji i złożoność obliczeniowa jest taka sama dla przypadku optymistycznego, średniego oraz pesymistycznego wynosząca: $O(n * log(n))$.

- Jest to bardzo szybki oraz stabilny algorytm sortowania. Jego wadą jest to, że do implementacji jest potrzeba stworzenia dodatkowej tablicy, co zajmuje więcej pamięci.

2.3 Sortowanie Shella

- Algorytm sortowania metodą Shella jest ulepszonym algorytmem sortowania przez wstawianie. Sortowana tablica dzielona jest na kilka podtablic utworzonych z elementów oddalonych o ustaloną liczbę pozycji. Tablice te są oddzielnie sortowane. W kolejnych krokach wykonywane są podziały z coraz mniejszym odstępem (dające coraz mniejszą liczbę podtablic) aż do uzyskania tylko jednej tablicy. Kluczowym problemem jest dobór odpowiedniej sekwencji odstępów przy podziałach tablicy. Najlepsze efekty można uzyskać korzystając z zasady zaproponowanej przez D. Knutha.

- Efektywność algorytmu sortowania metodą Shella zależy w dużym stopniu od ciągu przyjętych odstępów. Pierwotnie Shell proponował pierwszy odstęp równy połowie liczby elementów w sortowanym zbiorze. Kolejne odstępy otrzymujemy dzieląc odstęp przez 2 (dzielenie całkowitoliczbowe), po czasie okazało się, że ten ciąg jest jednym z najgorszych. Nie można dlatego jednoznacznie stwierdzić jaka jest jego złożoność obliczeniowa. Przy zastosowaniu metody D. Knutha dla wszystkich trzech przypadków złożoność obliczeniowa waha się w okolicy $O(n)$.

- Jest to szybki, ale niestabilny. Jego wadą jest dowolność w wyborze odstępu, choć istnieją matematyczne sposoby na znalezienie najlepszego rozwiązania.

3 Przebieg eksperymentów

3.1 Omówienie

Testy zostały przeprowadzone dla wszystkich trzech algorytmów. Za każdym razem program sortował 100 tablic, o różnych rozmiarach. Program był uruchamiany dla każdej z wartości procentowej już posortowanych elementów. Dodatkowo zostało zaprezentowane sortowanie tablic gdy są już one posortowane, ale w odwrotnej kolejności. Wyniki są przedstawione w tabelach oznaczonych według legendy. Za każdym pojedyńczym razem używałem innych tablic. Wszystkie testy wykonałem 3 razy. Pojedyńczy test polega na wybraniu sortowania, wybraniu rodzaju ustawienia początkowego danych i wtedy program da nam średni czas sortowania jednej tablicy przy posortowaniu 100 tablic dla wszystkich pięciu rozmiarów osobno.

3.2 Wyniki

- % - w tabeli oznacza procent już posortowanych pierwszych elementów
- t - czas [ms]
- @ - dla macierzy posortowanej w odwrotnej kolejności
- x - rozmiar tablicy (w tysiącach)

3.2.1 Quicksort (Sortowanie szybkie)

%	t dla tablicy x elem.				
	10	**50**	**100**	**500**	**1000**
@	1085	7374	20718	96154	256119
@	1086	7419	20866	98771	268841
@	1087	7551	21573	100371	268330
0	1951	13235	37384	175264	464979
0	1957	13361	37698	176562	469404
0	2014	13627	38668	181868	477733
25	1913	13052	37327	176923	480041
25	1909	13461	37884	181996	487955
25	1917	13175	37424	175514	470202
50	4835	48381	147941	1033975	3232944
50	3879	47507	152419	1080404	3565732
50	4935	49773	151922	1065709	3462012
75	1642	11738	33706	161620	440956
75	1691	11923	34438	167276	447586
75	1668	12128	35047	167621	445652
95	1709	12441	35983	178657	488305
95	1707	12675	36708	181608	490606
95	1711	12937	38430	190460	504245
99	1437	10294	29766	148533	406896
99	1449	10453	30103	158006	427915
99	1571	10544	30217	149783	409242
99.7	1317	9583	27741	137392	373102
99.7	1318	9494	27213	133038	366456
99.7	1321	9596	27542	134805	367403

3.2.2 Wnioski: Quicksort

Algorytm sortowania szybkiego. Jako pivot jest wybierany element, którego indeks znajduje się w środku tablicy. Testy potwierdzają tezę, że kluczową sprawą tutaj jest wybór właśnie tego elementu osiowego. Dla przypadków gdy liczba posortowanych elementów (na początku) jest mniejsza niż 50% to wyniki są praktycznie takie same. Jednakże jest zauważalna ogólna tendencja zmniejszania się czasu sortowania od ponad 50% posortowania początkowego tablicy. Nie wynika to tylko z faktu, że skoro jest więcej posortowanych elementów, to jest łatwiej (choć dla ponad 50 % jest to ułatwieniem). Wynika to z faktu, że pivot wraz z wzrostem liczby elementów przyjmuje wartości, które znajdują się bliżej wartości środkowej tablicy, a więc podział na podtablice jest równomierny. Szczególnie rzuca się w oczy przypadek dla 50 % gdyż wtedy trafiamy na pesymistyczny przypadek, a więc taki gdy element skrajny lub prawie skrajny jest pivotem. W tym przypadku elementem osiowym jest element największy po lewej stronie, a więc możliwe, że jest to jeden z największych elementów tablicy. Jeżeli chodzi o przypadek z tablicą odwrotną, to jest to najlepszy przypadek, gdyż pivot wówczas znajduje się na środku tablicy, więc tablice są dzielone symetrycznie, co jest najważniejszym czynnikiem w tym algorytmie. Niestety dla różnych trzech przypadków wyniki czasami nie są porównywalne. Wynika to z losowego wyboru pivota i z trafiania na różne przypadki.

3.2.3 Sortowanie przez scalanie

%	t dla tablicy x elem.				
	10	**50**	**100**	**500**	**1000**
@	1459	10417	29399	135636	361108
@	1487	10592	30058	139699	367637
@	1587	10568	29608	136835	362197
0	2339	16047	45445	214254	568027
0	2337	16192	46255	219968	583812
0	2668	16378	45882	213340	576984
25	2133	14531	40912	192217	521540
25	2135	15079	43013	201162	530308
25	2130	14602	41369	193842	515072
50	1895	13110	37033	172830	457993
50	1898	13082	36813	175438	471467
50	1897	13012	36919	172314	455515
75	1643	11210	31546	147040	389552
75	1649	11763	33460	152544	400769
75	1641	11292	31856	149109	395965
95	1456	9914	27866	129076	341926
95	1455	10151	28858	136569	362674
95	1453	9889	27790	128808	344529
99	1415	9690	27346	126962	336275
99	1432	10114	28679	133802	352376
99	1419	9805	27704	130188	356598
99.7	1409	9628	27043	125219	331794
99.7	1419	9812	28453	129144	344731
99.7	1412	9845	27834	128938	337846

3.2.4 Wnioski: Sortowanie przez scalanie

Algorytm sortowania przez scalanie. Testy pokazują, że im więcej jest posortowanych elementów tym lepiej działa ten algorytm. Wynika to ze specyfiki tego sortowania, bo wtedy jest mniej przenoszenia elementów. Jeżeli chodzi o przypadek tablicy posortowanej w odwrotnej kolejności to jest on jednym z szybszych przypadków, co nie dziwi, bo elementy z lewj strony zgodnie przechodzą na lewo, a z lewej na prawą, co jest istotą tego algorytmu, lepszy przypadek jest tylko dla prawie wszystkich posortowanych elementów. Jest to jednak algorytm wolniejszy od sortowania szybkiego. Dodatkowo wymusza generowanie dodatkowej tablicy pomocniczej, co jest największą wada algorytmu.

3.2.5 Sortowanie Shella

%	t dla tablicy x elem.				
	10	**50**	**100**	**500**	**1000**
@	898	6327	17488	83743	221460
@	898	6348	17867	84739	219687
@	903	6443	17849	87719	229008
0	2531	19149	57206	313088	889744
0	2498	19017	57222	305815	870082
0	2510	18849	55916	303722	865913
25	2457	18564	55195	298136	850813
25	2415	18259	54573	302243	869178
25	2470	19106	57700	314344	892492
50	2318	17647	52447	284065	804045
50	2531	17981	53663	292257	832124
50	2573	17956	52081	286363	837980
75	2174	16781	50161	267851	756133
75	2117	15883	47128	250758	722924
75	2306	16257	48223	252229	713602
95	1846	13253	38819	203876	576000
95	1687	13130	39231	208423	591208
95	1672	12919	38558	206792	572306
99	1234	9967	30050	162155	456974
99	1221	9881	29851	160250	451454
99	1223	9853	29738	165372	468157
99.7	884	7684	24042	134466	385269
99.7	871	7683	24162	134264	382447
99.7	872	7544	23662	132563	379418

3.2.6 Wnioski: Sortowanie Shella

Algorytm sortowania Shella z wykorzystaniem odstępów zaproponowanych przez D. Knutha. Widzimy, że czas sortowania maleje wraz z wzrostem ilości posortowanych elementów na początku, choć zdarzają się wyjątki, co świadczy o niestabilności tego algorytmu.Gdy jest mało posortowanych elementów lub gdy są one posortowane w odwrotnej kolejności to algorytm należy do tych wolniejszych. Gdy jednak prawie wszystkie elementy tablicy są posortowane to wtedy ten algorytm jest najbardziej efektywny ze wszystkich. Takie też były założenia twórcy tego algorytmu - miał on usprawnić sortowanie przez wstawianie właśnie dla przypadku, gdy już prawie wszystkie elementy są posortowane i to zostało potwierdzone przez nasze testy. Dla sortowania tablic posortowanych w odwrotnej kolejności wyniki są najlepsze dla większych tablic i prawie najlepsze dla tablic mniejszych, co wynika z tego, że ten algorytm bazuje na uporządkowywaniu danych, a jeżeli na początku są one w pewie sposób uporządkowane to ten algorytm dla takiego zestawu jest wydajny. Dla danych posortowanych odwrotnie i prawie całkiem posortowanych ten algorytm działa najszybciej. Dla innych natomiast działa najwolniej.

4 Źródła informacji

- Opisy algorytmów zawarte w treści zadania
- http://algorytmy.ency.pl/artykul/quicksort
- http://www.algorytm.edu.pl/algorytmy-maturalne/quick-sort.html
- https://eduinf.waw.pl/inf/alg/003_sort/0013.php
- https://eduinf.waw.pl/inf/alg/003_sort/0012.php
- http://www.algorytm.edu.pl/algorytmy-maturalne/sortowanie-przez-scalanie.html