Algorytmy sortujące

Sortowanie bąbelkowe

Sortowanie bąbelkowe - wstęp

Algorytm sortowania bąbelkowego jest jednym z najstarszych algorytmów sortujących.

Zasada działania opiera się na cyklicznym porównywaniu par sąsiadujących elementów i zamianie ich kolejności w przypadku niespełnienia kryterium porządkowego zbioru. Operację tę wykonujemy dotąd, aż cały zbiór zostanie posortowany.

Sortowanie w tego typu algorytmie odbywa się w miejscu.

Sortowanie bąbelkowe - przykład

Jako przykład działania algorytmu sortowania bąbelkowego posortujemy przy jego pomocy pięcioelementowy zbiór liczb **[6 5 4 3 2]**, który wstępnie jest posortowany w kierunku odwrotnym, co możemy uznać za przypadek najbardziej niekorzystny, ponieważ wymaga przestawienia wszystkich elementów.

Obieg	Zbiór					Opis operacji
1	**6**	**5**	4	3	2	Rozpoczynamy od pierwszej pary, która wymaga wymiany elementów
	5	**6**	**4**	3	2	Druga para też wymaga zamiany elementów
	5	4	**6**	**3**	2	Wymagana wymiana elementów
	5	4	3	**6**	**2**	Ostatnia para również wymaga wymiany elementów
	5	4	3	**2**	**6**	Stan po pierwszym obiegu. Zwróć uwagę, iż najstarszy element **(6)** znalazł się na końcu zbioru, a najmłodszy **(2)** przesunął się o jedną pozycję w lewo.

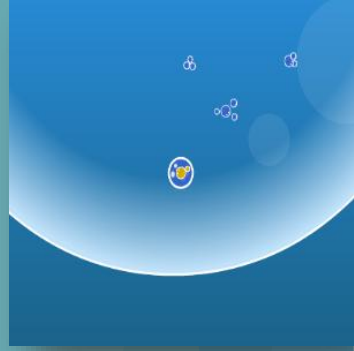
Sortowanie bąbelkowe – przykład c.d.

Obieg	Zbiór					Opis operacji
2	**5**	**4**	3	2	6	Para wymaga wymiany
	4	**5**	**3**	2	6	Para wymaga wymiany
	4	3	**5**	**2**	6	Para wymaga wymiany
	4	3	2	**5**	**6**	Elementy są w dobrej kolejności, zamiana nie jest konieczna.
	4	3	**2**	5	6	Stan po drugim obiegu. Zwróć uwagę, iż najmniejszy element **(2)** znów przesunął się o jedną pozycję w lewo. Z obserwacji tych można wywnioskować, iż po każdym obiegu najmniejszy element wędruje o jedną pozycję ku początkowi zbioru. Najstarszy element zajmuje natomiast swe miejsce końcowe.

Sortowanie bąbelkowe – przykład c.d.

Obieg	Zbiór					Opis operacji
3	**4**	**3**	2	5	6	Para wymaga wymiany
	3	**4**	**2**	5	6	Para wymaga wymiany
	3	2	**4**	**5**	6	Dobra kolejność
	3	2	4	**5**	**6**	Dobra kolejność
	3	**2**	**4**	**5**	**6**	Stan po trzecim obiegu. Wnioski te same.
4	**3**	**2**	4	5	6	Para wymaga wymiany
	2	**3**	**4**	5	6	Dobra kolejność
	2	3	**4**	**5**	6	Dobra kolejność
	2	3	4	**5**	**6**	Dobra kolejność
	2	**3**	**4**	**5**	**6**	Zbiór jest posortowany. Koniec

Sortowanie bąbelkowe – wnioski



Posortowanie przedstawionego zbioru wymaga 4 obiegów. Wynika to z tego, że w przypadku najbardziej niekorzystnym najmniejszy element znajduje się na samym końcu zbioru wejściowego.

Każdy obieg przesuwa go o jedną pozycję w kierunku początku zbioru. Takich przesunięć należy wykonać **n-1** (n - ilość elementów w zbiorze).

Uwaga!

Algorytm sortowania bąbelkowego jest uważany za **bardzo zły algorytm sortujący**. Można go stosować tylko dla **niewielkiej liczby elementów** w sortowanym zbiorze (do około 5000). Przy większych zbiorach czas sortowania może być zbyt długi.

Optymalizacja algorytmu

Przedstawiony algorytm sortowania bąbelkowego można zoptymalizować (ulepszyć) pod względem czasu wykonania. Po przeanalizowaniu obiegów wykonywanych w tym algorytmie, zauważamy, iż **po wykonaniu pełnego obiegu w algorytmie sortowania bąbelkowego najstarszy element wyznaczony przez przyjęty porządek zostaje umieszczony na swoim właściwym miejscu - na końcu zbioru.**

Wynika z tego, że w każdej kolejnej parze porównywanych elementów element starszy przechodzi na drugą pozycję. W kolejnej parze jest on na pierwszej pozycji, a skoro jest najstarszym, to po porównaniu znów przejdzie na pozycję drugą itd. - jest jakby ciągnięty na koniec zbioru (jak bąbelek powietrza wypływający na powierzchnię wody).

Optymalizacja algorytmu - przykład

Wykonamy jeden obieg sortujący dla zbioru pięcioelementowego **[9 4 2 7 0]**. Elementem najstarszym jest pierwszy element - liczba 9.

Obieg	Zbiór					Opis operacji
1	**9**	**4**	2	7	0	Para wymaga przestawienia elementów. Element najstarszy przejdzie na drugą pozycję w parze.
	4	**9**	**2**	7	0	Konieczne przestawienie elementów. Element najstarszy znów trafi na pozycję drugą w parze.
	4	2	**9**	**7**	0	Konieczne przestawienie elementów.
	4	2	7	**9**	**0**	Ostatnia para również wymaga przestawienia elementów.
	4	2	7	0	**9**	Koniec obiegu. Najstarszy element znalazł się na końcu zbioru.

Po każdym obiegu na końcu zbioru tworzy się podzbiór uporządkowanych najstarszych elementów. Zatem w kolejnych obiegach możemy pomijać sprawdzanie ostatnich elementów - liczebność zbioru do posortowania z każdym obiegiem maleje o 1.

Optymalizacja algorytmu – przykład c.d.

Dokończmy sortowania zbioru uwzględniając powyższe spostrzeżenia. Po pierwszym obiegu na końcu zbioru jest umieszczony element najstarszy. W drugim obiegu sortujemy zbiór 4 elementowy, w trzecim obiegu 3 elementowy i w obiegu ostatnim, czwartym - zbiór 2 elementowy.

Obieg	Zbiór					Opis operacji
2	**4**	**2**	7	0	**9**	Para wymaga przestawienia elementów.
	2	**4**	**7**	0	**9**	Dobra kolejność
	2	4	**7**	**0**	**9**	Konieczne przestawienie elementów.
	2	4	0	**7**	**9**	Koniec obiegu. Na końcu zbioru mamy 2 elementy uporządkowane.

Optymalizacja algorytmu – przykład c.d.

Obieg	Zbiór					Opis operacji
3	**2**	**4**	0	**7**	**9**	Dobra kolejność
	2	**4**	**0**	**7**	**9**	Para wymaga przestawienia elementów
	2	**0**	**4**	**7**	**9**	Koniec obiegu. Na końcu zbioru mamy 3 elementy uporządkowane
4	**2**	**0**	**4**	**7**	**9**	Konieczne przestawienie elementów.
	0	**2**	**4**	**7**	**9**	Koniec ostatniego obiegu - zbiór jest posortowany.

Porównując otrzymany wynik, do tabelki z Przykładu 1, nawet wzrokowo zauważamy istotne zmniejszenie ilości niezbędnych operacji do uzyskania tego samego efektu.

Optymalizacja algorytmu – wnioski

Z przedstawionych schematów wynika, iż ilość obiegów pętli wewnętrznej wynosi:

$$T_2(n) = (n-1) + (n-2) + \ldots + 2 + 1 = \frac{n(n-1)}{2} = \frac{1}{2}(n^2 - n)$$

Otrzymane wyrażenie nie jest jeszcze doskonałe (dla zainteresowanych: wciąż ma kwadratową klasę złożoności obliczeniowej). Osiągnęliśmy jednak większą efektywność działania dzięki wprowadzonym zmianom w porównaniu do poprzedniego przykładu.

Oczywiście możliwa jest jeszcze dalsza optymalizacja algorytmu.

Dalsza optymalizacja algorytmu

Algorytm sortowania bąbelkowego wykonuje dwa rodzaje operacji:

- test bez zamiany miejsc elementów – operacja ta nie sortuje zbioru, jest więc operacją pustą
- test ze zamianą miejsc elementów - operacja dokonuje faktycznej zmiany porządku elementów, jest zatem operacją sortującą

Ze względu na przyjęty sposób sortowania algorytm bąbelkowy zawsze musi wykonać tyle samo operacji sortujących. Tego nie możemy zmienić. Jednakże możemy wpłynąć na eliminację operacji pustych.
W ten sposób usprawnimy działanie algorytmu.

Po dokładnym przeanalizowaniu przykładu 1 i 2 można dokonać następujących spostrzeżeń:
- przykład 1 jest najmniej optymalną wersją algorytmu bąbelkowego. Wykonywane są wszystkie możliwe operacje sortujące i puste.
- przykład 2 redukuje ilość operacji pustych poprzez ograniczanie liczby obiegów pętli wewnętrznej (sortującej).

Możliwa jest dalsza redukcja operacji pustych, jeśli sprawdzimy, czy w pętli wewnętrznej były przestawiane elementy (czyli czy wykonano operacje sortujące). Jeśli nie, to zbiór jest już posortowany i możemy zakończyć pracę algorytmu.

Optymalizacja algorytmu – przykład c.d.

Posortujmy zbiór **[4 1 0 8 9]** zgodnie z wprowadzoną modyfikacją.

Obieg	Zbiór					Opis operacji
1	**4**	**1**	0	8	9	Para wymaga przestawienia elementów.
	1	**4**	**0**	8	9	Konieczne przestawienie elementów.
	1	0	**4**	**8**	9	Elementy w dobrej kolejności
	1	0	4	**8**	**9**	Elementy w dobrej kolejności
	1	0	4	8	**9**	Koniec pierwszego obiegu. Ponieważ były przestawienia elementów, sortowanie kontynuujemy

Optymalizacja algorytmu – przykład c.d.

Obieg	Zbiór					Opis operacji
2	1	0	4	8	**9**	Para wymaga przestawienia elementów
	0	**1**	**4**	8	**9**	Elementy w dobrej kolejności
	0	1	**4**	**8**	**9**	Elementy w dobrej kolejności
	0	1	4	8	9	Koniec drugiego obiegu. Było przestawienie elementów, zatem sortowanie kontynuujemy
3	**0**	**1**	4	8	9	Elementy w dobrej kolejności
	0	**1**	**4**	8	9	Elementy w dobrej kolejności
	0	**1**	**4**	8	9	Koniec trzeciego obiegu. Nie było przestawień elementów, kończymy sortowanie. Wykonaliśmy o 1 obieg sortujący mniej.

Bibliografia

- Sysło M, *Algorytmy* WSiP, Warszawa 1997

- John L Casti: *Five golden rules: great theories of 20th-century mathematics – and why they matter*. New York: Wiley-Interscience, 1996

- Donald E. Knuth: *Sztuka programowania*. T. 1. Warszawa: Wydawnictwo Naukowo-Techniczne, 2002

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: *Wprowadzenie do algorytmów*;WNT, wyd. VI 2004

- www.pl.wikipedia.org/

- www.algorytm.org/

- www.neuralnets.eu/

- www.wazniak.mimuw.edu.pl/

- www.computerchess.us/gtchess

- www.encyklopedia.pwn.pl/