Notatki z ASD do 1 kolokwium

Wersja 0.1 z dnia 23.11.2022

Najważniejsze algorytmy sortowań

Insertion sort

Złożoność algorytmu to $O(n - 1 + m)$, gdzie m jest liczbą inwersji (pary liczb w niepoprawnej kolejności) oraz n jest liczbą elementów w tablicy.
Bardzo prosty algorytm polegający na wstawianiu kolejnych elementów do tablic w odpowiednie miejsca.
Algorytm: stabilny(kolejność równych elementów się nie zmienia), w miejscu(nie zajmuje dodatkowej pamięci). Pesymistycznie kwadrat.

Bubble sort

Prosty algorytm sortujący kwadratory.
Liczba porównań: $\frac{n(n-1)}{2}$, liczba zamian: Inv($ciągu$) Inv to liczba inwersji.
Jest to algorytm w miejscu i stabilny.

Selection sort

Zawsze kwadratowy algorytm w miejscu, ale nie stabilny.
Polega na znajdowaniu elementu maksymalnego i wstawienie go na piersze miejsce.
Liczba porównań: $\frac{n(n-1)}{2}$, liczba zamian: $n - 1$

Merge sort

Algorytm stabilny, ale nie w miejscu o złożoności: $O(n \log n)$
Polega na dzieleniu ciągu na dwie równoliczne części w rekursji i łączenie ich potem.
Jest możliwa poprawa algorytmu aby był w miejscu, ale wtedy tracimy stabilność.

Quicksort

Algorytm podobny do merge sort, ale zamaist dzielić ciąg na dwa podciągi równe sobie długościowo, wybiera jeden element (zazwyczaj losowo ale można to zmienić np na mediane aby otrzymać drzewo o najniższej wysokości) i dzieli na dwa podciągi. W jednym znajdą się elementy mniejsze, a w drugim większe. Powtarzane jest to rekurencyjnie.
Pesimistyczna złożoność to $O(n^2)$ a oczekiwana to $O(n \log n)$
Łatwo jest utworzyś stabilny algorytm, ale nie jest on w miejscu.

Heap sort

Sortowanie jak kolejka priorytetowa. Kożysta z kopca czyli drzewa binarnego gdzie dla każdego noda jest on mnijszy/większy niż jego synowie.
Jest to algorytm w miejscu ale nie stabilny.
Złożoność: $O(n \log n)$
Można kopiec zbudować w czasie liniowym a operacje dodania i zabrania elementu jest w czasie logarytmicznym $O(\log n)$

Flaga polska

Sortowanie ciągu o dwóch możliwych wartościach.
Stabilny oraz w miejscu algorytm o złożoności $O(n \log n)$
Algorytm polega na zjandowaniu bloków 1..10..0
Zamieniamy takie bloki na poprawne(0..01..1) liniowo. Zauważmy, ze przy każdym wykonaniu takiego algorytmu powoduje zmniejszenie liczby bloków dwukrotnie, zatem złożonosc jest tak jak podana wyżej.
Algorytm można rozszeżyć aby sortował większą liczbę elementów poprzez utoższamianie elementów ze sobą. (Np dla ciągu o elementach $\{0, 1, 2\}$ najpierw utożsamiamy element 0 i 1 ze sobą sortujemy a potem elementy 1 i 2 i znowu sortujemy)

Sortowanie przez zliczanie (bucket sort)

Algorytm stabilny, **ale nie w miejscu**
Sortyjemy poprzez stworzenie kubełków w ilości liczby różnych elementów i zliczanie tych elementów w czasie liniowym.
Złożoność algorytmu: $O(n + m)$ gdzie n to liczba elementów, a m to liczba różnych elementów.

Sortowanie leksykograficzne

Zlożoność algorytmu: $O(R + m)$, gdzie m to wielkość alfabetu, a R to suma długości wszystkich słów.
Można sortować słowa o różnych długościach.

Inne algorytmy

Algorytm magicznych piątek

Algorytm pozwalający znaleźć k-ty element największy/najmniejszy.
Zlożoność: $O(n)$
Za pomocą niego można znaleźć np medianę. Łącząc to z algorytmem quicksort można ustabilizować jego złożoność do $O(n \log n)$
Za pomocą tego algorytmu można np posortować ciąg na ciąg k-dobry w złożoności $O(n \log \frac{n}{k})$

Drzewa decyzynje

Pełne drzewa binarne pozwalające na zbadanie maksymalnej zlożoności problemu. Można je zrobić kiedy nasz algorytm stosuje porównania (złożoności bucket sort nie da się udowodnić na drzewie decyzyjnym) lub inna funkcję binarną (zwracającą bool).
Pokazuje ono wszystkie możliwe kroki w naszym algorytmie. Jedna ścieżka w drzewie jest jakimś przypadkiem działania algorytmu.
Aby udowodnić **minimalną** (nie koniecznie zawsze osiągalną) złożonośc algorytmu.
Schemat liczenia złożoności jest następujący:
Należy policzyć wszystkie możliwe osiągane przez nasz algorytm permutacje ciągu, będą to liście w naszym drzwie. Następnie liczbymy maksymalną ścierzkę w drzewie do liścia (czyli wysokość).
Wzór: $\lceil \log(\text{liczba liści}) \rceil$
Przydatną aproksymacją, jest $\log n! \approx n \log n$

Koszt zamortyzowany

Metoda kosztu sumarycznego

Wykonajmy ciąg operacji $o_1, o_2, ..., o_n$
Niech c_i będzie rzeczywistym kosztem i-tej operacji.
Niech $C(n) = \sum_{i=1}^{n} c_i$ Wtedy koszt zamoryzowany i-tej operacji, oznaczony jako $\hat{c_i}$, jest równy: $\hat{c_i} = \frac{C(n)}{n}$

Metoda potencjału

Niech O_i będzie stanem naszego algorytmu, a funkcja $\phi(O_i)$ kredytem jaki posiada algorytm po wykonaniu operacji o_i
Wówczas kosztem zamortyzowanym jest $\hat{c_i} = c_i + \phi(O_i) - \phi(O_{i-1})$
$\sum_{i=1}^{n} \hat{c_i} = \sum_{i=1}^{n} c_i + \phi(O_n) - \phi(O_0)$
Zazwyczaj $\phi(O_0) = 0$ oraz $\phi(O_i) \geq 0$
Funkcję ϕ nalezy dobrze wyznaczyć tak aby zachowywała się jak niezmiennik w metodzie kredytowej