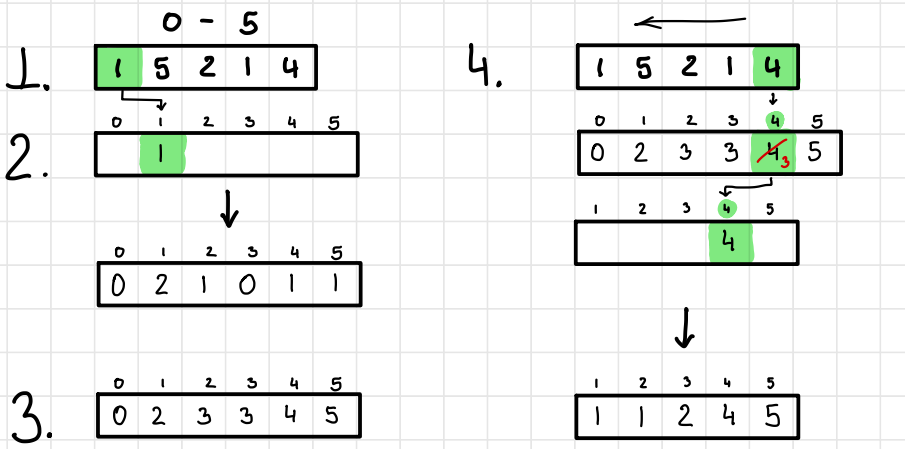
Sortowanie

Sortowanie przez zliczanie

1. Na wejściu dostajemy tablicę z wartościami od 0 do n
2. Zliczamy te wartości w osobnej tablicy
3. Do każdego elementu w tablicy zliczania dodajemy poprzedni
4. Iterując od końca tablicy wejściowej wstawiamy ten element w miejsce o indeksie, którego wartość znajdziemy w tablicy zliczania pod indeksem tego elementu. Zmniejszamy wartość zliczenia o 1.

to nam zapewnia stabilność sortowania

Przykład:



długość przedziału

Charakterystyka:

✓ Algorytm działa w czasie $O(n+k)$

✗ Dane są ograniczone

Counting Sort jest stabilny.

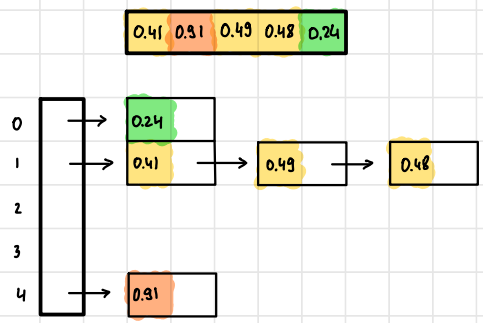
Sortowanie Kubełkowe

Sortowanie elementów 0-1 np. [0.41, 0.91, 0.49, 0.48, 0.24]

1. Stwórz listę n kubełków (array of linked lists)
2. Przypisz każdy element do kubełka wg. $\lfloor n \cdot arr[i] \rfloor$
3. Posortuj kubełki za pomocą insertSort

↑ który jest o mianę optymalny dla list związanych oraz jest stabilny.

Przykład:



Charakterystyka:

- ✓ złożoność oczekiwana $O(n)$
- ✗ dane z konkretnego przedziału
- ✗ zależy od danych (pesymistycznie $O(n^2)$)

Bucket Sort jest stabilny jeśli korzysta ze stabilnego alg. sortowania.

Sortowanie Leksykograficzne

a) Ciągi jednakowej długości

Spostrzeżenie: łatwo możemy zastosować sortowanie kubełkowe

Wtedy możemy rekurencyjnie stworzyć kubełek dla każdego znaku co da nam $O(dn)$ czasowo oraz $O(n^2)$ pamięciowo.

Lepszy sposób: Wykonać Radix Sort, który zoptymalizuje nam dodatkowo złożoność pamięciową.

Radix Sort stosuje Counting Sort dla każdej następnej litery zaczynając od tej najmniej istotnej. np. "abc" ← Zaczynamy od liter na pozycji trzeciej

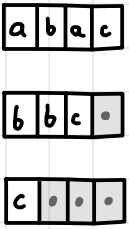
Wtedy mamy $O(d(n+k))$ gdzie d - długość słowa
n - liczba elementów
k - liczba liter w alfabecie

($n+k$: Counting Sort)

oraz $O(n)$ pamięciowo.

Radix Sort tak samo jak Counting Sort jest stabilny.

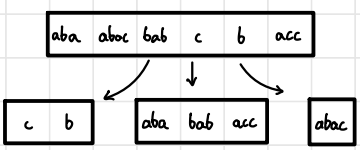
b) Ciągi różnej długości



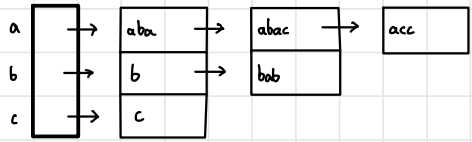
Możemy sprowadzić ten problem do tego z poprzedniego podpunktu poprzez dopełnianie z prawej strony znakami z min. wartością np spacja.

Inny sposób:

Podzielić tablicę słów na n tablic gdzie dla i-tej tablicy mamy słowa o długości i. Następnie wrzucamy do kubełków i sortujemy leksykograficznie. Ostatecznie łączymy wszystko w całość.



Dodajemy od najdłuższych do najkrótszych ciągów wykonując insert sorta porównując liczbę znaków z krótszym z ciągów.



Na końcu wszystko sklejamy

Lepszy sposób:

- Grupujemy słowa do tablic dla różnych długości słowa.
- Zaczynamy od tablicy z najdłuższymi słowami rozmiaru d

Algorytm

1. Dodaj słowa z tablicy słów rozmiaru d do tablicy wynikowej z lewej strony.
2. Posortuj tablicę Counting Sortem po znaku d od lewej

Przykład

$T = [a, b, ab, aa, bb, ba]$

1. ab, aa, bb, ba
2. aa, ab, ba, bb
3. a, b, aa, ab, ba, bb
4. a, aa, ab, b, ba, bb

Izomorfizm Drzew

1. Przejdź przez drzewo i przypisz każdemu węzłowi liczbę dzieci
2. Zacznij od korzenia
3. Wywołaj się rekurencyjnie - jeśli zwracany jest fałsz, to zwróć fałsz.
4. Weź dzieci tego węzła w drzewie T_1 i T_2, a następnie posortuj je po ilości ich dzieci z kroku pierwszego.
5. Jeśli posortowane dzieci węzła z drzewa T_1 są równe tym z drzewa T_2 pod kątem liczby ich dzieci, to zwróć prawda wpp. fałsz.