Counting Sort

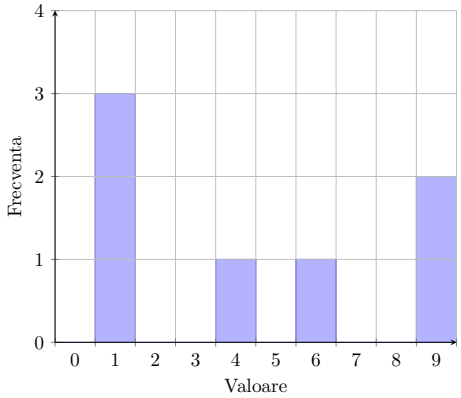
Counting sort

Este un algoritm ce se bazează pe faptul ca valorile care trebuie sortate se găsesc într-un anumit domeniu. Funcționează numărând de câte ori apare fiecare valoare din domeniul admisibil de valori în vector, iar pe baza frecventei de apariție se calculează pozițiile elementelor în cadrul vectorului de ieșire (i.e., vectorul sortat).

Algoritm counting sort cu exemplu

Fie șirul de numere `v`, de dimensiune `n = 7`, care conține valorile: `v = {1, 9, 4, 1, 1, 9, 6}`. Știind că avem de-a face cu numere întregi în intervalul $[0, 9]$, histograma lui `v` este:

```
Valoarea: 0. Numar aparitii: 0
Valoarea: 1.  Numar aparitii: 3
Valoarea: 2.  Numar aparitii: 0
Valoarea: 3.  Numar aparitii: 0
Valoarea: 4.  Numar aparitii: 1
Valoarea: 5.  Numar aparitii: 0
Valoarea: 6.  Numar aparitii: 1
Valoarea: 7.  Numar aparitii: 0
Valoarea: 8.  Numar aparitii: 0
Valoarea: 9.  Numar aparitii: 2
```



Pas 1: Se folosește vectorul de frecvențe `vf`, rezultat din calculul histogramei, care conține numărul de apariții al fiecărei valori din vectorul de sortat `v`:

index $= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, `vf` $= \{0, 3, 0, 0, 1, 0, 1, 0, 0, 2\}$.

Pas 2: Este parcurs vectorul de frecvențe, și fiecărui element se adaugă valoarea elementului anterior:

Operatie 1: `vaux` $= \{0, 3, 0, 0, 1, 0, 1, 0, 0, 2\}$

Operatie 2: `vaux` $= \{0, 3, 0, 0, 1, 0, 1, 0, 0, 2\}$

Operatie 3: `vaux` $= \{0, 3, 3, 0, 1, 0, 1, 0, 0, 2\}$

Operatie 4: `vaux` $= \{0, 3, 3, 3, 1, 0, 1, 0, 0, 2\}$

Operatie 5: `vaux` $= \{0, 3, 3, 3, 4, 0, 1, 0, 0, 2\}$

Operatie 6: `vaux` $= \{0, 3, 3, 3, 4, 4, 1, 0, 0, 2\}$

Operatie 7: `vaux` $= \{0, 3, 3, 3, 4, 4, 5, 0, 0, 2\}$

Operatie 8: `vaux` $= \{0, 3, 3, 3, 4, 4, 5, 5, 0, 2\}$

Operatie 9: `vaux` $= \{0, 3, 3, 3, 4, 4, 5, 5, 5, 2\}$

Operatie 10: `vaux` $= \{0, 3, 3, 3, 4, 4, 5, 5, 5, 7\}$

Pas 3: Se alege un vector de dimensiune `n`, în cazul nostru `n = 7`, în care o să stocăm valorile sortate. Pentru început, îl inițializăm cu valori din afara domeniului pentru a ține evidența elementelor modificate: `vs` $= \{-1, -1, -1, -1, -1, -1, -1\}$.

Pas 4: Pentru a completa vectorul sortat `vs`, vom parcurge vectorul inițial `v`, vom lua valoarea de pe indexul corespunzător din `vaux`, și o vom pune în vectorul sortat `vs` valoarea inițială din `v`. După ce vom pune valoarea în `vs`, vom decrementa valoarea corespunzătoare din `vaux`.

Operatie 1: `v` $= \{\mathbf{1}, 9, 4, 1, 1, 9, 6\}$, `vaux` $= \{0, \mathbf{3}, 3, 3, 4, 4, 5, 5, 5, 7\}$, `vs` $= \{-1, -1, \mathbf{-1}, -1, -1, -1, -1\}$
`v` $= \{\mathbf{1}, 9, 4, 1, 1, 9, 6\}$, `vaux` $= \{0, \mathbf{2}, 3, 3, 4, 4, 5, 5, 5, 7\}$, `vs` $= \{-1, -1, \mathbf{1}, -1, -1, -1, -1\}$

Operatie 2: `v` $= \{1, \mathbf{9}, 4, 1, 1, 9, 6\}$, `vaux` $= \{0, 2, 3, 3, 4, 4, 5, 5, 5, \mathbf{7}\}$, `vs` $= \{-1, -1, 1, -1, -1, -1, \mathbf{-1}\}$
`v` $= \{1, \mathbf{9}, 4, 1, 1, 9, 6\}$, `vaux` $= \{0, 2, 3, 3, 4, 4, 5, 5, 5, \mathbf{6}\}$, `vs` $= \{-1, -1, 1, -1, -1, -1, \mathbf{9}\}$

Operatie 3: `v` $= \{1, 9, \mathbf{4}, 1, 1, 9, 6\}$, `vaux` $= \{0, 2, 3, 3, \mathbf{4}, 4, 5, 5, 5, 6\}$, `vs` $= \{-1, -1, 1, \mathbf{-1}, -1, -1, 9\}$
`v` $= \{1, 9, \mathbf{4}, 1, 1, 9, 6\}$, `vaux` $= \{0, 2, 3, 3, \mathbf{3}, 4, 5, 5, 5, 6\}$, `vs` $= \{-1, -1, 1, \mathbf{4}, -1, -1, 9\}$

Operatie 4: `v` $= \{1, 9, 4, \mathbf{1}, 1, 9, 6\}$, `vaux` $= \{0, \mathbf{2}, 3, 3, 3, 4, 5, 5, 5, 6\}$, `vs` $= \{-1, \mathbf{-1}, 1, 4, -1, -1, 9\}$
`v` $= \{1, 9, 4, \mathbf{1}, 1, 9, 6\}$, `vaux` $= \{0, \mathbf{1}, 3, 3, 3, 4, 5, 5, 5, 6\}$, `vs` $= \{-1, \mathbf{1}, 1, 4, -1, -1, 9\}$
. . . . . .

Se continuă să se itereze până la finalul vectorului, astfel încât se va obține vectorul sortat: `vs` $= \{1, 1, 1, 4, 6, 9, 9\}$.

Aplicație (versiunea 1)

Se consideră următoarea structură:

```
struct Pair {
    int key; // Valori  unice 0-99
    int val;
};
```

Se cere:

1. Creați o bibliotecă pentru procesarea datelor de tipul `Pair`.
2. (1p) Implementați o funcție pentru citirea datelor de la tastatură pentru o variabila de tip `Pair`. Creați și citiți un vector de tip `Pair` alocat dinamic, denumit `map`.
3. (4p) Sortați vectorul `map` folosind merge sort (implementarea în cursul 2), criteriul de sortare fiind câmpul *key* al fiecărui element de tip `Pair` din vector.
4. (3p) Realizați pe foaie pașii pentru sortarea vectorului de la cerința anterioara pentru metoda de sortare.
5. (2p) Este această metodă de sortare stabilă? Care este complexitatea? Cum se încarcă stack-ul la apelul funcției implementate (luați ca exemplu un vector la întâmplare)?

Bonus (1p): Sortați vectorul `map` folosind counting sort, criteriul de sortare fiind câmpul *key* al fiecărui element de tip `Pair` din vector.

Aplicație (versiunea 2)

Se consideră algoritmul standard quick sort, prezentat în Cursul 2. Se cere:

1. Creați o bibliotecă care va procesa valori de tipul `float`.
2. (1p) Creați o funcție pentru alocarea dinamică, și o funcție pentru citirea unui vector de tip `float`, denumit `v`.
3. (4p) Sortați **descrescător** vectorul folosind quick sort (implementare cursul 2).
4. (3p) Adaptați algoritmul astfel încât să alegeți ca pivot valoarea mediană dintre primul element, ultimul element, și elementul de la mijlocul vectorului (discuție în cursul 2). Realizați pe foaie pașii pentru sortarea modificată vectorului (folosind un exemplu propriu). Implementați sortarea, și testați codul.
5. (2p) Este această metodă de sortare stabilă? Care este complexitatea? Cum se încarcă stack-ul la apelul funcției implementate (luați ca exemplu un vector la întâmplare)?

Bonus (1p): Implementați counting sort, și sortați vectorul `v` convertit la vector de întregi.

Aplicație (versiunea 3)

Se consideră algoritmul standard quick sort, prezentat în Cursul 2. Se cere:

1. Creați o bibliotecă care va procesa valori de tipul `double`.
2. (1p) Creați o funcție pentru alocarea dinamică, și o funcție pentru citirea unui vector de tip `double`, denumit `v`.
3. (4p) Sortați vectorul `v` folosind quick sort (implementarea în cursul 2). Adaptați algoritmul astfel încât să grupați valorile egale cu pivotul (vedeți discuție curs 2).
4. (3p) Realizați pe foaie pașii pentru sortarea modificată vectorului (folosind un exemplu propriu). Implementați sortarea, și testați codul.
5. (2p) Este această metodă de sortare stabilă? Care este complexitatea? Cum se încarcă stack-ul la apelul funcției implementate (luați ca exemplu un vector la întâmplare)?

Bonus (1p): Implementați counting sort, și sortați vectorul `v` convertit la vector de întregi.