**Ordenación por distribución**

**1. Introducción**

El algoritmo de ordenación por distribución, también conocido como "counting sort", es un método de ordenación eficiente que funciona bien cuando el rango de valores de la entrada es relativamente pequeño en comparación con el tamaño de la entrada. Este algoritmo no compara los elementos de la lista como lo hacen los algoritmos de ordenación comparativa, como el quicksort o el mergesort, lo que lo hace especialmente útil en situaciones donde los valores a ordenar son números enteros dentro de un rango limitado.

**2. Definición:**

El algoritmo de ordenación por distribución funciona contando el número de ocurrencias de cada elemento único en la lista de entrada y utilizando esta información para reconstruir una lista ordenada. Es especialmente eficiente cuando el rango de valores es pequeño y conocido de antemano.

**3. Características:**

* Eficiente para listas con un rango de valores pequeño.
* No compara directamente los elementos de la lista.
* Requiere conocimiento previo del rango de valores de la lista.
* Tiene una complejidad temporal lineal O(n + k), donde "n" es el tamaño de la lista y "k" es el rango de valores.

**4. Ejemplo:**

Supongamos que queremos ordenar una lista de números enteros utilizando el algoritmo de ordenación por distribución implementado en C++ de forma orientada a objetos y separada por archivos.

***CountingSort.h***

#ifndef ORDENAMIENTO\_COUNTINGSORT\_H

#define ORDENAMIENTO\_COUNTINGSORT\_H

#include <list>

**class** **CountingSort** {

**public:**

**static** **void** ordenar(std::list<**int**>& lst);

};

#endif

***CountingSort.cpp***

#include "CountingSort.h"

#include <algorithm>

**void** CountingSort::ordenar(std::list<**int**>& lst) {

**int** max = \*std::max\_element(lst.begin(), lst.end());

**int** min = \*std::min\_element(lst.begin(), lst.end());

**int** rango = max - min + **1**;

std::list<**int**> conteo(rango), salida;

**for** (**int** num : lst) {

conteo[num - min]++;

}

**for** (**auto** it = conteo.begin(); std::next(it) != conteo.end(); ++it) {

\*std::next(it) += \*it;

}

**for** (**auto** it = lst.rbegin(); it != lst.rend(); ++it) {

salida.emplace\_front(\*it);

conteo[\*it - min]--;

}

lst = salida;

}

***main.cpp***

#include <iostream>

#include <list>

#include "CountingSort.h"

**int** **main**() {

std::list<**int**> lst = {**4**, **2**, **2**, **8**, **3**, **3**, **1**};

std::cout << "Lista original: ";

**for** (**int** num : lst) {

std::cout << num << " ";

}

std::cout << std::endl;

CountingSort::ordenar(lst);

std::cout << "Lista ordenada: ";

**for** (**int** num : lst) {

std::cout << num << " ";

}

std::cout << std::endl;

**return** **0**;

}

**5. Conclusiones**

El algoritmo de ordenación por distribución, conocido también como "counting sort", representa una herramienta valiosa en el arsenal de algoritmos de ordenación gracias a su eficiencia en situaciones específicas. A diferencia de los algoritmos de comparación, como el quicksort o el mergesort, que operan comparando directamente los elementos de la lista, el counting sort trabaja contando las ocurrencias de cada elemento único y luego utilizando esta información para ordenar la lista. Esta característica lo hace particularmente eficiente cuando se trabaja con un rango de valores limitado, ya que su complejidad temporal es lineal en relación con el tamaño de la lista más el tamaño del rango de valores. Sin embargo, esta eficiencia está condicionada por la necesidad de conocer previamente el rango de valores, lo que puede limitar su aplicabilidad en situaciones donde este rango es desconocido o muy amplio. A pesar de estas limitaciones, el counting sort sigue siendo una opción valiosa en escenarios donde se cumple su suposición fundamental de un rango de valores acotado, proporcionando una alternativa eficiente y sencilla para la ordenación de listas.

**6. Referencias**

* Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to Algorithms (4th ed.). MIT Press.

<https://dl.ebooksworld.ir/books/Introduction.to.Algorithms.4th.Leiserson.Stein.Rivest.Cormen.MIT.Press.9780262046305.EBooksWorld.ir.pdf>

* Drozdek, Adam. "Data Structures and Algorithms in C++." 5th Edition, Cengage Learning, 2019.

<https://itlectures.ro/wpcontent/uploads/2016/04/AdamDrozdek__DataStructures_and_Algorithms_in_C_4Ed.pdf>