B U C K E T - S O R T

0250009 - Luis Cedillo Maldonado

Documentación de Algoritmos de Ordenación

1. Introducción

Los algoritmos de ordenación son fundamentales en la ciencia de la computación y se utilizan en una amplia variedad de aplicaciones. Esta documentación cubre dos algoritmos de ordenación: Bucket Sort y Quick Sort.

2. Lista Doblemente Enlazada (`List`)

Una lista doblemente enlazada es una estructura de datos que consta de un conjunto de nodos conectados secuencialmente. Cada nodo contiene un valor y dos punteros, uno que apunta al siguiente nodo y otro que apunta al nodo anterior.

Código resumido de la Lista Doblemente Enlazada:

```
#pragma once

#include <iostream>
class List
{
public:
    List() : head(nullptr), tail(nullptr), size(0) {}
    ~List();
    void insert(const T &val);
    void insert(const T &val, int index);
    void set_at(const T &val, int index);
    T getAt(int index) const;
    T remove_at(int index);
    int size_of_list() const { return size; }

private:
    class Node
    {
    public:
        Node(const T &val) : value(val), next(nullptr), prev(nullptr) {}
```

```
        T get_value() const { return value; }
        void set_value(const T &val) { value = val; }
        Node *get_next() const { return next; }
        void set_next(Node *n) { next = n; }
        Node *get_prev() const { return prev; }
        void set_prev(Node *p) { prev = p; }

    private:
        T value;
        Node *next;
        Node *prev;
    };

    Node *head;
    Node *tail;
    int size;
};
```

Explicación de las funciones de la Lista Doblemente Enlazada:

- `List()`: Constructor por defecto. Inicializa una lista vacía.
- `~List()`: Destructor. Libera la memoria ocupada por los nodos de la lista.
- `insert(const T &val)`: Inserta un nuevo nodo con el valor `val` al final de la lista.
- `insert(const T &val, int index)`: Inserta un nuevo nodo con el valor `val` en la posición especificada por `index`.
- `set_at(const T &val, int index)`: Establece el valor del nodo en la posición `index` al valor `val`.
- `getAt(int index) const`: Devuelve el valor del nodo en la posición `index`.
- `remove_at(int index)`: Elimina el nodo en la posición `index` y devuelve su valor.
- `size_of_list() const`: Devuelve el número de nodos en la lista.

3. Bucket Sort

Bucket Sort es un algoritmo de ordenación que distribuye los elementos de un arreglo en varios "buckets" o cubos. Luego, cada cubo se ordena individualmente, ya sea usando un algoritmo de ordenación diferente o de forma recursiva aplicando el mismo algoritmo de Bucket Sort.

Código de Bucket Sort:

```
#include "list.h"
#include "quick_sort.h"
#include <vector>
#include <cmath>

template <typename T>
class BucketSort
{
public:
    static void sort(List<T> &list);
private:
    struct MinMaxValues;
    static MinMaxValues findMinMax(const List<T> &list);
    static void distributeToBuckets(const List<T> &list, std::vector<List<T>> &buckets, T
minVal, T maxVal);
    static void sortBuckets(std::vector<List<T>> &buckets);
    static void concatenateBuckets(List<T> &list, const std::vector<List<T>> &buckets);
};
```

Explicación de las funciones de Bucket Sort:

- `sort(List<T> &list)`: Es la función principal que se encarga de ordenar la lista utilizando el algoritmo de Bucket Sort.
- `findMinMax(const List<T> &list)`: Esta función busca y devuelve los valores mínimo y máximo en la lista. Es útil para determinar el rango de valores que se distribuirán en los cubos.
- `distributeToBuckets(const List<T> &list, std::vector<List<T>> &buckets, T minVal, T maxVal)`: Distribuye los elementos de la lista en los cubos según su valor. Los cubos se crean en función del rango de valores (minVal y maxVal).

- `sortBuckets(std::vector<List<T>> &buckets)`: Ordena individualmente cada cubo. En este caso, se utiliza el algoritmo Quick Sort para ordenar los cubos.
- `concatenateBuckets(List<T> &list, const std::vector<List<T>> &buckets)`: Combina los cubos ordenados de nuevo en la lista original, dando como resultado la lista completamente ordenada.

4. Quick Sort

Quick Sort es un algoritmo de ordenación por comparación. Funciona seleccionando un "elemento pivote" del arreglo y particionando los otros elementos en dos subarreglos, según si son menores o mayores que el pivote.

Código de Quick Sort:

```
#include "list.h"
template <typename T>
class QuickSort
{
public:
    static void sort(List<T> &list);
private:
    static void sort(List<T> &list, int low, int high);
    static int partition(List<T> &list, int low, int high);
    static void swap(List<T> &list, int i, int j);
};
```

Explicación de las funciones de Quick Sort:

- `sort(List<T> &list)`: Es la función principal que ordena la lista utilizando el algoritmo Quick Sort.
- `sort(List<T> &list, int low, int high)`: Es una función recursiva que ordena una sublista de la lista original. La sublista está definida por los índices `low` y `high`.
- `partition(List<T> &list, int low, int high)`: Esta función selecciona un elemento pivote y particiona la sublista en dos: los elementos menores que el pivote y los elementos mayores que

el pivote. Devuelve el índice del pivote después de la partición.

- `swap(List<T> &list, int i, int j)`: Es una función auxiliar que intercambia dos elementos de la lista, ubicados en los índices `i` y `j`.

5. Conclusión

Tanto Bucket Sort como Quick Sort son algoritmos de ordenación eficientes y versátiles. La elección entre uno u otro dependerá de las características específicas del conjunto de datos a ordenar y de los requisitos de la aplicación.
