Análisis y Algoritmos

Roberto Charreton Kaplun
Universidad de Artes Digitales

Guadalajara, Jalisco

Email: idv17c.rcharreton@uartesdigitales.edu.mx

Profesor: Efraín Padilla

Mayo 30, 2019

1) Counting Sort

El ordenamiento por conteo es una técnica de clasificación basada en claves entre un rango específico. Funciona contando el número de objetos que tienen valores clave distintos.

Desarrolle el codigo cambiando el contador en el indice [i] para que el contador[i] ahora contenga la posición real de este carácter en la matriz de salida, despues copie el arreglo de salida al vector actual obteniendo los datos ordenados, posteriormente analice el vector con los multiples casos Ascendente, Descente o Random, de estos dos obtuve el mejor y el peor caso en su benchmarking.

Complejidad: O(n+k)

Codigo

```
vector<int> CManager::CountingSort(vector<int>& Vector)
{
        int max = *max_element(Vector.begin(), Vector.end());
        int min = *min_element(Vector.begin(), Vector.end());
        int range = max - min + 1;
        vector<int> count(range), output(Vector.size());

        // Store count of each character
        for (int i = 0; i < Vector.size(); i++)
                count[Vector[i] - min]++;
        // Change count[i] so that count[i] now contains
        //actual position of this character in output array
        for (int i = 1; i < count.size(); i++)
                count[i] += count[i - 1];
        // Gen the output array
        for (int i = Vector.size() - 1; i >= 0; i--)
        {
                output[count[Vector[i] - min] - 1] = Vector[i];
                count[Vector[i] - min]--;
        }
        // Copy the output array to vector, so that Vector now
        // contains sorted characters
        for (int i = 0; i < Vector.size(); i++)
                Vector[i] = output[i];

        return Vector;


}
```

2) Bucket Sort
Bucket Sort distribuye todos los elementos a ordenar entre un número finito de casilleros. Cada casillero sólo puede contener los elementos que cumplan unas determinadas condiciones..

Complejidad: Omega(n Log n)

Codigo

```
vector<int> CManager::BucketSort(vector<int> &Vector)
{
        int i, j;
        int count[20];
        int n = Vector.size();
        for (i = 0; i < n; i++)
                count[i] = 0;

        for (i = 0; i < n; i++)
                (count[Vector[i]])++;

        for (i = 0, j = 0; i < n; i++)
                for (; count[i] > 0; (count[i])--)
                        Vector[j++] = i;

        m_bucket = Vector;
        return m_bucket;

}
```

3) Radix Sort
Radix Sort es un algoritmo de ordenamiento que ordena enteros procesando sus dígitos de forma individual. Como los enteros pueden representar cadenas de caracteres y, especialmente, números en punto flotante especialmente formateados, radix sort no está limitado sólo a los enteros.

Desarrolle el codigo buscando digito por digito empezando por el ultimo valor en el indice hasata el primero, posteriormente hize la funcion recorsiva con el vector con los multiples casos Ascendente, Descente o Random, de estos dos obtuve el mejor y el peor caso en su benchmarking.

Complejidad:O(n+k).
Codigo

```
vector<int> CManager::radixSort(vector<int>& Vector)
{
        m_vec.resize(10);
        int temp, m = 0;

        for (int i = 0; i < 7; i++)
        {
                for (int j = 0; j < Vector.size(); j++)
                {
                        temp = (int)(Vector[j] / pow(10, i)) % 10;
                        m_vec[temp].push_back(Vector[j]);
                }
                for (int k = 0; k < 10; k++)
                {
                        for (int l = 0; l < m_vec[k].size(); l++)
                        {
                                Vector[m] = m_vec[k][l];
                                m++;
```

```
                        }
                        m_vec[k].clear();
                }

                m = 0;
        }


        return Vector;
}
```