Análisis y Algoritmos

Sergio Juan Diaz Carmona
Universidad de Artes Digitales

Guadalajara, Jalisco

Email: idv17c.sdiaz@uartesdigitales.edu.mx

Profesor: Efraín Padilla

Mayo 5, 2019

1) Algoritmos de Ordenamiento

En computación y matemáticas un algoritmo de ordenamiento es un algoritmo que pone elementos de una lista o un vector en una secuencia dada por una relación de orden, es decir, el resultado de salida ha de ser una permutación —o reordenamiento— de la entrada que satisfaga la relación de orden dada.

BUCKET-SORT

El ordenamiento por casilleros (bucket sort o bin sort, en inglés) es un algoritmo de ordenamiento que distribuye todos los elementos a ordenar entre un número finito de casilleros. Cada casillero sólo puede contener los elementos que cumplan unas determinadas condiciones. En el ejemplo esas condiciones son intervalos de números. Las condiciones deben ser excluyentes entre sí, para evitar que un elemento pueda ser clasificado en dos casilleros distintos. Después cada uno de esos casilleros se ordena individualmente con otro algoritmo de ordenación (que podría ser distinto según el casillero), o se aplica recursivamente este algoritmo para obtener casilleros con menos elementos.
Cuando los elementos a ordenar están uniformemente distribuidos la complejidad computacional de este algoritmo es de O(n).
el codigo es el siguiente

```
std::vector<int> BucketSort(std::vector<int> &vec)
{
        int i, j;
        int* count = new int(vec.size());
        for (i = 0; i < vec.size(); i++)
                count[i] = 0;

        for (i = 0; i < vec.size(); i++)
                (count[vec[i]])++;

        for (i = 0, j = 0; i < vec.size(); i++)
                for (; count[i] > 0; (count[i])--)
                        vec[j++] = i;

        return vec;
}
```

COUNTIG-SORT

El ordenamiento por cuentas (counting sort en inglés) es un algoritmo de ordenamiento en el que se cuenta el número de elementos de cada clase para luego ordenarlos. Sólo puede ser utilizado por tanto para ordenar elementos que sean contables

El primer paso consiste en averiguar cuál es el intervalo dentro del que están los datos a ordenar (valores mínimo y máximo). Después se crea un vector de números enteros con tantos elementos como valores haya en el intervalo [mínimo, máximo], y a cada elemento se le da el valor 0 (0 apariciones). Tras esto se recorren todos los elementos a ordenar y se cuenta el número de apariciones de cada elemento (usando el vector que hemos creado). Por último, basta con recorrer este vector para tener todos los elementos ordenados.

la complejidad de este ordenamineto es O (n + k) donde n es el número de elementos en la matriz de entrada y k es el rango de entrada.
el codigo es el siguiente

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

        for (int i = 0; i < Vec.size(); i++)
                counting[Vec[i] - min]++;
        for (int i = 1; i < counting.size(); i++)
                counting[i] += counting[i - 1];

        for (int i = Vec.size() - 1; i >= 0; i--)
        {
                output[counting[Vec[i] - min] - 1] = Vec[i];
                counting[Vec[i] - min]--;
        }
        for (int i = 0; i < Vec.size(); i++)
                Vec[i] = output[i];

        return Vec;
}
```

RADIX-SEARCH
El ordenamiento Radix (radix sort en inglés) es un algoritmo de ordenamiento que ordena enteros procesando sus dígitos de forma individual. Como los enteros pueden representar cadenas de caracteres (por ejemplo, nombres o fechas) y, especialmente, números en punto flotante especialmente formateados, radix sort no está limitado sólo a los enteros. Se clasifica en digito menos significativo (LSD) digito significativo (MSD)
La complejidad temporal del algoritmo es el siguiente: Supongamos que los números de entrada n tiene dígitos máximo k. A continuación, el procedimiento se llama Ordenar Contando con un total de k veces. Contando Sort es un algoritmo lineal o O (n). Así que todo el procedimiento de Radix sort toma tiempo O (kn). Si los números son de tamaño finito, el algoritmo se ejecuta en O (n) tiempo asintótica.

```
std::vector<int> radixSort(std::vector<int>& vec)
{
        vector<vector<int>> vector;
        vector.resize(10);
        int temp, m = 0;
        int *n = new int[vec.size()];

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

```
                    vector[k].clear();
                }
                m = 0;
            }
            return vec;
}
```