30-5-2020

Néstor Monzón González - 735418

Andrés Otero García - 757755

PRÁCTICA 1

Radix Sort

Algoritmia Básica

1 – Implementación

En primer lugar, se siguió una implementación iterativa del algoritmo. Tras considerarlo, se decidió reformarlo en una función recursiva, ya que se reducía el número de copias del vector auxiliar al original.

El código principal de la función es el siguiente:

// v el vector de numeros (formato string), n\_dig el numero de digitos

// ordena v con el algoritmo de radix LSF

void radix\_rec(vector<string>& v, const int n\_dig, const int i\_dig) {

vector<int> cuenta = CUENTA\_INICIAL; // lleva la cuenta del numero de veces que aparece cada digito

int n\_eltos = v.size();

vector<string> aux(n\_eltos); // auxiliar del mismo tamaño que el original

for (auto e : v) {

int digito = e[i\_dig] - '0'; // i-ésimo digito

//cout << "digito: " << digito << endl;

cuenta[digito]++; // Contamos el digito correspondiente

}

// Acumulamos las cuentas de los digitos de izquierda a dcha:

acumular(cuenta);

// Y recolocamos cada elemento del original en el auxiliar según la cuenta:

for (int j = n\_eltos - 1; j >= 0; j--) { // (empezando por la dcha)

string elemento = v[j]; // Tomamos el j-esimo elemento

int digito = elemento[i\_dig] - '0'; // Y su i-esimo digito

int indice = cuenta[digito]--; // Posicion en el auxiliar (y reducimos esa cuenta)

aux[indice] = elemento;

}

// Finalmente, actualizamos el vector original:

If (i\_dig > 0) {

radix\_rec(aux, n\_dig, i\_dig-1);

}

}

Este código asume que todos los elementos tienen longitud n\_dig, por lo que requiere que el vector v se inicialice con elementos de esta longitud (los números con menos dígitos deberán tener ceros a la izquierda).

El coste temporal de esta función, a grandes rasgos será O(n\_dig(n\_eltos+base)), donde n\_dig es el número de dígitos, n\_eltos el número de números y base la base, en este caso 10. Cuando n\_eltos sea muy grande en comparación a la base (n\_eltos>>base), la base será despreciable y el orden será O(n\_dig\*n\_eltos).

Este coste se debe a que el primer bucle tiene n\_eltos iteraciones; acumular(cuenta), una función simple que suma en cada elemento de <cuenta> la cuenta de los elementos anteriores, es decir, unas 10 iteraciones, un coste despreciable; y el bucle final, de n\_eltos iteraciones de nuevo.

Así, en total, esto supone n\_eltos+10+n\_eltos=O(n\_eltos) por cada llamada no recursiva a esta función (hasta el if final). Añadiendo la recursión, esto se ejecuta un total de n\_dig veces, llevando al coste final de O(n\_dig\*n\_eltos).

2 – Pruebas

Se ha desarrollado el programa principal de forma que acepte varios tipos de invocaciones. Si se invoca sin parámetros, permitirá elegir entre el bucle de prueba predefinido y el modo interactivo, que permite un bucle infinito en el que se introducen los parámetros n\_digitos y n\_eltos y muestra los tiempos de la ordenación con nuestra función y con la función sort estándar de c++.

Además, permite elegir si se muestran los vectores antes y después de su ordenación, para así comprobar el correcto funcionamiento del algoritmo.

Por otra parte, invocándolo con dos argumentos, el primero corresponderá al número de dígitos, y el segundo al número de elementos. Se puede añadir un tercero (S/N) que determine si se mostrarán los vectores, como en el modo interactivo. Con este modo, se facilita la invocación desde scripts externos.

Para realizar las mediciones de los tiempos con precisión, se ha utilizado la biblioteca chrono de c++.

Ordenando con 1 digitos y 10 elementos

Tiempo Radix: 3.5e-06

Tiempo Sort: 3.2e-06

radix es mejor: 0

log(n\_eltos)=2.30259

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 1 digitos y 100 elementos

Tiempo Radix: 1.63e-05

Tiempo Sort: 1.89e-05

radix es mejor: 1

log(n\_eltos)=4.60517

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 1 digitos y 1000 elementos

Tiempo Radix: 9.56e-05

Tiempo Sort: 0.0002907

radix es mejor: 1

log(n\_eltos)=6.90776

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 1 digitos y 10000 elementos

Tiempo Radix: 0.0008939

Tiempo Sort: 0.0034265

radix es mejor: 1

log(n\_eltos)=9.21034

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 1 digitos y 100000 elementos

Tiempo Radix: 0.0102882

Tiempo Sort: 0.0428377

radix es mejor: 1

log(n\_eltos)=11.5129

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 1 digitos y 1000000 elementos

Tiempo Radix: 0.0840578

Tiempo Sort: 0.521597

radix es mejor: 1

log(n\_eltos)=13.8155

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 2 digitos y 10 elementos

Tiempo Radix: 5e-06

Tiempo Sort: 2.8e-06

radix es mejor: 0

log(n\_eltos)=2.30259

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 2 digitos y 100 elementos

Tiempo Radix: 3.63e-05

Tiempo Sort: 3.12e-05

radix es mejor: 0

log(n\_eltos)=4.60517

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 2 digitos y 1000 elementos

Tiempo Radix: 0.0004073

Tiempo Sort: 0.0003081

radix es mejor: 0

log(n\_eltos)=6.90776

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 2 digitos y 10000 elementos

Tiempo Radix: 0.0024382

Tiempo Sort: 0.0041693

radix es mejor: 1

log(n\_eltos)=9.21034

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 2 digitos y 100000 elementos

Tiempo Radix: 0.0181351

Tiempo Sort: 0.0430092

radix es mejor: 1

log(n\_eltos)=11.5129

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 2 digitos y 1000000 elementos

Tiempo Radix: 0.190851

Tiempo Sort: 0.54008

radix es mejor: 1

log(n\_eltos)=13.8155

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 3 digitos y 10 elementos

Tiempo Radix: 6.3e-06

Tiempo Sort: 2.9e-06

radix es mejor: 0

log(n\_eltos)=2.30259

log(n\_eltos) > n\_digs = 0

----------------------------------------------------------------

Ordenando con 3 digitos y 100 elementos

Tiempo Radix: 2.14e-05

Tiempo Sort: 2.19e-05

radix es mejor: 1

log(n\_eltos)=4.60517

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 3 digitos y 1000 elementos

Tiempo Radix: 0.0001811

Tiempo Sort: 0.0002783

radix es mejor: 1

log(n\_eltos)=6.90776

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 3 digitos y 10000 elementos

Tiempo Radix: 0.0020819

Tiempo Sort: 0.0036482

radix es mejor: 1

log(n\_eltos)=9.21034

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 3 digitos y 100000 elementos

Tiempo Radix: 0.0181945

Tiempo Sort: 0.043474

radix es mejor: 1

log(n\_eltos)=11.5129

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 3 digitos y 1000000 elementos

Tiempo Radix: 0.216371

Tiempo Sort: 0.511602

radix es mejor: 1

log(n\_eltos)=13.8155

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 4 digitos y 10 elementos

Tiempo Radix: 6.9e-06

Tiempo Sort: 3e-06

radix es mejor: 0

log(n\_eltos)=2.30259

log(n\_eltos) > n\_digs = 0

----------------------------------------------------------------

Ordenando con 4 digitos y 100 elementos

Tiempo Radix: 2.66e-05

Tiempo Sort: 2.21e-05

radix es mejor: 0

log(n\_eltos)=4.60517

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 4 digitos y 1000 elementos

Tiempo Radix: 0.0002507

Tiempo Sort: 0.0002866

radix es mejor: 1

log(n\_eltos)=6.90776

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 4 digitos y 10000 elementos

Tiempo Radix: 0.003821

Tiempo Sort: 0.0038715

radix es mejor: 1

log(n\_eltos)=9.21034

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 4 digitos y 100000 elementos

Tiempo Radix: 0.0262888

Tiempo Sort: 0.0547108

radix es mejor: 1

log(n\_eltos)=11.5129

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 4 digitos y 1000000 elementos

Tiempo Radix: 0.275736

Tiempo Sort: 0.528453

radix es mejor: 1

log(n\_eltos)=13.8155

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

El radix era mejor cuando el logaritmo del numero de elementos era

mayor que el numero de digitos en 17 casos de 24 (0.708333)

root@LAPTOP-HMNPRTML:/mnt/d/Universidad/Algoritmia Basica/Algoritmia-Basica/Practica1/Practica1# ./Practica1

Seleccione el modo:

1 -> interactivo

2 -> pruebas prefabricadas

2

Ordenando con 1 digitos y 10 elementos

Tiempo Radix: 5.9e-06

Tiempo Sort: 6.2e-06

radix es mejor: 1

log(n\_eltos)=2.30259

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 1 digitos y 100 elementos

Tiempo Radix: 2.82e-05

Tiempo Sort: 3.24e-05

radix es mejor: 1

log(n\_eltos)=4.60517

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 1 digitos y 1000 elementos

Tiempo Radix: 7.89e-05

Tiempo Sort: 0.000307

radix es mejor: 1

log(n\_eltos)=6.90776

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 1 digitos y 10000 elementos

Tiempo Radix: 0.00167

Tiempo Sort: 0.0037616

radix es mejor: 1

log(n\_eltos)=9.21034

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 1 digitos y 100000 elementos

Tiempo Radix: 0.0082097

Tiempo Sort: 0.043875

radix es mejor: 1

log(n\_eltos)=11.5129

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 1 digitos y 1000000 elementos

Tiempo Radix: 0.0874717

Tiempo Sort: 0.521012

radix es mejor: 1

log(n\_eltos)=13.8155

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 2 digitos y 10 elementos

Tiempo Radix: 5.4e-06

Tiempo Sort: 2.9e-06

radix es mejor: 0

log(n\_eltos)=2.30259

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 2 digitos y 100 elementos

Tiempo Radix: 1.46e-05

Tiempo Sort: 2.12e-05

radix es mejor: 1

log(n\_eltos)=4.60517

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 2 digitos y 1000 elementos

Tiempo Radix: 0.0002063

Tiempo Sort: 0.0003041

radix es mejor: 1

log(n\_eltos)=6.90776

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 2 digitos y 10000 elementos

Tiempo Radix: 0.0018806

Tiempo Sort: 0.0034854

radix es mejor: 1

log(n\_eltos)=9.21034

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 2 digitos y 100000 elementos

Tiempo Radix: 0.0174737

Tiempo Sort: 0.0429969

radix es mejor: 1

log(n\_eltos)=11.5129

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 2 digitos y 1000000 elementos

Tiempo Radix: 0.180742

Tiempo Sort: 0.503792

radix es mejor: 1

log(n\_eltos)=13.8155

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 3 digitos y 10 elementos

Tiempo Radix: 5.5e-06

Tiempo Sort: 2.8e-06

radix es mejor: 0

log(n\_eltos)=2.30259

log(n\_eltos) > n\_digs = 0

----------------------------------------------------------------

Ordenando con 3 digitos y 100 elementos

Tiempo Radix: 2.06e-05

Tiempo Sort: 2.15e-05

radix es mejor: 1

log(n\_eltos)=4.60517

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 3 digitos y 1000 elementos

Tiempo Radix: 0.0001692

Tiempo Sort: 0.0002783

radix es mejor: 1

log(n\_eltos)=6.90776

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 3 digitos y 10000 elementos

Tiempo Radix: 0.0017938

Tiempo Sort: 0.0036896

radix es mejor: 1

log(n\_eltos)=9.21034

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 3 digitos y 100000 elementos

Tiempo Radix: 0.0180585

Tiempo Sort: 0.0436941

radix es mejor: 1

log(n\_eltos)=11.5129

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 3 digitos y 1000000 elementos

Tiempo Radix: 0.207747

Tiempo Sort: 0.507505

radix es mejor: 1

log(n\_eltos)=13.8155

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 4 digitos y 10 elementos

Tiempo Radix: 6.8e-06

Tiempo Sort: 3e-06

radix es mejor: 0

log(n\_eltos)=2.30259

log(n\_eltos) > n\_digs = 0

----------------------------------------------------------------

Ordenando con 4 digitos y 100 elementos

Tiempo Radix: 2.62e-05

Tiempo Sort: 2.19e-05

radix es mejor: 0

log(n\_eltos)=4.60517

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 4 digitos y 1000 elementos

Tiempo Radix: 0.0002225

Tiempo Sort: 0.0002835

radix es mejor: 1

log(n\_eltos)=6.90776

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 4 digitos y 10000 elementos

Tiempo Radix: 0.0031568

Tiempo Sort: 0.0038234

radix es mejor: 1

log(n\_eltos)=9.21034

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 4 digitos y 100000 elementos

Tiempo Radix: 0.0245497

Tiempo Sort: 0.0478288

radix es mejor: 1

log(n\_eltos)=11.5129

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

Ordenando con 4 digitos y 1000000 elementos

Tiempo Radix: 0.269179

Tiempo Sort: 0.513946

radix es mejor: 1

log(n\_eltos)=13.8155

log(n\_eltos) > n\_digs = 1

----------------------------------------------------------------

El radix era mejor cuando el logaritmo del numero de elementos era

mayor que el numero de digitos en 20 casos de 24 (0.833333)