El poeta

Viquez Alpizar Camila y Mata Mora Leonardo

***Abstract*—**

I. INTRODUCCION

El poeta es un programa creado para la autogeneracion de texto. El programa fue creado en el entorno de desarrollo de visual studio, utilizando el leguaje c#. El programa hace uso de algoritmos geneticos y funciones de distancia de similitud entre los textos generados y el texto meta. Las distancias implemnetadas corresponden a Manhatthan, Chebyshev y Campana. Este programa es Capz de recibir cualquier poema y generar poemas que se asemejan en algunas palabras a la meta. El principal objetivo del proyecto es el estudio y anlisis detallado de cada uno de estos algoritmos, esto con el fin de conocer su orden. Esto se lograr mediante notacion asintotica, especificamente el calculo de O grande. Los filtros estudiados se dividen en dos grupos, escala de grises y convoluviones.

II. ANALISIS DE LAS FUNCIONES

A. *Diccionarios y n-grams*

Al inicar el programa el usuario debe realizar la carga inicial de datos la cual consiste. Esto se refiere a que el programa debe crear su base de datos de la cual tomara las palabras para ir poco a poco, mediante cruces y mutaciones, acercandose mas a su meta.

El programa genera los n-grams con un n que va de dos hasta cinco. Todos los n-grams son escritos en una lista llamada datosBase.

Esta basedatos me permite generar mi polacion inicial, y la cantidad de individios que habitan ella, esta cantidad de individuos es dicha por el usuario.

Los n-grams tambien son generados para el poema meta. Esto par la implementacion de las mutaciones mas adelante. Esta funcion recibe como parametros un numero que indica el n para los n-grams y texto contiene el texto leido de la base de datos.

El barometro de esta funcion esta dado por el largo del texto que entra como parametro.Finalmente retorna una lista que contiene todos los n-gramas Esta funcion implementa un ciclo simple por lo cual aporta un n al orden de la funcion por lo tanto en el peor de los casos el orden esta dado por O(n).

Los diccionarios se calculan para cada individuo. Cada diccionario tiene como llave las palabras que se forman de la union de las palabras creadas en el individuo y las palabras del poema meta. Esto se puede ver como la union de dos conjuntos, por lo tato implica la unicidad de las llaves.

Tanto para el poema meta como para el poema individuo se crean un hisitograma, mediante la funcion crearHistograma, y la llave usada para estos diccionarios es cada palabra de la lista union,explicada anteriormente. Estos histogramas indican cuantas veces aparece la palabra de la union en cada uno de ellos.

La funcion recibe como parametro la lista union, que tiene las palabras compartidas entre el poema generado y el poema meta, y texto que corresponde a la base de datos que contiene todos los n-grams.

La funcion crear histograma tiene como barometro el largo de la lista union que recibe como parametro. Esta funcion realiza dos ciclos simples dentro de ella. Uno para recorrer la lista union y crear las llaves del diccionario, este ciclo aporta un n al orden de la funcion. El otro ciclo es para

```
4 referencias
private List<string> GenerarNgrams(int num, List<string> texto)
{
    List<string> ngramList = new List<string>();
    for (int k = 0; k < (texto.Count - num + 1); k++)
    {
        String s = "";
        int inicio = k;
        int fin = k + num;
        for (int j = inicio; j < fin; j++)
        {
            s = s + " " + texto[j];

        }
        basedatos.Add(s);

    }
    if (num == 2)...
    else if (num == 3)...
    else if (num == 4)...
    else if (num == 5)...
    return basedatos;
}
```

Fig. 1. Funcion Generar n-grams

```
if (num == 2)
{
    dos = basedatos.Count() + ngramList.Count() - 1;
}
else if (num == 3)
{
    tres = basedatos.Count() + ngramList.Count() - 1;
}
else if (num == 4)
{
    cuatro = basedatos.Count() + ngramList.Count() - 1;
}
else if (num == 5)...
return basedatos;
```

Fig. 2. Funcion Generar n-grams creacion base de datos

separar las palabras que provienen de la base de datos ya que son n.gramas entinces estan unidos por espacios. Una vez realizado esto se procede a ver si la palabra clave existe en el histrograma, de ser asi el valor de la llave, que funciona como contador aumenta, de lo contrario se matiene en 0. Este ciclo aporte un valor de n al algoritmo, por lo tanto en el peor de los cases el orden de este algoritmo es O(n^2).

```
private Dictionary<string, int> CrearHistograma(List<string> union, List<string> texto)
{
    Dictionary<string, int> histograma = new Dictionary<string, int>();
    foreach (string palabra in union)
    {
        histograma.Add(palabra,0);
    }
    List<string>separada = separar(texto);
    foreach (string g in separada)
    {
        if (histograma.ContainsKey(g))
        {
            histograma[g] = histograma[g] + 1;
        }

    }
    return histograma;
}
```

Fig. 3. Funcion Crear Histogramas

La Funsion conjunto palabra es la encargada de crear la lista union. esta funcion realiza un ciclo simple ,en el cual recorre los datos meta, que coresponden a las palabras del poema meta y la poblacion que es el poema individuo que se genero y que esta siendo analizado, esto aporta un valor de n a la funcion.El barometro esta dado por la cantidad de palabas que tiene el poema individuo que recibe como parametro. Note que dentro de esta funcion se llama a la funcion separar que es la encargada de gacer un split de las palabras de los n-grams, esta funcion aporta un valor n a la funcion conjuntoPlabras.En el peor de los casos el orden de esta funcoin es O(n^2).

```
private List<string> cruzarIndividuos(List<string> poema1, List<string> poema2)

{
    List<string> individuoCruces = new List<string>();
    string nuevoValor = " ";
    int cantidadActual = cantidadPalabras(individuoCruces);
    while(cantidadActual < cantidadPalabrasMeta-1)
    {
        int num1 = rnd.Next(0, poema1.Count());
        int num2 = rnd.Next(0, poema2.Count());
        string valor1 = poema1[num1];
        string valor2 = poema2[num2];
        nuevoValor = valor1 +" "+valor2;
        individuoCruces.Add(nuevoValor);
        cantidadActual = cantidadPalabras(individuoCruces);
    }
    string faltante = mutar();
    individuoCruces.Add(faltante);
    return individuoCruces;
}
```

Fig. 4. Funcion Conjunto Palabras, union

B. Calculo de similitud entre documentos

Para esta seccion los histrogramas tuvieron un papel my importante. Ya que para el calculo de las distancias se debe realizar entre vectores tomando los valor p y q respectivamente de cada uno para realizar los calculo.

```
private List<string> separar(List<string> poblacion)
{

    List<string> text = new List<string>();
    string[] a;
    for (int i = 0; i < poblacion.Count(); i++)
    {

        a = poblacion[i].Split(new[] { " " }, StringSplitOptions.None);
        int largo = a.Count();
        foreach (string palabra in a)
        {
            if(palabra != "")
            {
                text.Add(palabra);
            }

        }
    }
    return text;
}
```

Fig. 5. Funcion Separar

1) Distancia Manhatthan: La distancia manjathan esta definida por

$$\sum_{i=0}^{n} i = |p_i - q_i|$$

La funcion para calcular la distancia manhattahn recibe por parametro los histogframas del poema individuo y del poema meta. Los valores p se obtinen del histogramaIndividuo y los valores para q se obtienen del histograma meta.
Esta funcion tiene como barometro el largo del histograma individuo, ambos histogramas son del mismo tamano.Dentro de ella se lleva acabo un ciclo cimple por lo que el orden de esta funcion en el peor de los casos es linea. El acceso a los diccionarios es lineal, por esta razon es beneficiosa la implementacion de los mismo. El orden de esta funcion corresponde a O(n^2).

```
private int distanciaManhatthan(Dictionary<string, int> histogramaIndividuo, Dictionary<string, int> histogramaMeta)
{
    int fitness = 30;
    int p = 0;
    int q = 0;
    int cont = 0;
    int sumatoria = 0;
    int contDist = 0;
    while (cont < histogramaIndividuo.Count)
    {
        p = histogramaIndividuo[cont];
        q = histogramaMeta[cont];
        int resul = Math.Abs(p - q);
        sumatoria += resul;
        cont++;
        string sumaS = Convert.ToString(sumatoria);
        List<string> escribir = new List<string>();
        if (sumatoria < fitness)
        {
            escribir.Add(sumaS);
            System.IO.File.WriteAllLines(@"C:\Users\camil\OneDrive\Documents\TEC\Análisis\ElPoeta\ditanciaM" + contDist + ".txt", escribir);
            contDist++;

        }

    }
    contDist++;
    return sumatoria;
}
```

Fig. 6. Funcion distancia Manhatthan

2) Distancia Chebyshev: La distancia Chebyshev esta definida por $(max|p_i - q_i|)$ La funcion para calcular la distancia chebyshev recibe por parametro los histogframas del poema individuo y del poema meta. Los valores p se obtinen del histogramaIndividuo y los valores para q se obtienen del histograma meta.
Esta funcion tiene como barometro el largo del histograma individuo, ambos histogramas son del mismo tamano.Dentro de ella se lleva acabo un ciclo cimple por lo que el orden de esta funcion en el peor de los casos es linea. El acceso

a los diccionarios es lineal, por esta razon es beneficiosa la implementacion de los mismo.La distancia tiene un fitness de 30, las que esten por debajo de este son agregadas a la lista de distancias. El orden de esta funcion corresponde a O(n^2).

```
private int Chebyshev(Dictionary<string, int> histogramaIndividuo, Dictionary<string, int> histogramaMeta)
{
    int p = 0;
    int q = 0;
    int cont = 0;
    int result = 0;
    int resta = 0;
    int fitness = 30;
    List<int> restas = new List<int>();
    int contDist = 0;
    while (cont < histogramaIndividuo.Count())
    {
        p = histogramaIndividuo[cont];
        q = histogramaMeta[cont];
        resta = p - q;
        if (resta < fitness)
        {
            restas.Add(resta);
        }
        string restaS = Convert.ToString(resta);
        List<string> escribir = new List<string>();
        escribir.Add(restaS);
        System.IO.File.WriteAllLines(@"C:\Users\camil\OneDrive\Documents\TEC\Análisis\ElPoeta\ditanciaC" + contDist + ".txt", escribir);
        contDist++;
        cont++;
    }
    result = restas.Max();
    return result;
}
```

Fig. 7. Funcion distancia Chebyshev

3) Distancia Campana: Esta funcion toma el primer valor del histrograma meta y lo selecciona como media, se tiene un varianza de 30, esta va a funcionar como fitness. Se crean dos valores un minimo, que corresponde el valor del histograma menos la varianza y un maximo que corresponde al valor del histograma mas la varianza. Con este rango se recorren los valores del hisograma individuo, si los valores resectivos a las llaves son validos en el rango se agregan a la lista de distancias prometedoras, de lo contrario no se agregan. Esto se ejecuta con el siguente valor del histograma meta.
El barometro de esta funcion esta definido por el largo del histograma indivisuo que asigna la condicion de parada del ciclo mas anidado, dentro de la funcion se ejecutan dos ciclos simples, uno parar recorrer el histograma meta y otro para recorrer el histograma individuo, cada uno aporta un valor de n.El orden del algoritmo corresponde a O(n^2).

```
private int campana(Dictionary<string, int> histogramaIndividuo, Dictionary<string, int> histogramaMeta)
{
    int varianza = 30;
    int p = 0;
    int q = 0;
    int cont = 0;
    int contDist = 0;
    int maxRango = 0;
    int minRango = 0;
    int distPrometedora = 0;
    while (cont < histogramaMeta.Count)
    {

        p = histogramaIndividuo[cont];
        q = histogramaMeta[cont];
        maxRango = q + varianza;
        minRango = q + varianza;
        foreach (KeyValuePair<string, int> result in histogramaIndividuo)
        {
            if (minRango < histogramaIndividuo[result.value] && histogramaIndividuo[result.value] < maxRango)
            {
                distPrometedora += histogramaIndividuo[result.value];

            }
        }
        cont++;
```

Fig. 8. Funcion distancia Campana

```
        string sumaS = Convert.ToString(distPrometedora);
        List<string> escribir = new List<string>();
        escribir.Add(sumaS);
        System.IO.File.WriteAllLines(@"C:\Users\camil\OneDrive\Documents\TEC\Análisis\ElPoeta\ditanciaCampana" + contDist + ".txt", escribir);
        contDist++;
    }
    contDist++;
    return distPrometedora;
}
```

Fig. 9. Funcion distancia Campana, continuacion

C. Algoritmos Genticos

1) Cruces: La cantidad de individuaos de una poblacion es elegida por el usuario. Esta cantidad es la que indica cuantos oindividuos se vana generar por poblacon, por lo tanto para realizar los cruces se deben generar de tal forma que siempre se mantenga esta cantidad de individuos para la siguente poblacion.
La funcion generar individuos validos, ento quiere decir que cuenta la cantidad de palabras que hay en el poema meta y segun esta cntidad genera los individuos, para que estos sean del mismo tamano. Al crear los indivisuos se crean totalmente aleatprios. esto se implementyo mediante la funcion random. Se cuenta la cantidad de ngramas de dos, de tres, de cuatro y de cinco que ay en la base de datos, y con estos numeros se cran rangos. El numero randos que salga se ubica dentro de estos rangos y dependiendo del rango aleatorio el poema individuo que se esta creando toma oraciones de esa cantidad de n gramas. De esta forma se generan completamente aleatorio. Se toma el maximo de palabras posibles del poema menos seis, para validar que no se pase, y estas seis palabras restantes son tomadas del poema meta.
El barometro de la funcion generar individuo valido esta dada por el maximo de palabras que es la cantidad de palabras que tiene el poema meta. Este alforitmo hace uso de ciclos simples que aportan un valor de n a la funcion y hace una llamada a la funcion agregar faltantes que esta no aporta un valor al alldoritmo ya que solo genera numeros random y asignaciones, por lo que en el peor de los casos el orden de la funcion generar individuos validos es de O(n^3).
Una vez creados individuos validos se procede a tomar dos individuos que se van a cruzar. Para efectual este cruce se genera un valor random que indicara la posicion de cada lista, esta sera la posicion a tomar para crear al nuevo individuo. Esta funcion de cruces realiza un siclo simple dentro de ella por lo que el orden esta dado por O(n).

2) Mutaciones: Como se vio en la explicacion de cruces, se puede ver como el agregar faltantes que privienen del poema meta se esta generando una mutacion, por cada individuo se agegan seis palabras provenientes del poema meta, esto asegura que en algun momentyo se va a llegar a un poema bastante cercano a la meta. Esto sucede en la funcion agregar faltantes, la cual cumple la funcion de mutar. Luego de esto se procede a generar las poblaciones siguientes, por cada poblacion se repetira el codigo de mutaciones y cruces.

III. Experimentos

Para el primer experimeto se hizo una corrida del programa incrementando poblaciones e individuos. Sobre el eje x puede observar la cantidad de poblaciones y sobre el eje y la cantidad de individuos. Entre mas individuos y poblaciones se generen es mas el numero de cruces que se deben efectuar por lo tanto el tiempo de procesamiento incrementa bastante. Comparando con el tiempo emirico se obtuvo que 250 individuos en 100 poblaciones dura 250seg, pero segun el tiempo teorico este debe durar lo mismo

```
private List<string> generarIndividuoValido(int palabrasMeta)
{
    List<string> listaIndividuo = new List<string>();
    int cont = 0;
    List<int> lineas = new List<int>();
    int maxPalabras = palabrasMeta;
    while (cont <= maxPalabras - 6)
    {
        int num = rnd.Next(0, maxPalabras);
        lineas.Add(num);
        int rango = rangosPalabra(num);
        cont += rango;
    }
    int indicesBD = lineas.Count() - 1;
    List<int> lineaFaltante = new List<int>();
    lineaFaltante = agregarFaltantes(maxPalabras - cont);
    for (int i = 0; i < lineaFaltante.Count(); i++)
    {
        lineas.Add(i);
    }
    cont = 0;
    while (cont <= indicesBD)
    {
        int posicion = lineas[cont];
        string palabra = basedatos[posicion];
        listaIndividuo.Add(palabra);
        cont++;
    }
```

Fig. 10. Funcion Generar individio valido

```
        while (cont < lineas.Count())
        {
            int posicion = lineas[cont];
            string palabra = basedatosMeta[posicion];
            listaIndividuo.Add(palabra);
            cont++;
        }
        return listaIndividuo;
    }
```

Fig. 11. Funcion Generar individio valido, continuacion

```
private List<int> agregarFaltantes(int faltante)
{
    List<int> resul = new List<int>();
    if (faltante == 6)
    {
        int num = rnd.Next(dos + 1, tres);
        resul.Add(num);
        int num1 = rnd.Next(dos + 1, tres);
        resul.Add(num1);
        return resul;

    }
    else if (faltante == 5)
    {
        int num = rnd.Next(0, dos);
        resul.Add(num);
        int num1 = rnd.Next(dos + 1, tres);
        resul.Add(num1);
        return resul;

    }
    else if (faltante == 4)...
    else if (faltante == 3)...
    else if (faltante == 2)...
    return resul;

}
```

Fig. 12. Funcion agregar faltantes

```
private void siguientesPoblaciones(List<string> poblacion)
{
    int cont = 0;
    while (cont < poblacion.Count())
    {
        List<string> conjuntoIndividuos = new List<string>();
        conjuntoIndividuos = individuos(basedatos, basedatosMeta, cantidadPalabrasMeta, numIndividuos);
        cruzarIndividuo(conjuntoIndividuos);
        System.IO.File.WriteAllLines(@"C:\Users\camil\OneDrive\Documents\TEC\Análisis\ElPoeta\poblacion" + cont + ".txt", conjuntoIndividuos);
        cont++;
    }
}
```

Fig. 13. Funcion generar siguientes poblaciones

simepre, el numero de individuos al ser constante no debe afectar el orden del algoritmo. or lo tanto se contradice el tie mpo teorico con el practico. Para el segundo experimeto se corrio el programa pero solo generando una base de datos con n grams de dos, luego de tres luego de 4. Se obtuvieron mejores resultados al generar unicamente n gramas de dos, ya que oraciones de dos palabras tenian probabilidades mas altas de conicidir y de generarse reetidamente en el random, al ser un rango mas pequeo. En cuanto al tiempo teorico y practico se tiene tambien una contradiccion, ya que sin importar el n respectivo al n gram. Para el tercer experimeto se corrio el programa creando n grams muy grandes, de n igual a diex y n igual a veinte. Lo que sucedio fue que se escontraron muy pocas o casi ninguna aparicion de estos, por lo tanto un n viable para la generacion de n gramas se determino que fuera un n entre el rango de dos a cinco. El eje x representa el numero de n grams y el eje y representa la cantidad de coincidencias encontradas

IV. CONCLUSIONES

Se logro la implementacion de la solucion de los kakuros, incluida la poda, las permutaciomes. El programa no posee interfaz grafica, y no genera los kakuros, estos deben ser introducidos. Los hilos y los forks no fueron implementados Al realizar un analisis teorico mediante el calculo de O grande se optiene el mayor orden posible que puede llegar a tener dicho algoritmo evaluado.Si el orden de un algoritmo se ha definido correctamente no cambiara. Al comparar los resultados de los analisis teoricos y los empiricos no es mucha la diferencia entre ambos. Esto se debe al resultado de los productos cartesianos que incrementan con el tamao de la matriz ya que las posibilidades de las casillas blancas son mayores. la unica forma de que este orfden se contradijera seria el caso en que un cacuro solo tenga pares de casillas blancas, en este caso el prosucto cartesiano se reduciria notoriamente.

Esto nos afirma que un experiemnto empirico no necesariamente representa el orden real de un algoritmo.

REFERENCES

[1] Brassard, G. and Bratley, P. (2008). Fundamentos de algoritmia. Madrid: Pearson Prentice Hall. Please enable flash to have the best experience

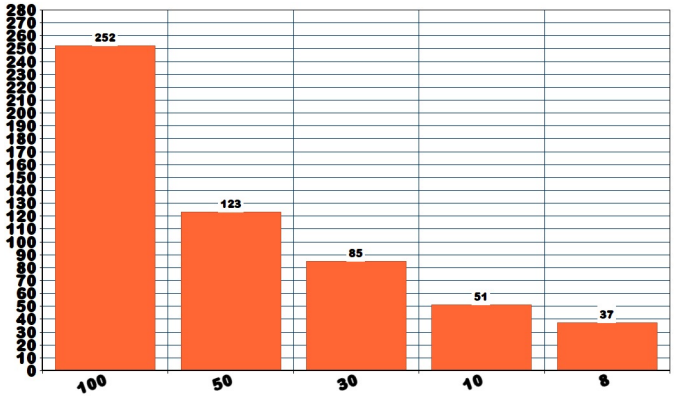

Fig. 14. Experimento uno

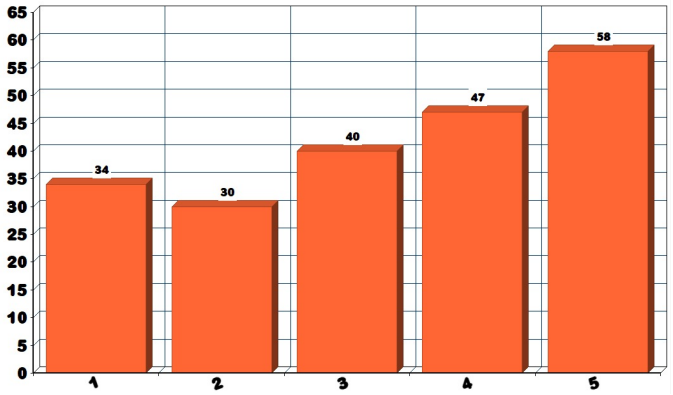

Fig. 15. Experimento dos

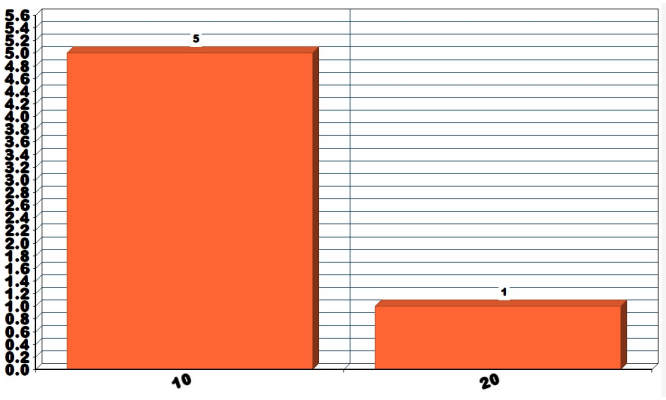

Fig. 16. Experimento tres