Proyecto - Monografía

Domínguez Aspilcueta, Pedro Francisco - 201910375
Ríos Vásquez, Paul Jeremy - 201910038
Lama Carrasco, Miguel Angel - 201910199

Universidad de Ingeniería y Tecnología
Ciencia de la computación
Análisis y Diseño de Algoritmos 1.00
Docente: Gutierrez Alva, Juan Gabriel
TA: Lopez Condori, Rodrigo

1. Problema de codificación de texto

Problema MIN-COD Dada una cadena s sobre el conjunto C, encontrar una codificación de C óptima.

- Sea s una cadena no vacía tal que $s = \{a_1 a_2 \cdots a_n\}$.
- $\{C_0, C_1, C_2, C_3\}$ los modos de codificar a.
- B una cadena de dos bits $\{00, 01, 10, 11\}$ que indica cual C comienza una codificación de s.
- $\{T_0, T_1, T_2, T_3\}$ las cadenas de transición a otro C. Tomando en cuenta que T puede ser vacío si no se requiere ninguna transición.

Podemos suponer que una cadena s siempre puede ser codificada por al menos un modo de codificación C. Y decimos que r es una solución óptima tal que r empieza con una cadena B de dos bits seguido de un número de codificaciones y transiciones mínimos. $r = \{B \cdot C(a_1) \cdot T \cdot C(a_2) \cdot T \cdot C(a_3) T \cdots T \cdot C(a_n)\}$

1.1. Pregunta 1

Heurística Voraz: Utilizar el modo de codificación que permita una codificación mínima para los caracteres a_i y a_{i+1}.

RECIBE: un cadena s con un número de caracteres par no vacía.
DEVUELVE: Una cadena r de bits que codifica a s y su tamaño.

VORAZ(s)
$C_j = C_0$
for i = 0 to n - 1, step = 2 **do**
    C_k es el modo mínimo de codificación válido para a_i, a_{i+1}.
    if $C_j \neq C_k$ **then**
        $r+ = T_k$
    r += $C_k(a_i)$
    r += $C_k(a_{i+1})$
    $j = k$
return r

1.2. Pregunta 2

Sea X una solución óptima para el problema. Existen tres casos dependiendo de la codificación del último caracter a_n de s.

- **Caso 1:** $C_j(a_n) \notin X$. En ese caso, a_n no puede ser codificado por el modo C_j. Por lo que no existe una codificación óptima que incluya a $C_j(a_n)$.
- **Caso 2:** $C_j(a_n) \in X$. En ese caso, a_n puede ser codificado por el modo C_j. Considere que $X' = X/\{a_n\}$. Note que X' es una solución óptima para el subproblema $s = a_1a_2 \cdots a_{n-1}$.
  ¿Porqué? Considere por contradicción que X' no es una solución óptima, entonces existe una solución Y' para este subproblema que codifica con menor tamaño que X'. Luego $Y = Y' \cup \{n\}$ sería una solución mejor que X para el problema original, siendo esto una contradicción. Tomando en cuenta que debe haber una transición si a_{n-1} no puede ser codificado por el modo C_j.

Lemma 2.1: Sea $OPT(i,j)$ el tamaño de la codificación mínima óptima para el subproblema que considera a la subcadena $s' = \{a_1a_2 \cdots a_i\}$.

$$OPT(i,j) \begin{cases} B_j + C_j(a_i) & i = 1; a_i\ se\ \ puede\ codificar\ con\ C_j \\ \infty & a_i\ no\ se\ \ puede\ codificar\ con\ C_j \\ min\{OPT(i-1,j) + C_j(a_i), & i > 1 \\ OPT(i-1,(j+1) \bmod 4) + T_{(j+1) \bmod 4} + C_j(a_i), & \\ OPT(i-1,(j+2) \bmod 4) + T_{(j+2) \bmod 4} + C_j(a_i), & \\ OPT(i-1,(j+3) \bmod 4) + T_{(j+3) \bmod 4} + C_j(a_i)\} & \end{cases}$$

Prueba: Sea X una solución óptima para el subproblema que considera a la subcadena $\{a_1a_2 \cdots a_i\}$. Suponga que a_i puede ser codificado por C_j. Sea $X' = X/\{a_i\}$. Note que X' es una solución óptima para el subproblema. (Vea análisis anterior). Por lo tanto:

- Cuando a_{i-1} puede ser codificado por el modo C_j. Entonces:

$$OPT(i,j) = OPT(i-1,j) + C_j(a_i)$$

- Cuando a_{i-1} no puede ser codificado por el modo C_j. Se debe añadir una transición T_k del modo C_k donde $1 \leq k \leq 4$ que sí codifica a_{i-1}

$$\begin{aligned} OPT(i,j) = min\{&OPT(i-1,(j+1) \bmod 4) + T_{(j+1) \bmod 4} + C_j(a_i), \\ &OPT(i-1,(j+2) \bmod 4) + T_{(j+2) \bmod 4} + C_j(a_i), \\ &OPT(i-1,(j+3) \bmod 4) + T_{(j+3) \bmod 4} + C_j(a_i)\} \end{aligned}$$

Ahora suponga que a_i no puede ser codificado por C_j. Entonces:

$$OPT(i, j) = \infty$$

El análisis anterior nos permite diseñar un algoritmo recursivo para el problema.

1.3. Pregunta 3

RECIBE: un índice final i de una cadena s y un índice j de C.
DEVUELVE: Una cadena r que codifica a s de manera mínima óptima con $C_j(a_i)$.

OPT(i, j)

if (a_i puede ser codificado por C_j) y $i = 1$ **then**
    return $B_j + C_j(a_i)$
if (a_i no puede ser codificado por C_j) **then**
    return ∞
$A_0 = C_j(a_i)$
$A_1 = T_{(j+1) \bmod 4} + C_j(a_i)$
$A_2 = T_{(j+2) \bmod 4} + C_j(a_i)$
$A_3 = T_{(j+3) \bmod 4} + C_j(a_i)$
$r = min(OPT(i-1, j) + A_0, OPT(i-1, (j+1) \bmod 4) + A_1, OPT(i-1, (j+2) \bmod 4) + A_2, OPT(i-1, (j+3) \bmod 4) + A_3)$
return r

RECIBE: un cadena s no vacía.
DEVUELVE: Una cadena que codifica a s de manera mínima óptima.

MIN-COD(s)

Sea a_n el caracter final de s.
return $min(OPT(a_n, 0), OPT(a_n, 1), OPT(a_n, 2), OPT(a_n, 3))$

1.3.1. Complejidad

En *OPT(i,j)* se realizan cuatro llamadas recursivas para intentar codificar con los modos C_1, C_2, C_3, C_4 a cada caracter de la cadena s. Por lo tanto, $T(n) = \Omega(4^n)$.

En *MIN-COD(s)* se envía a *OPT(i,j)* los cuatros modos diferentes con los que se puede codificar el caracter a_n. Sin embargo, aún cuando se llama cuatro veces a OPT, la recursividad total sigue siendo $\Omega(4^n)$.

1.4. Pregunta 4

RECIBE: un índice final i de una cadena s y un índice j de C.
DEVUELVE: Una cadena r que codifica a s de manera mínima óptima con $C_j(a_i)$.

OPT-MEMOIZADO(i, j)

1: **if** i == 0 OR $A[i][j] \neq \infty$ **then**
2:     **return** $A[i][j]$
3: **if** a_i puede ser codificado por C_j **then**
4:     $A_0 = \text{OPT-MEMOIZADO}(i-1, j) + C_j(a_i)$
5:     $A_1 = \text{OPT-MEMOIZADO}(i-1, (j+1) \bmod 4) + T_{(j+1) \bmod 4} + C_j(a_i)$
6:     $A_2 = \text{OPT-MEMOIZADO}(i-1, (j+2) \bmod 4) + T_{(j+2) \bmod 4} + C_j(a_i)$
7:     $A_3 = \text{OPT-MEMOIZADO}(i-1, (j+3) \bmod 4) + T_{(j+3) \bmod 4} + C_j(a_i)$
8:     $A[i][j] = min(A_0, A_1, A_2, A_3)$
9: **return** $A[i][j]$

RECIBE: un cadena s no vacía.
DEVUELVE: Una cadena que codifica a s de manera mínima óptima.

MIN-COD-MEM(s)

1: $A[0][0] = 00$
2: $A[0][1] = 01$
3: $A[0][2] = 10$
4: $A[0][3] = 11$
5: **for** i = 1 to n **do**
6:     $A[i][0 \cdots 3] = \infty$
7: Sea a_n el caracter final de s.
8: **return** $min(OPT(a_n, 0), OPT(a_n, 1), OPT(a_n, 2), OPT(a_n, 3))$

1.4.1. Complejidad

En *OPT(i,j)* se realizan cuatro llamadas recursivas para intentar codificar con los modos C_1, C_2, C_3, C_4 a cada caracter de la cadena s. Sin embargo, una vez que se haya calculado la solución $A[i][j]$ esta se guarda en una matriz A. Asumiendo que la concatenación de bits de las líneas 4-7 tiene costo $O(1)$ entonces la recurrencia memoizada es $O(n)$.

En *MIN-COD(s)* se envía a *OPT(i,j)* los cuatros modos diferentes con los que se puede codificar el caracter a_n. En la matriz A se asignan los valores previos iniciales B, así como valores ∞ para el resto de la matriz.

1.5. Pregunta 5

RECIBE: una cadena s.
DEVUELVE: Una cadena r que codifica a s de manera mínima óptima.

MIN-COD-DIN(s)

Sea A una matriz que guarda la solución.
$A[0][0] = 00$
$A[0][1] = 01$
$A[0][2] = 10$
$A[0][3] = 11$
for i = 1 to n **do**
    $A[i][1] = \infty$
    $A[i][2] = \infty$
    $A[i][3] = \infty$
    $A[i][4] = \infty$
    for j = 0 to 3 **do**
        if Si a_i puede ser codificado con C_j **then**
            $M_1 = A[i-1][j] + C_j(a_i)$
            $M_2 = A[i-1][(j+1) \bmod 4] + T_{(j+1) \bmod 4} + C_j(a_i)$
            $M_3 = A[i-1][(j+2) \bmod 4] + T_{(j+2) \bmod 4} + C_j(a_i)$
            $M_4 = A[i-1][(j+3) \bmod 4] + T_{(j+3) \bmod 4} + C_j(a_i)$
            $A[i][j] = min(M_1, M_2, M_3, M_4)$
$r = min(A[n][1], A[n][2], A[n][3], A[n][4])$
return r

1.5.1. Complejidad

En *MIN-COD-DIN(s)* asumiendo que la concantenación de cadenas de bits es $O(1)$ es claro que el algoritmo es $\Omega(n)$. Los cuatros modos diferentes de la recurrencia aumentan el costo lineal en cuatro. La respuesta óptima del algoritmo está en el valor mínimo de $A[n][1], A[n][2], A[n][3]$ y $A[n][4]$.

2. Software de codificación y decodificación

2.1. Pregunta 6 (Codificación heurística)

```cpp
std::string min_cod_heu(std::string cadena)
{
    std::string output = "";

    std::string temp;
    std::string temp_anterior = "C0";

    for (int i = 0; i < cadena.length() - 1; i+=2)
    {
        temp = interseccion_minima(cadena[i], cadena[i+1]);
        if (temp_anterior != temp)
        {
            if (temp_anterior == "C0")
            {
                output += std::bitset<2>(int(temp[1] - '0') - 1).to_string();
            }
            else
            {
                output += traduccion(temp,temp_anterior);
            }
        }
        output += codificar(cadena[i], temp);
        output += codificar(cadena[i+1], temp);
        temp_anterior = temp;
    }

    if (cadena.length() % 2 != 0)
    {
        if (cadena.length() == 1)
        {
            temp = interseccion_minima(cadena.back(), cadena.back());
            output += std::bitset<2>(int(temp[1] - '0') - 1).to_string();
            output += codificar(cadena.back(), temp);
        }
        else
        {
            temp = interseccion_minima(cadena[cadena.length()-2], cadena.back());
            if (temp_anterior != temp)
            {
                if (temp_anterior == "C0")
                {
                    output += std::bitset<2>(int(temp[1] - '0') - 1).to_string();
                }
                else
                {
                    output += traduccion(temp,temp_anterior);
                }
            }
            output += codificar(cadena.back(), temp);
        }
    }
    return output;
}
#endif
```

Figura 1: Código en C++ - Algoritmo heurístico

2.2. Pregunta 7 (Codificación óptima)

```cpp
std::string min_cod_din(std::string s)
{
    std::vector<std::vector<std::tuple<std::string, int, int>>> A;

    A.resize(s.size()+1, std::vector<std::tuple<std::string, int, int>>(4));
    int i, j;


    for (i = 0; i < s.size()+1; i++)
    {
        for (j = 0; j < 4; j++)
        {
            A[i][j] = std::make_tuple("",-1,-1);
        }
    }

    int m1, m2, m3, m4;

    std::string c1, c2, c3, c4;
    int ic1, ic2, ic3, ic4;

    std::string t1, t2, t3;

    for(i = 0; i < 4; i++)
    {
        A[0][i] = std::make_tuple(B[i], 2, -1);
    }

    for(i = 1; i <= s.size(); i++)
    {
        A[i][0] = std::make_tuple(" ", MAX, -1);
        A[i][1] = std::make_tuple(" ", MAX, -1);
        A[i][2] = std::make_tuple(" ", MAX, -1);
        A[i][3] = std::make_tuple(" ", MAX, -1);

        for(j = 0; j < 4; j++)
        {
            c1 = codificar(s[i-1], cod[j]);

            ic1 = c1.size();

            if(ic1)
            {
                t1 = traduccion(cod[j],cod[(j+1)%4]);
                t2 = traduccion(cod[j],cod[(j+2)%4]);
                t3 = traduccion(cod[j],cod[(j+3)%4]);

                m1 = std::get<1>(A[i-1][j]) + ic1;
                m2 = std::get<1>(A[i-1][(j+1)%4]) + ic1 + t1.size();
                m3 = std::get<1>(A[i-1][(j+2)%4]) + ic1 + t2.size();
                m4 = std::get<1>(A[i-1][(j+3)%4]) + ic1 + t3.size();

                int v[] = { m1,m2,m3,m4 };
                int i1 = minelement(v);

                if(i1 == m1) A[i][j] = std::make_tuple(c1, m1, j);
                else if (i1 == m2) A[i][j] = std::make_tuple(t1 + c1, m2, (j+1)%4);
                else if (i1 == m3) A[i][j] = std::make_tuple(t2 + c1, m3, (j+2)%4);
                else if (i1 == m4) A[i][j] = std::make_tuple(t3 + c1, m4, (j+3)%4);
            }


        }
    }
```

Figura 2: Código en C++ - Algoritmo óptimo

2.3. Pregunta 8 (Decodificación)

```cpp
std::string decode(std::string s)
{
    std::string sreturn;

    int codact;

    std::string temp = s.substr(0,2);
    codact = std::stoi(temp, 0, 2);


    for(int i = 2; i < s.size();)
    {
        if(codact == 0 )
        {
            temp = s.substr(i,3);
            int retraduc = retraduccion(temp,temp.size());

            if( retraduc == -1) sreturn += decodec1(temp);
            else codact = retraduc;
            i+=3;
        }
        else if(codact == 1 )
        {
            temp = s.substr(i,5);
            int retraduc = retraduccion(temp,temp.size());

            if( retraduc == -1) sreturn += decodec2(temp);
            else codact = retraduc;

            i+=5;
        }
        else if(codact == 2 )
        {
            temp = s.substr(i,6);
            int retraduc = retraduccion(temp,temp.size());

            if( retraduc == -1) sreturn += decodec3(temp);
            else codact = retraduc;

            i+=6;
        }
        else if(codact == 3 )
        {
            temp = s.substr(i,7);
            int retraduc = retraduccion(temp,temp.size());

            if( retraduc == -1) sreturn += decodec4(temp);
            else codact = retraduc;

            i+=7;
        }
    }
    return sreturn;
}
#endif
```

Figura 3: Código en C++ - Decodificador

2.4. Pregunta 9 (Software de codificación y decodificación)

```cpp
std::cout<<std::endl<<"MENU"<<std::endl;
std::cout<<"1. Generar Aleatorio"<<std::endl;
std::cout<<"2. Codificar"<<std::endl;
std::cout<<"3. Decodificar"<<std::endl;

do
{
    std::cout<<"\nEliga una Opción: ";
    std::cin>>choice;
}while(choice<0 and choice>4);


switch (choice)
{
case 1:
    aleatorio();
    break;
case 2:
    codificar();
    break;
case 3:
    decodificar();
    break;
}
```

Figura 4: Menú del software

Con el objetivo de que el software sea de utilidad y fácil manejo, se creó un menú del cual de un archivo recibido realiza:

- **aleatorio()** Solicita en consola el tamaño n de la cadena que se creará de manera aleaoria. La longitud máxima puede ser de 1M.
- **codificar()** Si se elige esta opción, se recibe un archivo de texto al que se desee codificar, ya sea con un método heurístico u óptimo mínimo. El resultado es guardado como un archivo binario comprimido.
- **decodificar()** Se recibe un archivo binario el cual es decodificado y guarda la respuesta en un archivo de texto.

2.5. Pregunta 10 (Análisis experimental)

Para el analisis experimental, se comparó ambos algoritmos del proyecto con diferentes tamaños de tests cases.

Size del encoding	**Heurística(ms)**	**Dinámica(ms)**
10	0.015	0.056
100	0.041	0.378
1000	0.354	3.737
10000	3.252	47.679
100000	37.132	19123.2
1000000	299.102	194970