Multimédia II

Universidade Fernando Pessoa
2015/2016

Método Run Length Encoding (RLE)

Elaborado por:
Gonçalo Silva 26329
Pedro Galocha 25999
João Alves 27785

Índice

1. Introdução 4
1.1 Objectivos do trabalho 4
1.2 Justificação 4
1.3 Áreas de aplicação 4
2. O algoritmo do “título do codec ou método de compressão abordado neste artigo” ... 5
2.1 Codificação 5
2.2 Descodificação 6
2.3 Exemplo de aplicação 7
2.3.1 Codificação 7
2.3.1 Descodificação 9
3. Implementação do algoritmo “título” 10
4. Comparação com outros métodos semelhantes 10
5. Conclusão 10
Bibliografia 10

Resumo
Este artigo cientifico para a disciplina de multimédia 2 tem como objetivo o estudo do Método Run Length Encoding, mais especifico os algoritmos de compressão e descompressão direcionada ao texto. Também pretendemos detalhar como funciona cada algoritmo assim como o seu código.

1. Introdução

- Neste artigo cientifico vamos abordar o codec Run Length Encoding.
- Run Length Encoding é um processo para comprimir caracteres quando existe uma sequência longa de caracteres repetidos (4 ou mais). Este funcionamento é considerado codificação simples.
- O codec Run Length Encoding são codificadores sem predas e supressão de sequências repetitivas.

1.1 Objectivos do trabalho

O objetivo deste trabalho é apresentar um algoritmo numa linguagem especifica para o método Run Length Encoding, a linguagem que escolhemos foi o javascript.

1.2 Justificação

Após uma reunião com o professor foi sugerido o tema RLE, apesar de este método não ser eficiente pois na língua portuguesa as repetições de 2 letras iguais consecutivas são raras mas como é um algoritmo bastante usado e também escolhemos este codec por ser uma compressão sem perdas.

1.3 Áreas de aplicação

Este codec é usados para a compressão de texto e imagens, especialmente em áreas onde não se pode perder informação pois estes métodos são sem perdas.
Neste trabalho especifico só vamos abortar a compressão de texto.

2. O algoritmo do Método Run Length Encoding

2.1 Codificação

- O algoritmo de codificação funciona da seguinte forma:
  - Determinar uma FLAG que não exista no texto a comprimir.
  - Ler um caracter. Ler os próximos caracteres enquanto forem iguais ao primeiro caracter lido.
  - Se o número total de caracteres lidos for igual ou superior a 4, comprimir essa cadeia de caracteres da seguinte forma: FLAG + nº de repetições + caracter.
  - Se o numero total de caracteres lidos for inferior a 4, não se efetua compressão, logo essa cadeia de caracteres permanece inalterável.

- Algoritmo de codificação em pseudo-código

```
INICIO
VARIAVEIS
flag,i,c,j,count,compress,string;

i ← 0;
j ← 0;
count ← 0;
flag ← searchForFlag();
string ← “aaabbbssscccddd”;
size ← length of string
compress ← “ “

FOR i<-0 to size
        c ← caracter na posição i
        FOR j← i to size
                IF c == caracter na posição j
                        count ← count + 1;
                ELSE
```

```
                    IF count >= 4

                        compress ← compress + flag + count + c;
                        i ← i + count -1 ;
                        count ← 0;
                        END FOR
                    ELSE
                        compress ← compress + substring(i , i + count);
                        i ← i + count - 1;
                        count ← 0;
                        END FOR
                    END IF
                END IF
            END FOR
        END FOR
        FIM
```

2.2 Descodificação

- O algoritmo de descodificação funciona da seguinte forma:
  - Flag previamente definida.
  - Ler um caracter. Se esse caracter for flag identifica i+1 como sendo o número de repetições para o char na posição i+2;
  - Se não for flag, copia o char diretamente para a string auxiliar.
- Algoritmo de descodificação em pseudo-código

```
INICIO
VARIAVEIS
FLAG,i,c,j, Nrepetições ,Decompress,STRrepets;

FOR i=0 , i< size, i++
IF( string.char(i)==FLAG)
Nrepetições=parseInt.String.char(i+1);
c = string.char(i+2);
FOR(j=0;j<Nrepetições;j++)
STRrepets= STRrepets.concat( c ) ;
```

END FOR

Decompress = Decompress.concat(STRrepets);

STRrepets = “”;

i =i+2

END IF

ELSE

Decompress = Decompress.concat(string.char(i)) ;

END ELSE

END FOR

Imprime_String_Descomprimida_RLE(Decompress);

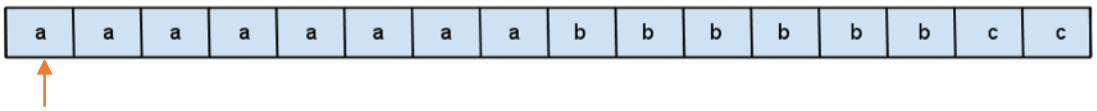
FIM

2.3 Exemplo de aplicação

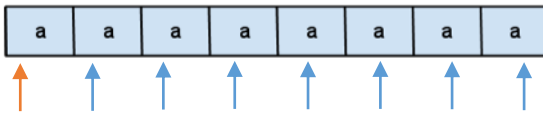
2.3.1 Codificação

FLAG: #

Lê primeira letra: ‘a’



Lê as próximas letras enquanto forem iguais à letra ‘a’.

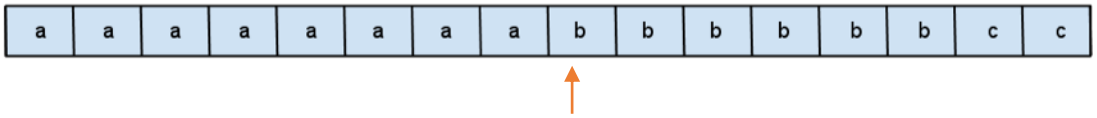


Conjunto total letras repetidas: a a a a a a a a (8x).

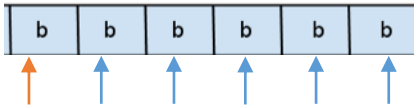
Numero de Repetições >= 4 logo, é comprimido em: **#8a**

Efetua um salto de 8 posições.

Lê primeira letra: ‘b’



Lê as próximas letras enquanto forem iguais à letra 'b'.

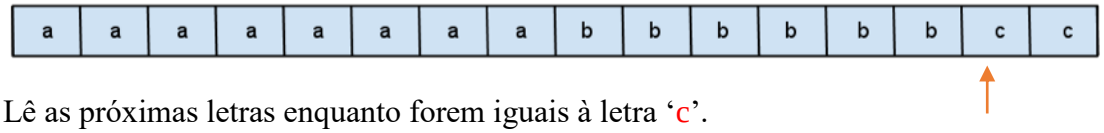


Conjunto total letras repetidas: b b b b b b (6x).

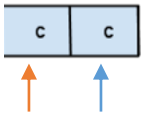
Numero de Repetições >= 4 logo, é comprimido em: **#6b**

Efetua um salto de 6 posições.

Lê primeira letra: 'c'



Lê as próximas letras enquanto forem iguais à letra 'c'.



Conjunto total letras repetidas: c c (2x).

Numero de Repetições < 4 logo, não é comprimido. Permanece como estava: **cc**

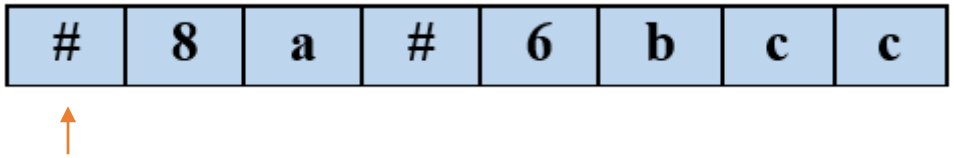
Resultado final da compressão:

#	8	a	#	6	b	c	c

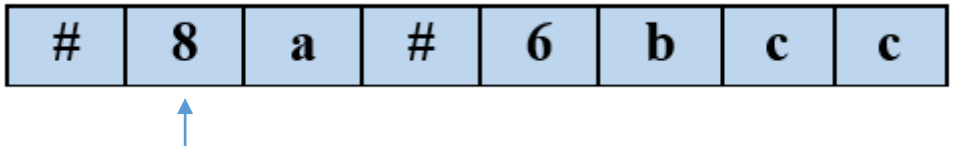
Rácio de Compressão: $\frac{16}{8} = 2:1$

2.3.2 Descodificação

Flag: #
Cria uma StringAuxiliar vazia.
Lê a primeira posição da String original.



Encontra Flag.

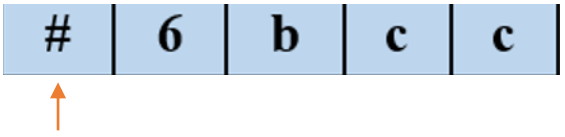


Vai na casa i+1,da string original, obter o número de repetições para o char e o size de uma string char_repete.

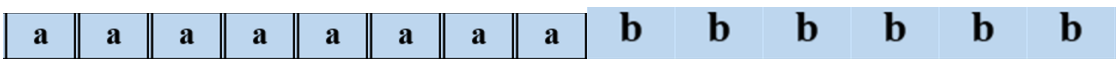
Cria a string char_repete com size 8 , e com o seguinte conteúdo:

De seguida , faz a concatenação / junção com a string Auxiliar,como esta estava vazia vai ficar com o conteúdo igual à da char_repete.

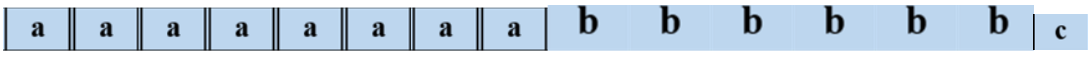
De seguida zeramos a String char_repete para futuro uso.
Na principal , damos dois saltos em i , ou seja , i=i+2, incrementa e é como se tivesse avançado 3 posições.



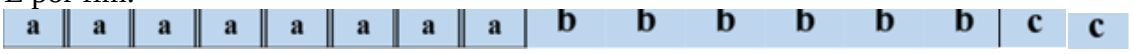
Encontra flag e repete os passos anteriores.



No caso de encontrar apenas um simples char , ou seja, que não seja uma Flag, ele irá simplesmente concatenar à StringAuxiliar.



E por fim:



3. Implementação do algoritmo Run-length Encoding

Para desenvolver o algoritmo criamos um programa usando linguagem javascript.
A nossa abordagem é especifica para a compressão de caracteres e não de números.
Esta limitação deve-se ao facto de o texto a comprimir ser lido caracter a caracter.
Em vez disso, deveria ser lido em modo binário, ou seja, byte a byte o que já resolvia a limitação da compressão de caracteres apenas.

4. Comparação com outros métodos semelhantes

O método Run-length Encoding é um codec sem perdas e de supressão de sequências repetitivas.
Como já foi mencionado o codec Run-length Encoding não é dos mais eficientes para os diferentes idiomas porque é difícil ter a mesma letra várias vezes seguidas.

5. Conclusão

Na realização deste artigo cientifico ficamos a saber mais ao pormenor os algoritmos de RLE e conseguimos aprender uma nova linguagem pois nenhum elemento do grupo tinha trabalhado anteriormente com javascript. Deparamos com algumas dificuldades nomeadamente, com a leitura em modo binário do texto a comprimir. Apenas conseguimos implementar o modo de leitura caracter a caracter.
No entanto, o conceito já se encontra implementado, porém o modo de leitura deve ser alterado para modo binário para uma maior eficiência dos algoritmos de compressão e descompressão.
O método de compressão RLE não é eficiente se for utilizado, por exemplo, menos de quatro letras repetidas e seguidas, por este motivo e em comparação com os outros processos este é um pouco ineficaz.
Este trabalho pode ser melhorado através do conhecimento de melhores codificadores.

Bibliografia

- Salomon, D., “Data Compression”, Springer, 4th. Edition, 2007
- Nuno Magalhães Ribeiro, Método Run Length Encoding, http://multimedia.ufp.pt/, 23 de Maio de 2016
- Nuno Ribeiro e José Torres, “Tecnologias de Compressão Multimédia”, FCA, Setembro de 2009