Universidade do Minho, Departamento de Informática
MIEI, 2020-2021

Comunicação de Dados

Relatório do Trabalho Prático N°1

Data de entrega: 03-01-21

Autores:

Luís Francisco
a93209
MIEI

Diogo Casal Novo
a88276
MIEI

Catarina Gonçalves
a93259
MIEI

Mafalda Costa
a83919
MIEI

Tiago Carneiro
a93207
MIEI

Gonçalo Santos
a93279
MIEI

Alexandre Silva
a93315
MIEI

Gabriela Prata
a93288
MIEI

Índice

Organização

De acordo com a divisão do trabalho em 4 módulos, decidimos fazer grupos de 2, cada um com o seu módulo respetivamente:

Módulo F: Francisco Faria (a93209) e Diogo Casal Novo (a88276)

Módulo T: Catarina Gonçalves (a93259) e Mafalda Costa (a83919)

Módulo C: Tiago Carneiro (a93207) e Gonçalo Santos (a93279)

Módulo D: Alexandre Silva (a93315) e Gabriela Prata (a93288)

Estratégias Utilizadas e Otimizações

- **Módulo F**

Para o módulo F, decidimos optar por começar por preencher um array de caracteres correspondente ao resultado do bloco apos a compressão RLE. Seguidamente, avaliamos se a compressão é eficaz, e caso seja ou o utilizador o tenha explicitado, usamos esse mesmo array para preencher o ficheiro RLE e para preencher também uma estrutura com as informações necessárias á criação dos ficheiros .freq e .rle.freq. Sendo que cada estrutura corresponde a um bloco, estas vão ser organizadas numa lista ligada para que posteriormente se possa construir os ficheiros que faltam.

- **Módulo T**

Para o módulo T, concordamos desde o início em criar uma lista de structs tornando possível a ordenação da lista, tanto por frequências como por valor do símbolo. Com isto em mente, conseguimos ler os ficheiros e transformá-los diretamente nessa lista (no entanto não o fazíamos assim inicialmente). Essa lista facilitava-nos bastante o seu manuseamento na criação dos códigos Shano-Fano. Finalmente, ordenávamos essa lista novamente de modo a obter a ordem necessária para a criação do ficheiro de saída.

- **Módulo C**

Neste módulo decidimos utilizar o método das threads de forma a tornar o nosso código o mais rápido possível, cada uma criando matrizes de símbolos para escrever o resultado.

- **Módulo D**

Para este módulo, inicialmente pensamos em adotar um modelo que usava um struct de arrays no entanto rapidamente nos apercebemos da sua ineficiência e passamos a usar um modelo de procura com árvore binaria o que facilitou bastante a descodificação e a própria interpretação do código em si.

Análise das principais Funções

- **Módulo F**

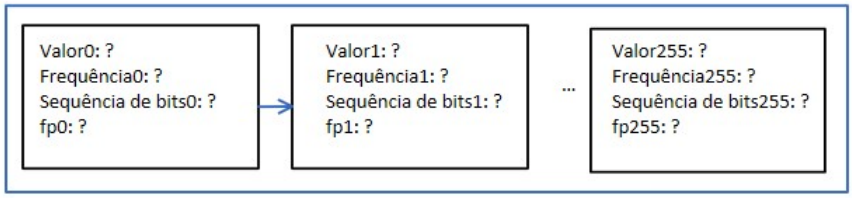
Para a compressão RLE usamos duas funções principais: *applyRLECompression* e *freqFileBuild*.

A primeira é a função que cria o ficheiro RLE e a lista da struct *blockfreq* para cada bloco. A struct *blockfreq* consiste em guardar a frequência dos carateres e o tamanho de um bloco do ficheiro original e do ficheiro comprimido RLE para a construção dos ficheiros freq e freq.RLE.

A segunda é a função que cria o ficheiro freq e RLE.freq, com auxílio da lista da struct *blockfreq* criada na função *applyRLECompression*.

- **Módulo T**

Na leitura temos a função mais importante *converte1blocoData*. Esta é a função que cria cada lista de struct para cada bloco, em que cada struct consiste em guardar todos os dados de cada frequência, incluíndo o valor do símbolo, a sua frequência, o espaço para a sua futura sequência de bits e o seu fp.



No tratamento de dados temos a função *shannonFano* recursiva, que com a ajuda de funções auxiliares como a *indiceFreqMeio* (descobre o índice onde se divide a lista) e *adiciona* (adiciona um bit à codificação) consegue criar a codificação de cada símbolo.

```
if (end!=start){
    for (i=start ; i<indFreqMeio+1 ; i++)
        adiciona(0,lista,i);

    for (i; i<end+1 ; i++)
        adiciona(1,lista,i);

    shannonFano(lista,start,indFreqMeio);
    shannonFano(lista,indFreqMeio+1,end);
```

"shannonFano"

```
lista[indice].fp++;
lista[indice].bits[lista[indice].fp] = i;
```

"adiciona"

- **Módulo C**

```
int read(FILE *fp,int *tblocos,pdarr codes,unsigned char buffer[],int c)
```

Esta função recebe o ficheiro e utilizando um buffer armazena a informação presente no mesmo para que depois ao longo do código sejamos capazes de utilizar essa informação para escrever o resultado final. Uma das vantagens desta função é o facto de ser capaz de guardar os simbolos presente no ficheiro. No entanto, uma falha presente é o facto de utilizar um buffer relativamente pequeno

```
void pencode(ptarg arg)
```

Esta é a função designada a cada thread, sendo ela quem cria e compila cada bloco do ficheiro. Uma das principais características desta função é ser capaz de sincronizar todas as threads na hora de escrita.

```
int moduloC(char *path)
```

Nesta função, são inicializadas as variáveis e é também criado o sistema de threads.

- **Módulo D**

Para a descompressão usamos duas funções principais: decompressSF e decompressRLE.

A primeira é responsável pela descompressão RLE. Esta é feita bloco a bloco tendo em consideração o tamanho dos mesmos obtidos através de:

```
if(modo == 'L'){
    int _bloco;
    for(_bloco = 0; _bloco < nr_blocos; _bloco++){
        tam_bloco = buffer_sizes_rle[_bloco];
        CheckPointer(buffer_rle = (unsigned char*) malloc(sizeof(unsigned char)*tam_bloco));
        fread(buffer_rle, sizeof(char), tam_bloco, fp_rle);
        decompressBlockRLE(fp_original, tam_bloco, buffer_rle);
        free(buffer_rle);
    }
```

FICHEIRO .FREQ

```
}else{
    tam_bloco = *buffer_sizes_rle; /* No caso de tam_bloco diferenciar do valor dado pelo utilizador
    CheckPointer(buffer_rle = (unsigned char *) malloc(sizeof(unsigned char)*tam_bloco));
    while(tam_bloco == *buffer_sizes_rle){
        tam_bloco = fread(buffer_rle, sizeof(char), tam_bloco, fp_rle);
        fseek(fp_rle , decompressBlockRLE(fp_original,tam_bloco,buffer_rle) , SEEK_CUR);
        nr_blocos++;
    }
    free(buffer_rle);
}
```

UTILIZADOR

No caso do utilizador pretender que sejam executadas ambas as descompressões, esta função, apesar de ser referente á descompressão ShannonFano, contem uma parte do código relativa à descompressão RLE, tendo em vista a otimização do código, mais concretamente, reutilização de um buffer, em vez de ler um ficheiro.

```
if(fp_new2) decompressBlockRLE(fp_new2,tam_bloco_new,buffer_new);
```

Resultados da execução dos ficheiros exemplificativos

- **Módulo F**

Ficheiro	Tempo (ms)
aaa.txt	32
aaa.txt.rle	33
Shakespeare.txt	106
Shakespeare.txt.rle	107
bbb.zip	10113
bbb.zip.rle	10121

- **Módulo T**

Ficheiro	Tempo (ms)
aaa.txt.freq	1.23
aaa.txt.rle.freq	2.716
Shakespeare.txt.freq	11.954
Shakespeare.txt.rle.freq	14.594
bbb.zip.freq	142.038
bbb.zip.rle.freq	148.24

- **Módulo C**

Ficheiro	Tempo (ms)
aaa.txt	0.658
aaa.txt.rle	0.542
Shakespeare.txt	62.769
Shakespeare.txt.rle	56.935
bbb.zip	5446.5751
bbb.zip.rle	5468.0087

Nota : Devido ao facto de estarmos a usar threads, não somos capazes de calcular o tempo correto que o programa demora a executar, portanto qualquer valor que apareça nesta tabela, tal como qualquer valor apresentado no nosso código como sendo o tempo de execução será sempre superior ao verdadeiro tempo de execução.

- **Módulo D**

Ficheiro	Tempo de descompressão(ms)	Argumentos
aaa.txt.rle	0.521	-r
aaa.txt.rle	0.444	-r -b K
aaa.txt.rle.shaf	0.615	-s
aaa.txt.rle.shaf	0.770	Sem argumentos (RLE & SF)
Shakespeare.txt.rle	31.624	-r
Shakespeare.txt.rle	33.329	-r -b
Shakespeare.txt.rle.shaf	130.168	-s
Shakespeare.txt.rle.shaf	153.220	Sem argumentos (RLE & SF)
bbb.zip.rle	2949.034	-r
bbb.zip.rle.shaf	31526.555	-s
bbb.zip.rle.shaf	34746.434	Sem argumentos (RLE & SF)

Conclusões

- **Módulo F**

Neste módulo, as otimizações que poderiam ser feitas passavam por reduzir a quantidade de informação que estaria guardada em memória. Na função *applyRLECompression*, há várias variáveis que são guardadas para só serem usadas posteriormente. Este é o caso da estrutura que guarda informação para criar os ficheiros de frequências.

- **Módulo T**

Neste módulo gostávamos de ter implementado as threads na sua execução e também uma abordagem não recursiva da função que realiza o Shannon-Fano. No entanto encontramos conflitos nas atualizações das variáveis e acabávamos por não conseguir os solucionar.

- **Módulo C**

Visto termos concluído o trabalho com a aplicação de threads gostaríamos de ter conseguido, adicionalmente, mostrar o verdadeiro tempo que o programa demora a executar.

- **Módulo D**

Concluindo, achamos que o nosso código poderia estar mais otimizado. Com isto em mente, gostaríamos de ter implementado threads, e o método que recorre a matrizes abordado pelos docentes, na parte relativa á descodificação Shannon-Fano.

Referências bibliográficas

- **Módulo T**

Algoritmo de ordenação na função *ordenaStructPorFreq* fornecido na cadeira de Algoritmos de 2º ano na universidade do Minho.