Relatórios Dos Trabalhos de Teoria de Informação

24 de Dezembro de 2011

Trabalho elaborado por: Luís Polha n.20464 e Marlene Oliveira n.25999

Introdução

No âmbito da disciplina de Teoria da Informação foi-nos solicitada a elaboração de um programa que procedesse à compressão e descompressão de um ficheiro de imagem no formato PBM e a elaboração de uma rotina em Python que permita calcular a capacidade de um canal discreto sem memória recorrendo ao algoritmo de Blahut. A elaboração do programa de compressão/descompressão corresponde à primeira fase do trabalho final desta disciplina e a elaboração da rotina em Python que permitirá calcular a capacidade de um canal discreto sem memória consiste na segunda fase do mesmo. .

A compressão de ficheiros é um processo que permite codificar a informação contida num ficheiro de modo a economizar espaço. A compressão pode ter um de dois tipos: compressão com perdas ou compressão sem perdas. Se não existir perda de informação durante a compressão de um ficheiro, será possível recuperar o ficheiro original a partir da descompressão do ficheiro comprimido, ou seja, a informação obtida após a descompressão é igual à informação original. No caso da compressão com perdas, a informação obtida após a descompressão apresenta diferenças em relação à informação original. Porém, esta informação será suficientemente semelhante à informação original para poder ser considerada útil. Esta perda de informação pode ser degenerativa se, à medida que são efetuadas compressões sucessivas, se perde cada vez mais informação. A perda de informação depende do algoritmo de compressão que é utilizado.

A compressão de um ficheiro é efetuada com recurso a um algoritmo de compressão, algoritmo esse que deverá ser o mais eficaz possível tanto ao nível do tempo que demora a comprimir um ficheiro como à forma como efetua a compressão do mesmo. Porém, mesmo que um algoritmo de compressão seja o mais eficaz possível, quando aplicado várias vezes ao mesmo ficheiro, este chegará a um ponto em que não poderá comprimir o ficheiro sem que ocorra perda de informação.

Como foi referido anteriormente, neste trabalho pretende-se desenvolver um programa que proceda à compressão de um ficheiro de imagem no formato PBM. O formato PBM é um dos formatos de imagem definidos pelo projecto Netpbm e, citando, "*serves as the common language of a large family of bitmap image conversion filters*"[1] . O formato PBM pode conter uma imagem ou uma sequência de imagens. Citando: "*There are na data, delimiters, or padding before, after, or between images*."[2]. O formato que utilizaremos neste trabalho será PBM P4, um dos dois formatos monocromáticos que o formato PBM possui. O formato PBM tem ainda um outro formato monocromático, PBM P1, cuja única diferença em relação ao formato PBM P4 é, citando, "*There are no fill bits at the end of a row*" [3]. Em ambos os formatos cada bit representa um pixel, sendo o bit 0 um pixel branco e o bit 1 um pixel preto. No formato PBM P4 os bits são lidos da esquerda para a direita e armazenados do bit mais significativo para o bit menos significativo em cada byte do ficheiro, sendo estes armazenados do início para o final do ficheiro. Os formatos PBM P1 e P4 são bastante ineficientes, servindo apenas como uma "ponte" entre dois programas. O formato P4 possui a seguinte estrutura:

1. Um identificador para o tipo de ficheiro, que consiste em dois caracteres. Neste trabalho, o identificador será "P4".
2. Um espaço em branco.[4]
3. Um caractere ASCII que corresponde ao comprimento da imagem.
4. Um espaço em branco.
5. Um caractere ASCII que corresponde à largura da imagem.
6. Um único espaço em branco (neste caso, normalmente, utiliza-se o caractere ASCII correspondente ao newline). Este espaço em branco fará a delimitação entre os bits que compõem a imagem e todos os outros elementos referidos anteriormente.
7. "*A raster of Height rows, in order from top to bottom. Each row is Width bits, packed 8 to a byte, with don't care bits to fill out the last byte in the row.*" [5]
8. Antes do espaço em branco que delimita o início dos bits que compõem a imagem podem ainda ser colocados comentários, demarcados com o caractere "#". Quaisquer caracteres que se encontrem após o caractere que delimita um comentário serão ignorados até que seja encontrado um carriage return ou um newline. Caso o comentário se encontre imediatamente antes do início dos bits que compõem a imagem, um único newline não será suficiente para fazer cumprir as funções do espaço branco descrito no ponto 6.

A capacidade de um canal discreto e sem memória corresponde ao máximo da informação mútua média, onde a maximização é efectuada considerando todas as distribuições de probabilidade dos símbolos da fonte. Analiticamente, é difícil encontrar uma solução para o cálculo da capacidade de um canal arbitrário. Em 1972, Richard Blahut e Suguru Arimoto desenvolveram um algoritmo que permite calcular a capacidade de um canal de transmissão de dados. Este algoritmo baseia-se no facto de que a capacidade do canal é limitada superior e inferiormente por funções de uma matriz Px. Baseando-se nestes limites, o algoritmo irá actualizar recursivamente Px. O algoritmo de Blahut será, como foi referido anteriormente, utilizado numa rotina em Python que permita calcular a capacidade de vários tipos de canais discretos e sem memória.

[1] – Citado de: http://netpbm.sourceforge.net/doc/pbm.html#index (Acedido em: 1-12-2011)

[2] – Citado de: http://netpbm.sourceforge.net/doc/pbm.html#index (Acedido em: 1-12-2011)

[3] – Citado de: http://netpbm.sourceforge.net/doc/pbm.html#index (Acedido em: 1-12-2011)

[4] – NOTA: Este espaço em branco pode ser um caractere ASCII correspondente ao espaço ou um caractere ASCII correspondente ao tab ou um caractere ASCII correspondente ao carriage return ou um caractere ASCII correspondente ao newline.

[5] – Citado de: http://netpbm.sourceforge.net/doc/pbm.html#index (Acedido em: 1-12-2011)

Conteúdo

I Compressão e Descompressão **5**

1 A Compressão do Ficheiro **5**

2 Descompressão do Ficheiro **6**

3 O Nosso Ficheiro de Imagem **7**

II Capacidade de Canais Discretos sem Memória **8**

III Conclusão **11**

4 Referências **12**

Parte I
Compressão e Descompressão

1 A Compressão do Ficheiro

Para efectuar a compressão do ficheiro foi elaborado um algoritmo relativamente simples, que utiliza dicionários para armazenar blocos de informação. O algoritmo de compressão funciona do modo seguinte:

Lê um bloco de informação do array que contém os pixeis da imagem.

Se este bloco de informação já se encontra no dicionário, exporta o índice do dicionário correspondente ao bloco. Se o bloco de informação não se encontra no dicionário, coloca-se o bloco na primeira posição desocupada no dicionário e exporta-se o índice correspondente ao bloco.

Repete o processo até ter sido processada toda a informação constante no buffer que contém os pixeis da imagem.

Exporta o dicionário para um ficheiro, que posteriormente será utilizado para efectuar a descompressão da imagem.

O programa que fará a compressão do ficheiro de imagem é constituído pelas funções *fileOperations*, *pesqDic* e comprime. A função *fileOperations* terá como principal funcionalidade ler o ficheiro de imagem para um *buffer* temporário e subdividi-lo por três outros *buffers*. Esta função verifica também se o ficheiro que se pretende comprimir é aberto correctamente e se este se encontra no formato PBM que pretendemos tratar. A função *fileOperations* recebe como argumento o nome do ficheiro a comprimir. Esta função abre o ficheiro no modo de leitura e binário, utilizando a função *fopen*. Se o ficheiro for aberto correctamente, a função *fopen* irá retornar o apontador para o ficheiro. Caso contrário, esta função retornará um *NULL pointer*. *fileOperations* utilizará a informação retornada por *fopen* para verificar se o ficheiro foi aberto com sucesso. Caso o ficheiro não tenha sido aberto correctamente, o programa escreve uma mensagem de erro no standard de output. Se o ficheiro foi aberto correctamente, o programa continua.

A função *fileOperations* irá utilizar, seguidamente, a função *fread* para ler a partir do ficheiro a informação relativa ao formato PBM em que a imagem se encontra (P1,P4,etc) e às dimensões, colocando-a num *buffer* temporário. Também após este passo é efectuada uma verificação que permitirá descartar ficheiros PBM que não se encontrem no formato P4. Nesta verificação, se as primeiras duas posições do *buffer* que contém o formato e as dimensões da imagem não forem, respectivamente, 'P' e '4', o programa escreve uma mensagem de erro no standard de output. Caso contrário, o programa continua. Seguidamente procede-se ao cálculo do tamanho do *buffer* que irá conter apenas os comentários, se estes existirem, e os pixeis que constituem a imagem. Depois de criado o *buffer* temporário, o programa copia, a partir do ficheiro e para o referido *buffer*, a informação que consta no ficheiro e que não corresponde ao formato e às dimensões da imagem (esta informação já se encontra num outro *buffer*). De seguida, a função *fileOperations* irá calcular o tamanho dos comentários, de modo a que possa depois armazená-los num *buffer* próprio, separando-os assim dos pixeis que constituem a imagem. Quando os comentários já se encontrarem armazenados num *buffer* próprio, a função *fileOperations* calcula o número de bytes que constituem imagem. Se a imagem contiver *newlines, carriage returns* ou espaços entre os bits que a constituem, o programa conta-os e calcula o tamanho de um *buffer* que irá conter apenas os pixeis que constituem a imagem. Seguidamente, copia os pixeis para o referido *buffer*. Antes de proceder à compressão, o programa junta num buffer final a informação do *buffer* que contém as dimensões e o formato da imagem com a informação do buffer que contém os comentários. Este *buffer* final será um dos argumentos da função que fará a compressão, bem como o *buffer* temporário que irá conter os pixeis da imagem. Finalmente, esta função irá invocar a função comprime, que irá proceder à compressão da imagem. A função *pesqDic* tem como principal objectivo permitir a pesquisa de blocos de informação presentes no dicionário. Esta função recebe como argumentos um apontador para um array que irá conter o bloco a ser procurado (valor) e um dicionário. Esta função retornará um apontador para um array de inteiros, que irá conter um bit que indica se o bloco de informação foi encontrado no dicionário (este bit toma o valor 0 se o bloco não for encontrado no dicionário e 1 se o bloco for encontrado), um bit que indica o índice do dicionário onde se encontra o bloco (se o bloco não se encontrar no dicionário, este bit assume o valor de uma posição vazia do dicionário, onde pode ser inserido o bloco) e um bit que corresponde a um elemento do bloco. A função pesDic, que efectua a pesquisa, é relativamente simples. Esta função percorre o dicionário, comparando um bloco de informação com os blocos que já lá se encontram, em busca de um bloco de informação e, caso o encontre, retorna o índice do dicionário onde o mesmo se encontra. Caso contrário, procura uma posição no dicionário que esteja vazia e retorna o índice da mesma. A função comprime tem

como principal objectivo permitir a compressão da imagem PBM. Esta função recebe como argumentos um apontador para o buffer que contém o formato e as dimensões da imagem, um apontador para o buffer que contém a informação relativa à imagem, um inteiro que representa o tamanho do buffer do formato e das dimensões, um inteiro que representa o tamanho do buffer que contém a informação relativa à imagem e dois apontadores para FILE (um para o FILE para o qual se irão escrever os dados da imagem comprimida e um FILE para o qual se irá exportar o dicionário utilizado durante a compressão da imagem). A função comprime começa por escrever no ficheiro da imagem comprimida o conteúdo do buffer que contém o formato e as dimensões. Seguidamente, é calculada a dimensão que cada bloco de informação da imagem irá ter. Se o número de elementos no array que contém a imagem for par, os blocos terão quatro elementos. Caso contrário, os blocos terão três elementos. Seguidamente a imagem é copiada, já em blocos, para uma matriz temporária. O dicionário é então inicializado (por consenso, decidiu-se que o dicionário seria inicializado com todas as suas posições a '-'). Finalmente, o programa irá proceder à compressão da imagem. Durante este processo, o programa irá ler um bloco de informação da matriz temporária, cujo comprimento foi calculado acima, e irá comparar com os blocos existentes no dicionário. Se o bloco for encontrado no dicionário, o programa escreve o índice do dicionário que lhe corresponde no ficheiro da imagem comprimida. Caso contrário, o programa irá procurar no dicionário uma posição que se encontre vazia e inserirá nessa posição o bloco, exportando o índice do dicionário agora ocupado pelo bloco para o ficheiro da imagem comprimida. Este processo repete-se tantas vezes quantas as necessárias até que toda a informação da matriz temporária tenha sido processada. Quando a compressão é concluída, o programa escreve no standard de output uma mensagem a indicar o sucesso da operação.

2 Descompressão do Ficheiro

O algoritmo de descompressão irá utilizar a informação constante na imagem comprimida e do ficheiro temporário onde foi armazenado o dicionário, com o qual se efectuou a compressão da imagem. O algoritmo de descompressão funciona do seguinte modo: Lê a informação constante no buffer que possui a informação que se encontrava no ficheiro que continha o dicionário. Constrói o dicionário a partir do buffer. Lê a informação relativa à imagem comprimida. Lê um índice. Procura no dicionário o bloco correspondente e exporta-o para o ficheiro descomprimido. Repete o processo até que toda a informação da imagem comprimida tenha sido processada.

O programa que fará a descompressão do ficheiro de imagem é constituído pelas funções fileOperations e descodifica. À semelhança do que acontece com o programa que efectua a compressão da imagem, também o descompressor possui uma função fileOperations, que irá ter um funcionamento semelhante ao da sua homónima do programa de compressão, mas que possui algumas diferenças na forma como a informação é lida. Tal como ocorre na função fileOperations do compressor, nesta função fileOperations o ficheiro comprimido é também lido com recurso à função fopen do C, efectuando-se de seguida a verificação que permite ao utilizador saber se o ficheiro comprimido foi aberto com sucesso. Também nesta função utilizada a função fread do C para ler a informação correspondente ao formato e às dimensões da imagem para um buffer temporário. Seguidamente calcula-se o número de bytes de informação que ainda constam no ficheiro, depois de ter sido lida a informação correspondente ao formato e às dimensões da imagem. Ainda à semelhança do que ocorria na função fileOperations do compressor, nesta função também a informação será posteriormente copiada para um buffer temporário. Se a imagem contiver comentários, esta função fileOperations irá isolá-los num buffer temporário próprio para o efeito e, antes da descompressão, junta-los-à à informação constante no buffer que contém o formato e as dimensões da imagem, obtendo-se assim um buffer final com o formato, as dimensões e os comentários da imagem. A informação correspondente à imagem comprimida será colocada num outro buffer, que será acedido aquando da descompressão. Após isolar em buffers todos os diferentes componentes da imagem comprimida, a função fileOperations irá proceder ao tratamento dos dados contidos no ficheiro temporário que armazena o dicionário utilizado na compressão do ficheiro. fileOperations irá ler a informação contida neste ficheiro para um buffer temporário. Seguidamente contará os newlines, carriage returns e espaços que eventualmente possam existir no ficheiro, criando depois um outro buffer onde os dados do dicionário estarão contidos, mas onde não se encontrarão quaisquer newlines, carriage returns ou espaços. Finalmente, esta função invocará a função descomprime que, tal como o nome indica, irá descomprimir a imagem. A função descomprime tem como principal funcionalidade proceder à descompressão da imagem comprimida através do nosso sistema de compressão. Esta função recebe como argumentos um apontador para o array que contém o formato, as dimensões e eventuais comentários da imagem, um apontador para o array que contém a informação da imagem comprimida (esta parte consiste nos dados gerados após a compressão dos pixeis que constituem a imagem), um apontador para o array temporário que contém os

elementos que constituem o ficheiro temporário que contém o dicionário utilizado na compressão do ficheiro, três inteiros (correspondentes, respectivamente, às dimensões dos três arrays descritos anteriormente) e um apontador para um FILE para onde irá ser escrita a informação relativa à imagem descomprimida. A função descomprime começa por escrever a informação contida no array que contém o formato, as dimensões e eventuais comentários da imagem no ficheiro que irá constituir a imagem descomprimida. Seguidamente, e à semelhança do que acontece na função comprime, esta função irá calcular o tamanho de cada bloco de informação que consta no dicionário, procedendo-se ao preenchimento do mesmo imediatamente a seguir à conclusão deste processo. A função irá, depois disto, ler os índices resultantes da compressão, que se encontram no array que contém a informação da imagem comprimida, e convertê-los-á em inteiros e colocá-los-á num array temporário. Finalmente, a função descomprime irá proceder à descompressão da imagem. Durante o processo de descompressão, o programa irá ler um índice do array temporário e, depois de o comparar com os índices dos blocos existentes no dicionário, irá exportar para o ficheiro descomprimido o bloco correspondente do dicionário. Note-se que, como o processo de descompressão utiliza o mesmo dicionário que o processo de compressão, não ocorrerão casos em que o índice a ser procurado não se encontre no dicionário. Esta operação ocorre tantas vezes quantas as necessárias até que todos os índices constantes no array temporário tenham sido processados.

3 O Nosso Ficheiro de Imagem

Como foi referido anteriormente, o ficheiro de imagem PBM que iremos utilizar encontra-se no formato PBM P4. Assim, os primeiros dois caracteres correspondem ao formato da imagem, seguidamente existe um espaço em branco e depois podem ser encontradas as dimensões da imagem, separadas por um espaço em branco. Finalmente, podem ser encontrados os pixeis que constituem a imagem separados do resto dos componentes da imagem por um espaço em branco. Note-se que os bits que representam os pixeis brancos e pretos da imagem estão agrupados num único número decimal, cuja representação em binário será de 8 bits. A imagem utilizada não contém comentários. Porém, tanto o compressor como o descompressor prevêem o caso em que a imagem contém comentários. A informação contida em cada ficheiro de imagem PBM encontra-se codificada através de zeros e uns, que representam os pixeis brancos e pretos da imagem. Assim, o alfabeto considerado para este trabalho será: [] = {0,1}. A imagem que será utilizada é constituída por 424 bits, dos quais 292 são zeros e 132 são uns. Assim, a probabilidade de ser lido um bit da imagem que corresponde a um zero é de (292 / 424) ou, simplificando a fracção, (73/106). A probabilidade de ser lido um bit da imagem que corresponde a um 1 é de (132 / 424) ou, simplificando a fracção, (33/106). A Entropia é a medida da quantidade de informação necessária, em média, para descrever uma variável aleatória. Neste caso, o cálculo da Entropia permitir-nos-á medir, em média, a quantidade de informação necessária para descrever a imagem, ou seja, quantos bits são necessários, em média, para descrever a imagem que pretendemos comprimir. A Entropia é dada por:

$H(X) = \sum p(x) log_2 p(x)$

A Entropia da imagem original é dada por:

$H(X) = -(\frac{73}{106}) log_2(\frac{73}{106}) - (\frac{73}{106}) log_2(\frac{73}{106}) = 0,524 + 0.371 = 0,895 bits$

A taxa de compressão de dados a permite quantificar a redução no tamanho da representação de dados após a aplicação de um algoritmo de compressão de dados. A taxa de compressão é dada por:

Taxa de Compressão = 1 - Quociente de Compressão

Quociente de Compressão = Tamanho da Imagem Comprimida / Tamanho da Imagem Original

A imagem original possuía 424 bits. A imagem comprimida possui 160 bits. A quociente de compressão é:

$Quociente \quad de \quad compressão = \frac{160}{424} \approx 37.7\%$

A Taxa de Compressão atingida com o nosso algoritmo é de:

$Taxa \quad de \quad compressão = \frac{160}{424} \approx 37.7\%$

Sabe-se que a taxa de compressão deverá ser sempre menor que a Entropia e, se o algoritmo de compressão for altamente eficiente, esta taxa deverá ter um valor que seja o mais próximo da Entropia possível. Dado que a taxa de compressão obtida é inferior à entropia, ainda é possível melhorar mais.

Parte II

Capacidade de Canais Discretos sem Memória

Como foi referido anteriormente, o algoritmo de Blahut permite calcular a capacidade de um canal de transmissão de dados. Note-se que este algoritmo foi desenvolvido com o objectivo de permitir calcular a referida capacidade, uma vez que, no geral, não existe um modo explícito para o cálculo da mesma.

O funcionamento do algoritmo de Blahut processa-se do seguinte modo:

1. Inicializa Px com valores uniformes 1/N, em que N é o número de símbolos à entrada do canal.
2. Calcula Cj (capacidade).
3. Calcula o limite inferior.
4. Calcula o limite superior.
5. Se a diferença entre o limite superior e o limite inferior for menor que a margem de precisão:
   (a) A capacidade do canal, C, será igual ao valor do limite inferior.
   (b) Sai do algoritmo.
6. Senão:
   (a) Calcula um novo Pj, que será $P_j = P_j * (C_j * \frac{C_j}{\sum P_j * C_j})$
7. Volta ao passo 2.

Para testar o algoritmo foram utilizados os seguintes canais:

- Canal Binário Inversor
- Canal Binário Simétrico
- Canal Binário com Perdas
- Máquina de Escrever Ruidosa

Exemplo 1: Canal Binário Inversor

O Canal Binário Inversor é semelhante ao Canal Binário sem perdas, porém o símbolo transmitido é o inverso do símbolo recebido.

$$C = max_{P_j} I(X;Y) = max_{P_J}(H(Y) - H(X|Y)) = 1bit$$

Porque $H(X|Y) = 0bit$ e $H(Y) = 1bit$.

O máximo é atingido quando $P(x) = 0.5$.

Para calcular a capacidade do canal com recurso à nossa rotina em Python foram utilizados os seguintes dados:

- Número de símbolos de entrada: 2
- Número de símbolos de saída: 2
- Margem de Precisão: 10^{-7}
- Matriz de Transição: $M_t = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

A aplicação retornou o seguinte:

- Capacidade do Canal = 1.0bits
- Probabilidades de Entrada $P(x) = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}$

Note-se que a matriz que possui as probabilidades de saída é a matriz que permite atingir a capacidade do canal. Portanto, o algoritmo está a calcular correctamente a capacidade do canal e a matriz que a permite atingir.

Exemplo 2: Canal Binário Simétrico

A capacidade do Canal Binário Simétrico é dada por:

$$C = max_{P_j} I(X;Y) = 1 - H(P_e)$$

Em que P_e corresponde à probabilidade de ocorrer erro na transmissão de informação.

Para calcular a capacidade do canal com recurso à nossa rotina em Python foram utilizados os seguintes dados:

- Número de símbolos de entrada: 2
- Número de símbolos de saída: 2
- Margem de Precisão: 10^{-7}
- Matriz de Transição: $M_t = \begin{bmatrix} 0.1 & 0.1 \\ 0.9 & 0.9 \end{bmatrix}$

A aplicação retornou o seguinte:

- Capacidade do Canal = 0.531004406411bits
- Probabilidades de Entrada $P(x) = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}$

Note-se que a matriz que possui as probabilidades de saída é a matriz que permite atingir a capacidade do canal. Nos cálculos que efectuámos sem recurso à nossa rotina obtivemos os seguintes valores:

- $H(P_e) = -0.9 * log_2(0.9) - 0.1 * log_2(0.1) \approx 0.47 bits$
- $C = 1 - H(P_e) = 1 - 0.47 = 0.53 bits$

Portanto, o algoritmo está a calcular correctamente a capacidade do canal e a matriz que a permite atingir.

Exemplo 3: Canal Binário com Perdas

A capacidade do Canal Binário Com Perdas é dada por:

$$C = max_{P_j} I(X;Y) = 1 - P_e$$

Em que P_e corresponde à probabilidade de ocorrer um erro na transmissão de informação.

O máximo é atingido quando a probabilidade $P(x)$ for igual a 0.5.

Para calcular a capacidade do canal com recurso à nossa rotina em Python foram utilizados os seguintes dados:

- Número de símbolos de entrada: 2
- Número de símbolos de saída: 3
- Margem de Precisão: 10^{-7}
- Matriz de Transição: $M_t = \begin{bmatrix} 0.8 & 0.0 \\ 0.2 & 0.2 \\ 0.0 & 0.8 \end{bmatrix}$

A aplicação retornou o seguinte:

- Capacidade do Canal = 0.8bits
- Probabilidades de Entrada $P(x) = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}$

Note-se que a matriz que possui as probabilidades de saída é a matriz que permite atingir a capacidade do canal. Nos cálculos que efectuámos sem recurso à nossa rotina obtivemos os seguintes valores:

- $C = 1 - 0.2 = 0.8 bits$

Portanto, o algoritmo está a calcular correctamente a capacidade do canal e a matriz que a permite atingir.

Exemplo 4: Máquina de Escrever Ruidosa Nota: Este exemplo foi elaborado para um alfabeto que contém apenas quatro símbolos.

A capacidade do Canal "Máquina de Escrever Ruidosa" é dada por:

$$C = max_{P_j} I(X;Y) = log_2(\frac{Número\,de\,Simbolos\,do\,alfabeto}{2})$$

O máximo é atingido quando a probabilidade p(x) for igual a $\frac{1}{Número\,de\,Simbolos\,do\,alfabeto}$.

Para calcular a capacidade do canal com recurso à nossa rotina em Python foram utilizados os seguintes dados:

- Número de símbolos de entrada: 2
- Número de símbolos de saída: 3
- Margem de Precisão: 10^{-7}
- Matriz de Transição: $\begin{bmatrix} 0.5 & 0 & 0 & 0.5 \\ 0.5 & 0.5 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0.5 & 0.5 \end{bmatrix}$

A aplicação retornou o seguinte:

- Capacidade do Canal = 1bits
- Probabilidades de Entrada $P(x) = \begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix}$

Note-se que a matriz que possui as probabilidades de saída é a matriz que permite atingir a capacidade do canal. Nos cálculos que efectuámos sem recurso à nossa rotina obtivemos os seguintes valores:

- $C = log_2(\frac{4}{2}) = 1bits$

Portanto, o algoritmo está a calcular correctamente a capacidade do canal e a matriz que a permite atingir.

Parte III
Conclusão

A elaboração deste trabalho permitiu-nos construir um algoritmo que visa proceder à compressão de um ficheiro de imagem no formato PBM P4 com recurso à utilização de dicionários. Numa fase mais inicial do trabalho havia sido planeada a compressão do ficheiro com recurso ao algoritmo de Huffman Adaptativo. Porém, a implementação deste algoritmo provou ser complexa e tomou-nos bastante tempo útil, tendo por fim sido abandonada. Seguidamente, foi planeada a implementação de um dos algoritmos Lempel-Ziv. Porém, também essa ideia foi abandonada. Finalmente, encontrou-se um meio termo. O algoritmo final, à semelhança dos algoritmos Lempel-Ziv, utiliza dicionários, onde durante a compressão serão armazenados blocos de informação. Para efectuar a descompressão de um ficheiro de imagem, este algoritmo irá aceder ao dicionário utilizado durante a compressão (que após a mesma foi exportado para um ficheiro temporário) e, após proceder à sua recuperação a partir do ficheiro temporário, irá proceder à descompressão da imagem. Note-se que, com este algoritmo, a compressão está sujeita a perdas de informação. Tal perda de informação ocorrerá se a imagem tiver algum newline, carriage return ou espaço entre os píxeis que a constituem. Teoricamente, se estes caracteres não existirem, a compressão não estará sujeita a perdas de informação. O algoritmo elaborado é bastante ineficiente, tendo sido projectado apenas para imagens PBM de pequenas dimensões. Os custos em termos de memória e os tempos de desempenho do algoritmo aumentam significativamente quanto maior for a imagem. Conclui-se que, para um melhor desempenho de compressão deste tipo de imagem, é aconselhável usar um algoritmo especializado para o efeito ou um algoritmo de compressão Lempel-Ziv, no caso de não existir um algoritmo especializado em imagens PBM. Tal recomendação prende-se com o facto dos algoritmos Lempel-Ziv aliarem um bom desempenho a uma boa taxa de compressão, permitindo a obtenção de melhores resultados.

Em relação ao trabalho da computação das capacidades dos canais, conclui-se que este algoritmo facilita o calculo das estimativas da capacidade, utilizando os limites superior e inferior pelas funções de $P(x)$, e também da distribuição de probabilidade de entrada que permite alcançar a capacidade.

4 Referências

Livro:

Cover, T. M. e Thomas, J. A.. 2006. Elements of Information Theory, 2nd Edition, Wiley.

Fontes da Internet:

Poskanzer, J. 1988. The PBM Format (Online). Acedido em: 1-12-2011.
Disponível em: http://netpbm.sourceforge.net/doc/pbm.html

Fernandes, M. 2009. Compressão de Dados (Online). Acedido em: 29-11-2011.
Disponível em: http://pt.scribd.com/doc/23087326/Compressao-de-dados

Wikipédia, Entropia (teoria da informação) (Online). Acedido em: 20-11-2011.
Disponível em: http://pt.wikipedia.org/wiki/Entropia_(teoria_da_informa%C3%A7%C3%A3o)

Figueiredo, M. 2007. Elementos de Teoria da Informação (Online). Acedido em: 20-11-2011.
Disponível em: http://www.lx.it.pt/~mtf/teoria_informacao.pdf

Blahut, R. 1972. Computation of Channel Capacity and Rate-distortion Functions (Online). Acedido em: 05-12-2011. Disponível em: http://savannah.gatech.edu/people/lthames/dataStore/ECE/InfoTheory/0460blah.pdf

Wikipédia, Channel Capacity (Online). Acedido em: 15-12-2011.
Disponível em: http://en.wikipedia.org/wiki/Channel_capacity

Roweis, S. 2005. CSC310 – Information Theory, Lecture 13: Channel Capacity (Online). Acedido em: 20-12-2011. Disponível em: http://www.cs.nyu.edu/~roweis/csc310-2005/notes/lec13x.pdf