ICC 2 – Trabalho 06
Prof. Rodrigo Mello

Estagiários PAE

Martha Dais Ferreira
Fausto Guzzo da Costa

Data Máxima para Perguntas: 24/10/2014

Data de Entrega do Trabalho: 29/10/2014

Fórum para Envio de Perguntas: https://groups.google.com/d/forum/icc2-2014

Descrição

Considere um arquivo de áudio gravado em formato binário. Esse arquivo contém amplitudes do áudio gravadas em blocos de bytes. As amplitudes são resultantes de variações no diafragma do microfone. Ao gravar o áudio, conforme falamos no microfone, há variações no diafragma, que são codificadas em números a cada instante, e gravados em um arquivo binário.

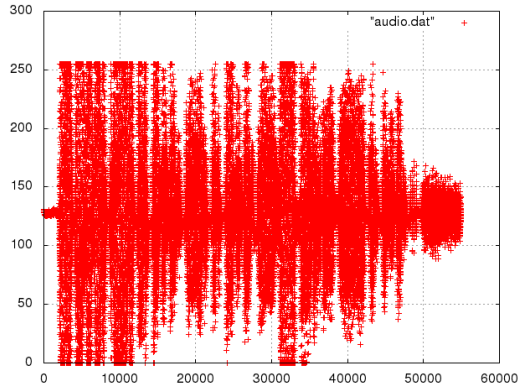
O valor da pressão de nossa voz sobre o diafragma do microfone é medido por um *unsigned byte*. Dessa maneira há 8 bits para representar a situação do diafragma em dado instante de tempo. Cada valor de amplitude varia de 0 até 255, pois ele é representado por um unsigned byte (ou seja, 1 byte sem sinal), em que há 8 bits que podem ser usados: logo $2^8 = 256$ possíveis valores. Como começa em 0 vai até 255.

O valor 127 (valor central) significa que o diafragma está em repouso, ou seja, ninguém está falando. Valores abaixo de 127 e acima de 127 significam variações do diafragma impostas por nossa voz.

Ao ler esse arquivo binário, **um byte por vez**, você teria valores como:

127 128 180 182 187 190 183 172 171 169 127 128

Ao construir um gráfico (plotar) esse arquivo com inteiros no formato texto (no Linux, pode-se usar o comando gnuplot para plotar) temos algo na forma:



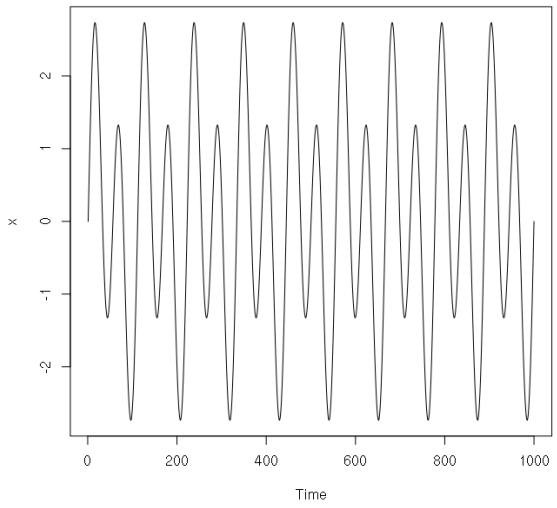
Em que temos o tempo no eixo x e a amplitude do diafragma (1 byte) no eixo y. Como o arquivo produzido foi gravado com 8000 Hertz, há 8000 observações do diafragma por segundo de gravação. Esse número define a amostragem no tempo. Na Figura acima há cerca de 55000 observações no total, cada uma corresponde a 1 byte. Dividindo as 55000 observações por 8000 (amostragem no tempo) é possível saber que esse arquivo representa 6,8 segundos de gravação, ou seja, foram coletadas 8000 variações do diafragma do microfone para cada segundo.

A sua tarefa neste trabalho compreende, essencialmente, a implementação da **Transformada Discreta do Cosseno**. Para que serve essa transformada? Vejamos.

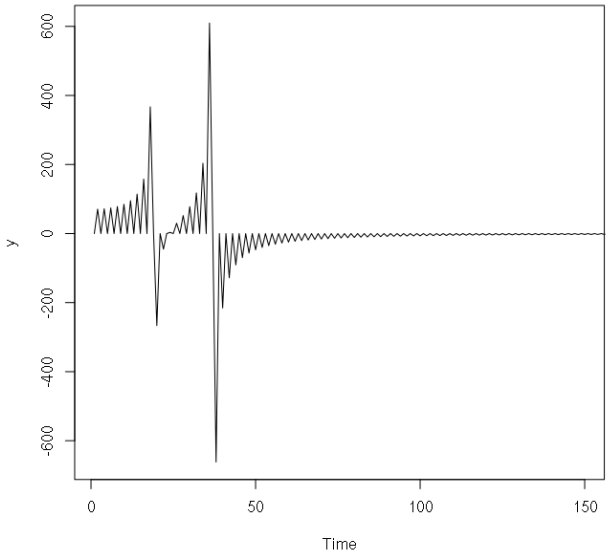
Suponha o comando abaixo que gera um sinal de áudio composto por duas funções senoidais (ele pode ser executado na linguagem R, mais detalhes em http://r-project.org):

*x = sin(2*pi*seq(0,9,len=1000)) + 2*sin(4*pi*seq(0,9,len=1000))*

Ao plotar *x* obtemos o seguinte gráfico:



Em que o eixo das abscissas corresponde ao tempo e o das ordenadas a valores de amplitude. Após aplicar a **Transformada Discreta do Cosseno** obtemos um gráfico como segue:



O que significa esse gráfico? Observe que há dois picos superiores e dois inferiores. Eles nos indicam que há duas senóides que compõem este sinal de áudio. O gráfico foi gerado como segue na linguagem R:

```
install.packages("dtt")
require(dtt)
dados_transformados = dct(x, variant=2, inverted=F);
ts.plot(dados_transformados);
```

Sendo assim, suponha que desejamos eliminar a senóide menos importante, ela ocorre no intervalo de 1 até aproximadamente 22 no tempo (em R começamos em 1 e não em 0). Na linguagem R executaríamos os comandos abaixo:

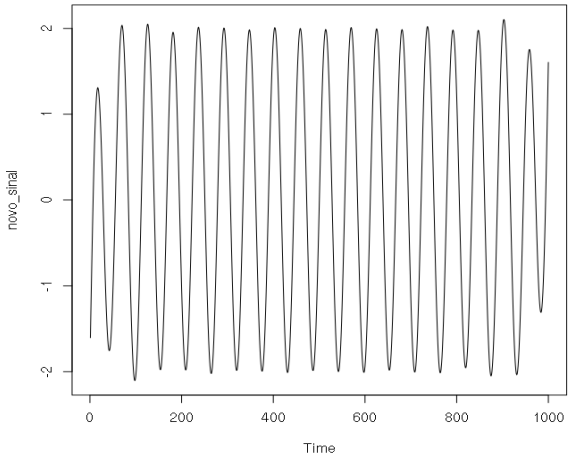
Aplicando a **Transformada Discreta do Cosseno**:

```
dados_transformados[1:22] = 0;
```

Em seguida, poderíamos executar a **Transformada Inversa** (segue exemplo na linguagem R) como segue:

```
novo_sinal = dct(dados_transformados, variant=2, inverted=T)
```

E obteríamos um sinal tal como o abaixo:

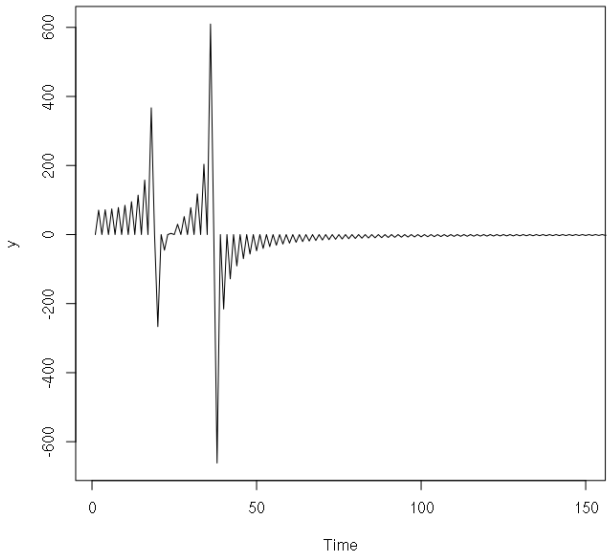


Observe que este novo sinal não tem a influência de uma das senóides, no entanto ele tem alguns "artefatos" que fazem com que o sinal não fique uma senóide perfeita.

Podemos usar a **Tranformada Discreta do Cosseno** para encontrar quantas senóides participam de um sinal de áudio bem como a importância de cada senóide. Neste caso ilustrado há duas senóides, aquela que produz o pico mais máximo e mínimo é a mais importante. Depois seria a senóide seguinte e assim sucessivamente.

Neste trabalho você deverá estudar procurar por referências para a Transformada Discreta do Cosseno na Variante II (ou DCT-II, veja mais em

http://en.wikipedia.org/wiki/Discrete_cosine_transform) e implementá-la. Em seguida, deverá ordenar, se maneira crescente, os valores resultantes da transformada e imprimir apenas os K maiores valores com duas casas decimais de precisão. Por exemplo, após aplicar a transformada em nosso exemplo obtivemos o gráfico abaixo:



Suponha que K = 5, logo imprimiríamos:

609.72
367.03
203.58
157.75
117.69

Dicas

1) Para gerar um arquivo de áudio binário no Linux utilize:

arecord -t raw -c 1 -r 8000 test.raw

2) Para ouvir o arquivo de áudio binário no Linux utilize:

aplay -t raw -r 8000 -c 1 -f u8 test.raw

Há outros aplicativos como o Audacity e o VLC Media Player que funcionam para outros sistemas operacionais além do Linux e que podem ajudar a gravar áudios e a tocá-los

3) Posso utilizar o código de um dos algoritmos de ordenação visto em aula? Sim. Isso contará na detecção de plágio? Não. Iremos informar o MOSS que esse trecho não deve ser considerado como plágio.

Motivação

Essa transformada é empregada em vários algoritmos. Por exemplo, o algoritmo que produz arquivos MP3 e o algoritmo que produz imagens JPEG utilizam a Transformada Discreta do Cosseno. O objetivo é o de encontrar as senoidais mais importantes em um áudio e ou em uma imagem e zerar as demais. Depois da **transformada inversa do cosseno**, podemos obter um sinal de áudio mais simples e ou uma imagem mais simples, que podem ser compactados de forma muito maior.

Formato para os Casos de Teste

Formato da entrada:
<nome do arquivo binário de áudio>
<número K de valores a serem impressos>

Formato da saída:
<valor 1>
<valor 2>
...
<valor K>

Exemplo de Caso de Teste

Entrada fornecida:
arquivo.raw
5

Saída esperada:
609.72
367.03
203.58
157.75
117.69

Informações Importantes

1) Um dos objetivos da disciplina de ICC2 é o aprendizado individual dos conceitos de programação. A principal evidência desse aprendizado está nos trabalhos, que são individuais neste curso. Você deverá desenvolver seu trabalho sem copiar trechos de código de outros alunos ou da Internet, nem codificar em conjunto. Portanto, compartilhem ideias, soluções, modos de resolver o problema, mas não o código.

1.1) O plágio vai contra o código de ética da USP;

1.2) Quando autores e copiadores combinam, estão ludibriando o sistema de avaliação, por isso serão utilizadas ferramentas de análise plágio tais como o MOSS (http://moss.stanford.edu/);

1.3) O trabalho em grupo e a cooperação entre colegas é em geral benéfico e útil ao aprendizado. Para ajudar um colega você pode lhe explicar estratégias e ideias. Por exemplo, pode explicar que é preciso usar dois loops para processar os dados, ou que para poupar memória basta usar uma certa estrutura de dados, etc. O que você não deve fazer é mostrar o seu código. Mostrar/compartilhar o código pode prejudicar o aprendizado de seu colega:

1.3.1) Depois seu colega ver seu código, será muito mais difícil para ele imaginar uma solução original e própria;

1.3.2) O seu colega não entenderá realmente o problema: a compreensão passa pela prática da codificação e não pela imitação/cópia.

1.4) Um colega que tenha visto a sua solução pode eventualmente divulgá-la a outros colegas, deixando você numa situação muito complicada, por tabela;

1.5) O texto acima foi baseado e adaptado da página http://www.ime.usp.br/~mac2166/plagio/, da qual recomendo a leitura completa.

2) Todos os códigos fontes serão comparados por um (ou mais) sistema(s) de detecção de plágio, e os trabalhos com alta similaridade detectada terão suas notas zeradas, tanto aqueles relativos ao código de origem quanto do código copiado. A detecção de plágio será reportada à Seção de Graduação para providências administrativas;
3) A avaliação incluirá a porcentagem de acertos verificada pelo Sistema de Submissão de Trabalhos e também a análise do seu código, incluindo identação, comentários, bom uso da memória e práticas de programação. Portanto faça seu código com cuidado, da melhor forma possível.

Sobre o sistema de submissão:

1. Seu código deverá incluir arquivo fonte .c .h e Makefile – todos os arquivos deverão obrigatoriamente conter no início um comentário com seu nome, número USP, turma e data da entrega do trabalho;

2. A data/hora de entrega do trabalho é aquela estipulada no sistema. Trabalhos entregues por email NÃO serão aceitos, mesmo que dentro da data/hora estipulada. Faça seu trabalho com antecedência para evitar entregar em cima da hora e ter problemas de submissão;

2.1. A submissão é de responsabilidade do aluno, e os problemas comuns à entrega próxima ao fechamento do sistema também. Portanto: problemas de acesso à rede não serão aceitos como desculpa para entrega por email ou fora do prazo;

3. A compilação e execução do código é feita no sistema pelos comandos:

make all
make run

4. A saída do seu programa deve ser exatamente igual à saída esperada, incluindo: espaços em branco, quebras de linha e precisão decimal;

5. Há um limite em segundos para a execução dos casos de teste e um limite de memória total para ser utilizado. Você deverá gerenciar bem o tempo de execução e o espaço ocupado para que seu programa fique dentro desses limites, para evitar uso excessivo, pois o sistema irá invalidar o caso em que o limite foi excedido;

6. Ao enviar, aguarde sem recarregar a página nem pressionar ESC, para obter a resposta. Caso demore mais do que 2 minutos para dar uma resposta, feche e abra novamente a página do servidor. Verifique se seu programa tem algum problema de entrada de dados (ou se tem algum loop infinito ou parada para pressionamento de tecla). Caso não tenha, aguarde alguns minutos e tente novamente;

7. O erro de “Violação de Memória” significa acesso indevido a arranjo ou arquivo. Use compilação com g e o programa valgrind para tentar detectar a linha de código com erro.

7.1. Exemplo 1 de violação de memória:

```
int **mat = (int **)malloc(sizeof(int *) * 3);
mat[0] = (int *)malloc(sizeof(int)*10);
for (i = 1; i < 3; i++) {
   free(mat[i]);
}
// apenas a posicao 0 de mat foi alocada as outras nao
// portanto esta liberando regiao nao alocada
// gerando violacao de memoria
```

7.2) Exemplo 2 de violação de memória:

```
int B[5] = {5, 6, 7, 8, 9};
int N = 5;
int *A = malloc(N*sizeof(int));
int j = 0, i = 1; // 'j' inicializado em 0, 'i' inicializado em 1
while (j < N){
A[i] = B[j]; // 'j' e 'i' sao indices diferentes
j++;         // quando j = (N1) i = (N1)+1 e
i++;
     // havera escrita indevida em A
}
```