Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Algoritmos e Estruturas de Dados II
1º Semestre de 2011
Prof. Gisele L. Pappa

Trabalho Prático II - Compressor de Arquivos

10 pontos (+1 desafio)

Introdução

Todo mundo um dia já deve ter se perguntado como um compressor de arquivos funciona. Pois bem, não há mágica alguma, tanto que esta será sua missão neste trabalho. Por sorte existe um algoritmo que necessita apenas de duas estruturas de dados simples e que você já conhece: Listas e Árvores Binárias. Este algoritmo é conhecido como código de Huffman (ou simplesmente algoritmo de Huffman). Logicamente implementar o descompressor também será sua tarefa, caso contrário não haveria graça alguma.

Descrição do Algoritmo

Esta seção apresenta uma descrição simples do algoritmo de Huffman. Você deve consultar as referências do final deste documento para mais detalhes.
Considere um arquivo sendo formado por um conjunto de caracteres, os quais possuem representação binária de tamanho fixo (por simplicidade, será assumido aqui que 1 caracter = k bits).
O truque básico consiste em perceber que em muitos casos adotar uma representação como essa gera redundância e faz com que mais bits do que o necessário sejam utilizados para representar uma informação. Considere o exemplo de um arquivo de cem mil caracteres formado do alfabeto (a, b, c, d, e, f), sendo cada caracter representado por um código de tamanho fixo com k = 3 bits. A Tabela abaixo apresenta a frequência de cada caracter no arquivo:

Tabela 1: Exemplo de um arquivo e representações de comprimento fixo e variável

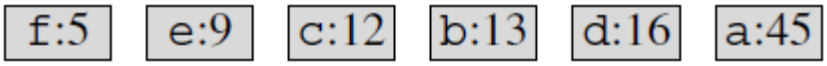
Caracter	Frequência (em milhares)	Representação fixa	Representação variável
a	45	000	0
b	13	001	101
c	12	010	100
d	16	011	111
e	9	100	1101
f	5	101	1100

De acordo com estas informações trezentos mil bits serão necessários para representar esse arquivo. Agora considere que ao invés disso fosse utilizada a representação de comprimento variável mostrada na última coluna da tabela acima. Então apenas 224 mil bits seriam necessários para representar a mesma informação.

O código de Huffman mostra como obter um código binário de comprimento variável que pode resultar em até 90% de economia de bits. Antes de apresentar o algoritmo, algumas definições são necessárias. Seja **C** o conjunto dos caracteres que formam um arquivo **A** que se deseja comprimir. Para cada **c** em **C**, considere freq(**c**) a frequência com que **c** apare em **A**. Considere ainda **L** sendo uma lista de árvores binárias ordenada pela frequência que cada nó raiz representa. O algoritmo a seguir apresenta uma ilustração do procedimento que deve ser implementado. A explicação é dada através de um exemplo.

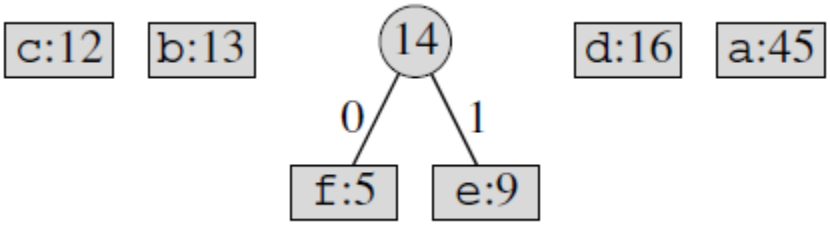
HUFFMAN(C)
1) n = número de elementos em C;
2) Inicie L, de forma que cada um de seus elementos represente um elemento de C. L deve ter comprimento n e estar ordenada em ordem crescente pela frequência de cada um de seus nós raiz;
3) Para i = 1 até n - 1 Faça
4) z = crie num novo nó de árvore binária;
5) x = remova menor elemento de L;
6) y = remova o menor elemento de L;
7) z.equerda = x;
8) z.direita = y;
9) freq(z) = freq(x) + freq(y);
10) Insira z em L de forma ordenada;
11) Retorne o menor elemento de L;

Seja C o conjunto dado na tabela da página anterior (duas primeiras colunas). A linha 1 faz n = 6; A linha dois cria uma lista L como a figura a seguir:

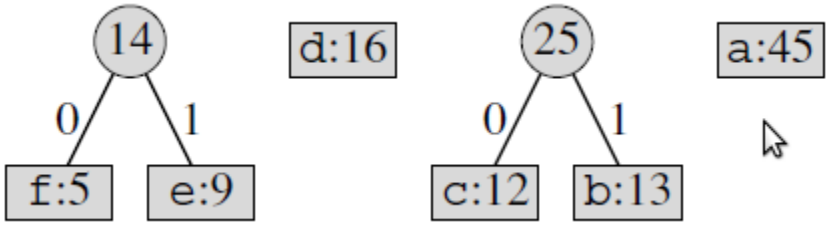


Os próximos passos do algoritmo consistem em modificar esta árvore de forma a construir o código de Huffman. *Em cada iteração associe 1 ao nó esquerdo e 0 ao direito*.

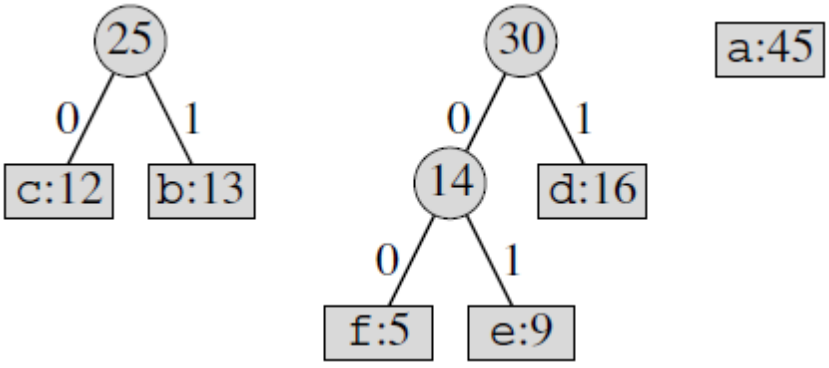
Quanto i = 1



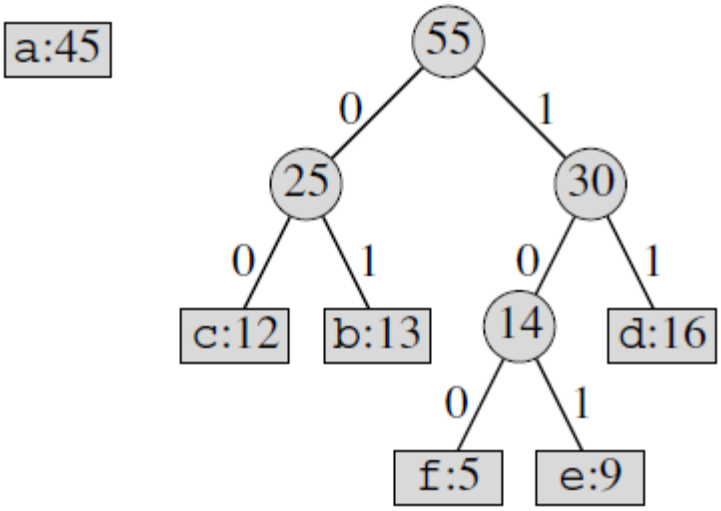
Quando i = 2



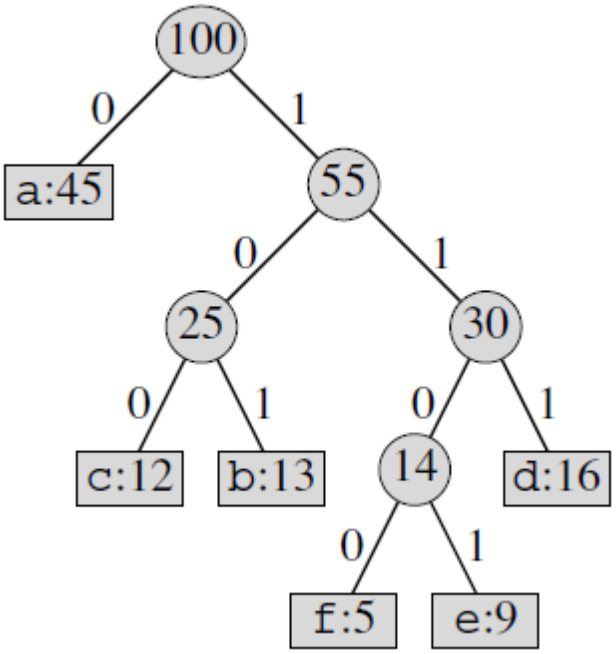
Quando i = 3



Quando i = 4



Quando i = 5



Agora repare que percorrendo a árvore de baixo para cima a partir das folhas os códigos apresentados na Tabela 1 são encontrados. Além disso é importante ressaltar que a raiz da última árvore deve representar o total de caracteres do arquivo (no exemplo, 100 é referente a 100 mil).

Dicas

O arquivo a ser compactado deverá ser lido byte a byte, isto é, deve-se considerar a existência de no máximo 256 símbolos. Seu programa deverá realizar uma primeira leitura
do arquivo de modo a determinar o número de ocorrências de cada símbolo e, a partir desta
informação, construir uma codificação de Huffman para o conjunto de dados. Uma segunda passada pelo arquivo será necessária para compressão do mesmo. De modo a permitir a decodificação do arquivo, uma tabela contendo o número de ocorrências de cada símbolo deverá ser armazenada no início do arquivo comprimido (antes dos dados codificados).
Dessa forma, o decodificador poderá recuperar do arquivo a frequência de cada um dos símbolos e, consequentemente, reconstruir a mesma codificação de Huffman utilizada na compressão. Uma vez restaurada a codificação utilizada, a descompressão das informações pode ser realizada facilmente.

Entrada

Você deve implementar dois módulos. Um para compressão e outro para descompressão de arquivos. O módulo de compressão deverá se chamar comprima (no Linux) ou comprima.exe (no Windows). O módulo de descompressão deverá se chamar descomprima (no Linux) ou descomprima.exe (no Windows).
Os módulos serão executados por linha de comando, da seguinte maneira:

- ./comprima <file_out> <file_in>

Onde <file_in> é o nome do arquivo a ser comprimido e <file_out> o nome do arquivo que será criado com o resultado da compressão;

- ./descomprima <file_out> <file_in>

Onde <file_in> é o nome do arquivo com o conteúdo comprimido e <file_out> é o nome do arquivo onde o resultado da descompressão deverá ser armazenado.

ATENÇÂO: Seu compressor/descompressor deve ser capaz de lidar com arquivos de até 1GB. Tome cuidado na hora de escolher os tipos de variáveis para não haver *overflow*.

Saída

As saídas de cada módulo são o arquivo comprimido para módulo de compressão e o arquivo descomprimido para o módulo de descompressão. Em ambos os casos não é necessário imprimir nada na tela durate/após o processamento.

ATENÇÃO: O conteúdo do arquivo resultante do módulo de descompressão deverá ser **exatamente igual** ao conteúdo do arquivo original que foi comprimido.

Desafio (1pt)

ATENÇÃO: Os desafios são opcionais apenas para os alunos das engenharias (1pt extra). Alunos da computação, sistemas de informação e matemática computacional devem considerar o desafio 1 como parte integral do trabalho, enquanto o desafio 2 é opcional e vale 1pt extra.

Dois desafios são lançados neste trabalho:

1. A eficiência do algoritmo de Huffman pode ser significativamente melhorada se ao invés de uma Lista você utilizar uma fila de prioridades. Implemente também esta versão do algoritmo e faça uma comparação do desempenho das duas abordagens;
2. Faça com que seu compressor seja capaz de comprimir/descomprimir uma árvore de diretórios. Todas as pastas, subpastas e arquivos dentro destas deverão ser comprimidos em apenas um arquivo.

O que deve ser entregue

- Código;
- Documentação do trabalho. Entre outras coisas, a documentação deve conter:
    1. Introdução: descrição do problema a ser resolvido e visão geral sobre o funcionamento do programa.
    2. Implementação: descrição sobre a implementação do programa. Deve ser detalhada a estrutura de dados utilizada (de preferência com diagramas ilustrativos), o funcionamento das principais funções e procedimentos utilizados, o formato de entrada e saída de dados, compilador utilizado, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado.
    3. Estudo de Complexidade: estudo da complexidade do tempo de execução dos procedimentos implementados e do programa como um todo (notação O). **Com exceção do desafio, você pode determinar apenas a ordem de complexidade**. Não é necessário determinar a função.
    4. **Testes**: descrição dos testes realizados e listagem da saída (não edite os resultados). Nesta parte você deve realizar uma avaliação de desempenho do seu compressor/ descompressor. Escolha pelo menos 10 arquivos de tamanho variado e monte uma tabela com a taxa de compressão de cada um deles. Além disso, para cada um destes arquivos apresente o tempo que foi gasto nas operações e a configuração da máquina usada. Preferencialmente apresente a medida de tempo como uma média de várias (pelo menos 10) execuções para o mesmo arquivo para que este seja um valor mais confiável.
    5. Conclusão: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
    6. Bibliografia: bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites

da Internet, se for o caso. Atente que a cópia de qualquer trecho de código da Internet deverá estar devidamente citada e extremamente bem motivada.

Considerações importantes

O trabalho tem o valor de 10 pontos. A pontuação é uma função da sua implementação e da sua documentação, logo, não se esqueça de fazer uma documentação criteriosa.
A data de entrega do trabalho é dia **23/05/2011, até às 23:59**. Isso significa que, submissões a partir de 00:01:00 do dia 24/05/2011 serão consideradas com 1 dia de atraso (*hard deadline!*).

Preste atenção nos seguinte pontos. Eles serão importantes na correção de seu trabalho:

- A especificação pode sofrer alterações caso surja necessidade. Fique atento às aulas e ao learnloop. É de suma importância que logo após o lançamento da especificacão, suas dúvidas sobre a mesma sejam postadas o mais rápido possível nos auxiliando a identificar requisitos não especificados corretamente;
- Dê preferência ao uso de Linux. Seu trabalho deverá ser **compilável** em Linux para ser devidamente corrigível pelos monitores. O executável pode ser voltado para qualquer plataforma, contudo, deve estar especificado na documentação;
- O trabalho deve ser implementado em C e não deve ser usada nenhuma biblioteca fora do padrão ANSI-C;
- Comece a fazer este trabalho logo (AGORA), enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar;
- O trabalho é individual;
- Caso haja suspeita de que seu trabalho foi copiado, comprado, doado, etc você será avaliado de acordo e poderá ser convidado ou se convidar a explicar-se sobre os fatores que geraram a suspeita;
- Programe de forma organizada modularizando o código de forma adequada. Comente seu código extensivamente;
- Procure dar nome significativo para suas variáveis;
- Para cada *malloc* deve haver um *free*. Vazamento de memória em excesso será um critério de penalização de sua nota. Utilize *valgrind* (http://valgrind.org) para lhe auxiliar nesta tarefa;
- A submissão será feita pelo Learnloop. Faça um zip ou similar com todos os arquivos, inclusive documentação;
- **Penalização por atraso: (2^(d) – 1) pontos, onde *d* é o número de dias de atraso;**

Referências

Cormen , T., Leiserson, C, Rivest R., Stein, C. Introduction to Algorithms, Second Edition, MIT Press, 2001.
Drozdek, A. Estruturas de dados e algoritmos em C++, Thompson Pioneira, 2002.