EP 2: Aplicação de estruturas de dados e algoritmos relacionados

Entrega: 25/02/2018, até às 23:55h

Tópicos de Programação – IME – USP

Verão/2018

Prof. Arthur Casals (arthur.casals at usp.br)

1. Introdução

Uma das aplicações de código de Huffman envolve a armazenagem temporária de mensagens em cenários envolvendo comunicação volumosa entre diferentes partes. Imagine, por exemplo, um sistema similar ao e-mail, que é responsável por receber mensagens de diversas pessoas e encaminhá-las a seus destinatários. Em ocasiões especiais (como feriados), é possível que o sistema não seja capaz de enviar todas as mensagens no momento em que são recebidas. Limitações como largura de banda de Internet ou capacidade de processamento do *hardware*, por exemplo, podem determinar que o sistema seja capaz de enviar apenas 5 mensagens por minuto, enquanto são recebidas 20 mensagens por minuto. Dessa forma, as mensagens que não são imediatamente enviadas têm de ser armazenadas e enfileiradas para envio posterior.

Uma das outras limitações deste sistema envolve espaço de armazenamento: por motivos relacionados a custos financeiros, o *hardware* utilizado pelo sistema possui pouco espaço disponível para armazenar mensagens enfileiradas. De forma a se mitigar esta limitação, implementou-se um algoritmo de compactação que utiliza código de Huffman para compactar cada mensagem que é recebida pelo sistema. Além disso, uma vez que o código de Huffman depende da frequência de ocorrência das letras em cada mensagem, implementou-se uma estrutura que possibilitasse não só a armazenagem da mensagem, mas também a tabela de frequência de caracteres associada a cada mensagem.

Pouco antes do Carnaval de 2018, Alice - uma programadora matriculada em um curso de verão na USP - tomou conhecimento da existência desse sistema e ficou bastante interessada em seu funcionamento. Por esse motivo, resolveu implementar uma solução que simulasse não só o processo de armazenagem e envio de mensagens, como também o processo de recebimento das mensagens sob o ponto de vista do destinatário. Além disso, Alice percebeu que talvez houvesse espaço para eventuais melhorias no sistema. Desta forma, resolveu utilizar seus conhecimentos em algoritmos e estruturas de dados para estudar o desempenho do sistema.

Após uma análise cuidadosa do sistema, Alice descobriu que o sistema poderia ser simulado a partir da implementação das seguintes funcionalidades:

- Recebimento de uma mensagem: como Alice não queria desenvolver um complicado sistema de comunicação, decidiu simular o recebimento de mensagens a partir da leitura de um arquivo texto, de forma que o conteúdo do arquivo seria correspondente a uma mensagem recebida.

- Tabela de frequência e código: a partir da mensagem "recebida", Alice percebeu que precisaria implementar um algoritmo que identificasse todos os caracteres na mensagem (letras somente – ignorando espaços e pontuação) e montasse a tabela de frequência equivalente em uma estrutura de dados apropriada. Desta forma, cada caractere teria um número associado, indicando sua frequência de ocorrência. Além disso, cada caractere possuiria um código binário gerado a partir da aplicação do código de Huffman, que também deveria ser armazenado na mesma estrutura. Assim, a partir de um caractere qualquer, Alice poderia consultar a estrutura de dados e recuperar tanto sua frequência de acesso quanto seu código correspondente.

- Uma vez que a mensagem precisaria ser "codificada" e "descodificada", Alice percebeu que precisaria implementar os algoritmos de Huffman tanto para codificação quanto para decodificação da mensagem.

- Sob o ponto de vista do destinatário, Alice percebeu que seu simulador precisaria também decodificar uma dada mensagem. Isso significa que o simulador deveria ser capaz de ler um par associando "mensagem codificada" e "tabela de frequência" para gerar uma "mensagem decodificada". Além disso, a mensagem decodificada deveria ser impressa na tela, de forma que ela pudesse comparar com a mensagem original e verificar se o processo de codificação e decodificação ocorreu sem problemas.

- Finalmente, de forma a analisar o uso de uma estrutura de dados alternativa para o sistema, Alice decidiu que poderia armazenar somente as palavras únicas da mensagem – ou seja, não armazenar palavras repetidas - e depois montar um algoritmo para recriar a mensagem original. Para armazenar as palavras, Alice sabia que poderia utilizar uma tabela *hash*. Para isso, seria necessário implementar uma função que calculasse o *hash* de cada palavra única de cada mensagem recebida, verificando a eventual ocorrência de colisões. Desta forma, utilizando o conceito de divisão e conquista, Alice resolveu trabalhar somente na função de *hash* neste momento. A função que Alice criou funcionava da seguinte forma:

- Cada palavra recebia um valor numérico, que era calculado somando-se os valores ASCII de seus caracteres; e

- O valor de cada palavra era dividido pelo número total de palavras únicas contidas na mensagem, de forma que o resto desta divisão era o *hash* calculado para a palavra.

Além disso, Alice implementou um algoritmo para detectar colisões, ou seja, comparar os *hashs* de todas as palavras únicas contidas na mensagem. Desta forma, ela poderia concluir se a função de *hash* escolhida era boa o suficiente para armazenar palavras contidas em mensagens.

2. Informações adicionais:

De forma que seja possível implementar os algoritmos de codificação e decodificação referentes ao código de Huffman, é necessário que alguns passos sejam seguidos, como mostrado abaixo:

a) Codificação:

- A árvore de Huffman deve ser montada e armazenada em memória (ver pseudocódigo dado em aula);

- A árvore deve ser percorrida (passeio) de forma que os caracteres recebam seus códigos de representação;

- Ao final do processo, cada caractere deve ser associado a uma frequência (ocorrência na sentença) e a um código de representação. Esta associação pode ser feita em uma estrutura separada em memória ou utilizando-se a própria árvore de Huffman criada, desde que a estrutura do nó-folha possua condições de armazenar todas as informações necessárias.

Como visto em aula, a atribuição de 0s e 1s pode ser feita de diferentes formas, e o caso de "frequências repetidas" (se houver três valores de frequências idênticos, quais somar?) pode ser resolvido de diferentes formas. No entanto, de forma a se simplificar o processo de atribuição de códigos de representação, recomenda-se uma abordagem programática – como, por exemplo, "fixar" as atribuições (0 ou 1) dependendo da direção tomada a partir de um nó (esquerda ou direita).

b) Decodificação:

- A tabela de codificação deve ser carregada do arquivo em disco (**não deve ser usada a estrutura criada no processo de codificação**);

- A sequência codificada deve ser analisada de forma que as representações de cada caractere sejam reconhecidas e "traduzidas" para uma estrutura de dados auxiliar (que irá armazenar a mensagem final decodificada);

- A análise da sequência codificada deve levar em conta que, de acordo com as propriedades do código de Huffman, nenhuma representação de caractere está contida como precedente de outra representação de caractere (por exemplo: se "a" é representado como "1", todas as outras representações de caracteres irão começar com 0).

O processo de decodificação pode ocorrer de duas formas diferentes, dependendo de como a tabela de codificação for utilizada. Pode-se, por exemplo, verificar-se o primeiro bit de representação de cada elemento na tabela, e depois o segundo, e assim por diante, até que seja reconhecida uma representação completa na sequência a ser descodificada. Uma outra alternativa seria montar uma árvore binária a partir da tabela, de forma que se pudesse caminhar por esta árvore utilizando 0s e 1s para guiar o caminho para a direita ou para a esquerda. Isto pode feito até que um nó-folha seja alcançado (contendo o caractere representado pela sequência de 0s e 1s utilizada para guiar o caminho).

3. Problemas a serem resolvidos neste EP:

O objetivo deste EP é replicar o trabalho feito por Alice e implementar um simulador para o sistema de envio de mensagens mencionado acima. Assim, existem os seguintes problemas a serem resolvidos:

- **Problema 1:** Receber e codificar uma mensagem. Este problema envolve:
  - Ler um arquivo texto existente contendo a mensagem a ser codificada, que **deve ser uma sentença com pelo menos dez palavras diferentes entre si e que faça sentido sob o ponto de vista semântico, utilizando a língua portuguesa** (nome fixo: "MENSAGEM.TXT");
  - Implementar uma estrutura de dados capaz de armazenar as frequências e códigos para cada caractere existente na mensagem (letras somente), de forma que a estrutura utilizada seja **pertinente ao problema**;
  - Implementar um algoritmo de codificação que, a partir da mensagem original, gere uma mensagem codificada;
  - Imprimir (mostrar) a mensagem codificada na tela;
  - Armazenar a mensagem codificada em disco, em um arquivo texto (nome fixo: "CODIFICADA.TXT");
  - Armazenar em disco os dados referentes a cada caractere (caractere/frequência/código), em um arquivo texto (nome fixo: "TABELA.TXT"). Qualquer organização pode ser utilizada neste arquivo.
- **Problema 2:** Receber uma mensagem codificada e descodifica-la. Este problema envolve:
  - Ler um arquivo texto contendo a mensagem codificada ("CODIFICADA.TXT");
  - Ler um arquivo texto contendo os dados de frequência e códigos referentes a cada caractere da mensagem ("TABELA.TXT");
  - Implementar um algoritmo de decodificação que, a partir dos dados carregados dos arquivos "CODIFICADA.TXT" e "TABELA.TXT", gere uma mensagem decodificada;
  - Imprimir (mostrar) a mensagem decodificada na tela;
  - Salvar a mensagem decodificada em disco (nome fixo: "DECODIFICADA.TXT").
- **Problema 3:** Receber uma mensagem não-codificada e calcular a função de espalhamento da mensagem, assim como verificar a existência de colisões. Este problema envolve:
  - Ler um arquivo texto existente contendo a mensagem (nome fixo: "MENSAGEM.TXT");
  - Converter cada palavra para um número inteiro, somando-se os valores ASCII de seus caracteres;
  - A partir do valor numérico obtido para cada palavra e do número de palavras não-repetidas existentes na mensagem, calcular seu *hash* (utilizando a mesma função criada por Alice);
  - A partir do *hash* de todas as palavras não-repetidas na mensagem, implementar um algoritmo que verifique a existência de colisões (*hashs* repetidos);
  - Imprimir (mostrar) as palavras da mensagem na tela, assim como seus *hashs* e a existência de colisões. O seguinte padrão pode ser usado:
    "PALAVRA: (palavra da mensagem) HASH: (*hash* equivalente da palavra) COLISÃO: (SIM/NÃO)"

4. Regras:

Podem ser utilizados todos os códigos vistos em sala de aula, assim como o material disponível nos slides e links de exemplos na página do curso. **NÃO PODEM** ser utilizadas bibliotecas de estruturas prontas, algoritmos, ou qualquer outra que reduza o esforço esperado para a solução do EP.

5. Dicas:

Desenvolva sua solução de forma incremental, implementando e testando cada trecho implementado. Após certificar-se que o trecho está 100%, avance sobre o próximo trecho.

6. Entrega:

A ENTREGA DO EP 2 SÓ SERÁ ACEITA ATRAVÉS DA PÁGINA DO CURSO!

A entrega do EP 2 pode ser feita através da entrega de um único arquivo compactado (.zip ou .rar), que **deverá conter**:

A. **Um arquivo texto (.doc, .docx, .pdf) referente ao relatório,** contendo:
   - A mensagem original utilizada nos testes da implementação
   - Uma tabela associando os caracteres contidos na mensagem a sua frequência de ocorrência e seu código de Huffman
   - Um parágrafo explicando a escolha da estrutura de dados utilizada para armazenar a tabela que associa caracteres a seus códigos/frequências de ocorrência
   - Um desenho da árvore de Huffman obtida a partir da tabela de frequência (utilizada para gerar os códigos de cada caractere)
   - Uma descrição dos algoritmos de codificação e decodificação implementados, contendo pseudocódigos para cada algoritmo e suas respectivas análises de complexidade
   - Uma tabela associando cada palavra a seu *hash* gerado, com uma coluna extra indicando se houve colisão para esta palavra
   - Um parágrafo analisando a função de *hash* implementada em relação ao problema que quer se resolver (armazenar palavras), dizendo se a função escolhida é boa ou não e justificando esta conclusão
   - Maiores dificuldades encontradas durante a resolução do EP
B. **Um arquivo (.c ou .java) reference ao código-fonte da implementação,** feita em C ou JAVA
C. **Um arquivo texto (.doc, .docx, .pdf) contendo instruções claras para a compilação do código implementado;**
D. **Arquivos texto utilizados na implementação:** “MENSAGEM.TXT”, “CODIFICADA.TXT”, “TABELA.TXT”, “DECODIFICADA.TXT”

Todos estes arquivos devem ser compactados em um único (.zip ou .rar), visto que a página de entrega do EP está configurada para aceitar o upload de somente um arquivo. **A não-conformidade com esta estrutura irá acarretar em perda de pontos no momento da avaliação.**