Trabalho Prático 3: O problema de compressão de mensagens de Rick Sanchez

Aluno: Guilherme Pereira
Matrícula: 2016023354

1. Introdução

O código foi feito com o intuito de resolver o problema de compressão de mensagens de Rick Sanchez. É preciso medir a frequência das palavras de uma determinada mensagem e comprimí-la, utilizando uma codificação de Huffman.

O trabalho foi dividido em duas partes, a primeira para contar quantas vezes cada palavra ocorreu em uma mensagem e a segunda para implementar a codificação de Huffman em si.

Devido a grandes dificuldades em compreender e implementar justamente o método de Huffman, somente a primeira parte do trabalho foi feita.

O objetivo deste trabalho é implementar diferentes estruturas de dados. Como foi feita somente a primeira parte, foi implementada uma tabela de hash.

2. Implementação

Estruturas de Dados:

Para a realização do trabalho foi criado um tipo de dado Cell, que possui três parâmetros, word, uma string que armazena uma palavra, next que é um apontador para a próxima Cell e count, um inteiro que armazena quantas vezes a palavra em questão ocorreu.

Além disso, foi criada a classe HashTable, responsável por justamente criar a hash table. Ela possui apenas dois parâmetros, size e table. Ou seja, o tamanho da table e a table em si.

Ela possui diversos métodos, que serão explicados com detalhe em seguida.

Funções e Procedimentos:

HashTable(int):
Construtor da hash table, que recebe como parâmetro o número de palavras da mensagem. Ele cria uma table com o tamanho igual ao das palavras com células vazias, em que o

nome de cada célula é um string vazio, o número de aparições é 0 e a próxima célula é nula.

~HashTable():
Destrutor da hash table, para evitar memory leaks. Ele chama o método deleteCell para cada chave da table e no fim deleta a table em si.

void deleteCell(Cell*):
Método recursivo para deletar todas as células de uma determinada chave. Funciona checando se a próxima célula é nula, ou seja, se a célula atual é a última. Caso positivo, a célula é deletada, caso contrário, a função é chamada recursivamente para a próxima célula e depois essa célula é deletada.

int hash(std::string):
Método que obtém as chaves da hash table. Recebe como parâmetro a palavra que será transformada em chave. Para isso, são somadas todas as suas letras e o resto da divisão dessa soma com o número total de palavras será a chave.

int getFreq(std::string):
Método que retorna o número de repetições de uma palavra. Para isso, primeiro é obtida a chave dessa palavra pelo método hash. Depois, é criado um apontador para célula com essa chave encontrada. Em seguida, essa sequência de células será percorrida até que a palavra desejada seja encontrado, e por fim, a contagem dessa palavra será retornada. Caso a palavra em questão não for encontrada, será retornado o valor 0.

void insert(std::string):
Método que insere uma nova palavra na hash table. Para isso, primeiro é obtida a chave dessa palavra pelo método hash. Depois, é criado um apontador para célula com essa chave encontrada. Em seguida, será verificado se a palavra em questão é inédita na table. Caso a palavra seja inédita, será criada uma nova célula que apontará para nullptr e a última célula anterior passará a apontar para ela, caso haja alguma. Caso contrário, apenas o count dessa palavra será acrescido de 1.

3. Instruções de compilação e execução

Para a compilação e execução foi usado um make file. O make file está localizado na pasta pasta guilherme_pereira. Para compila-lo é necessário um bash linux. Quando no bash, vá para o diretório guilherme_pereira, em que o make file está, como já foi dito. Depois, basta executar o seguinte comando “make ./tp3” sem parênteses para realizar a compilação. Para executar o programa, após compilá-lo, basta executar o comando “./tp3” também sem parênteses.

Após esses dois comandos a execução do programa irá começar. No primeiro momento é preciso inserir um inteiro, um número n de palavras. Em seguida, deve-se digitar a mensagem com esse n número de palavras. Depois disso, será possível fazer consultas de quantas vezes cada palavra digitada ocorreu. Para isso é preciso digitar a letra “q” (sem aspas) e a palavra desejada. Ex: “q acido” pesquisará quantas vezes a palavra acido apareceu, sempre sem aspas. O programa terminará quanto o usuário pressionar a combinação de teclas “Ctrl D”.

4. Análise de complexidade

HashTable(int):
Complexidade O(n) para espaço e O(n) para tempo, em que n é o número de palavras.

~HashTable():
Complexidade O(n*m) para espaço e O(n*m) para tempo, em que n é o número de elementos armazenados por uma chave e m o número de chaves. Possui complexidade espacial pois utiliza o método deleteCell, que é recursivo.

void deleteCell(Cell*):
Complexidade O(n) para espaço e O(n) para tempo, em que n é o número de elementos armazenados por uma chave. Possui complexidade espacial pois é uma função recursiva.

int hash(std::string):
Complexidade O(1) para espaço e O(n) para tempo, em que n é o número de caracteres em uma palavra.

int getFreq(std::string):
Complexidade O(1) para espaço e O(n/m) para tempo, em que m é o tamanho da tabela e n o número de registros.

void insert(std::string):
Complexidade O(1) para espaço e O(n/m) para tempo, em que m é o tamanho da tabela e n o número de registros.

Programa principal:
Como o programa principal usa basicamente todos os métodos citados, ele acaba tendo a complexidade O(n) para espaço e O(n) para tempo em média.

5. Conclusão

A implementação do trabalho mostra a importância da criação de algoritmos de ordenação eficientes. Foi possível perceber como cada algoritmo de ordenar possui vantagens e desvantagens e cada um deles é ideal para certas situações.

A implementação do trabalho mostra como estruturas como o hash podem diminuir o tempo de consulta. As complexidades O(n/m) das operações de inserção e de consulta podem ser consideradas constantes no tempo em média, o que é muito positivo.

Com tudo isso, a principal dificuldade foi a impossibilidade de realizar a segunda parte do trabalho, a implementação da codificação de Huffman, devido a sua grande complexidade de implementação.

6. Bibliografia

https://www.hackerearth.com/pt-br/practice/data-structures/hash-tables/basics-of-hash-tables/tutorial/