Estruturas de Dados e Algoritmos I
Trabalho prático 2020/2021
- Serviço de Mensagens Curtas Inteligente -

Autores:
Miguel Portugal - nº38128
Vasco Barnabé - nº42819

20 de Junho de 2021

1 Introdução

O Trabalho prático de Estruturas de Dados e Algoritmos I consiste no desenvolvimento de uma aplicação que implementa a tecnologia de mensagens curtas inteligente T9 (Text on 9 Keys).

Esta tecnologia permite escrever palavras com as teclas numéricas dos telemóveis: cada tecla corresponde a 3 ou 4 letras do alfabeto e a utilização de um dicionário permite introduzir uma palavra através dos algarismos correspondentes a cada uma das suas letras. Para isso, o utilizador tem como base os seguintes grupos de letras para que possa introduzir palavras através de algarismos que representem palavras que pretende obter para a sua mensagem:

- 2: a b c á à â ã ç
- 3: d e f é ê
- 4: g h i í
- 5: j k l
- 6: m n o ó ô õ
- 7: p q r s
- 8: t u v ú
- 9: w x y z

A aplicação a desenvolver permite escrever a mensagem palavra a palavra até a indicação de fim de mensagem (o utilizador digita 1 se pretender visualizar a mensagem). A aplicação recebe os algarismos digitados pelo utilizador, identificando um tipo de palavra pretendida e sugere uma palavra; se essa palavra não for a desejada, sugere outras. Caso não existam mais sugestões, a palavra é introduzida pelo teclado e adicionada ao dicionário.

2 Desenvolvimento

2.1 Estruturas de Dados utilizadas

Inicialmente, a aplicação recebe um dicionário de palavras, as quais terão que ser guardadas de alguma forma, para não ter que ser necessário efetuar uma busca diretamente ao ficheiro que contém as palavras de cada vez que se pretende obter uma destas. Por isso, foram implementadas estruturas como a Hash Table e as Linked Lists por forma a guardar as palavras do dicionário.

LinkedList

As Linked List foram implementadas com o objetivo de, em cada lista, apenas serem guardadas palavras do mesmo tipo (por exemplo, "pato"e "rato"são ambas palavras do tipo "7286"). Assim, cada nó da lista contém uma palavra, o tipo da palavra, e um apontador para a próxima palavra da lista.

Hash Table

A Hash Table criada contém, inicialmente, uma lista vazia em cada posição, e assim, no fim da inserção de todas as palavras, em cada posição da hash table existe uma lista que pode estar vazia ou pode conter um conjunto de palavras do mesmo tipo.

Foi utilizado o método Hashcode() que recebe como parâmetro o tipo de uma palavra, e gera o index onde essa palavra vai ser colocada na hash table. Em casos de colisões, aplica-se um tratamento linear.

A implementação destas duas estruturas de dados foi feita com base nos conhecimentos adquiridos nas aulas teóricas e práticas da disciplina, tendo sido implementados apenas métodos com utilidade para a aplicação.

2.2 Mecanismo da aplicação

2.2.1 Processamento dos Dados

Começando pela leitura dos dados, as palavras do dicionário são lidas e analisadas, uma a uma, caractere a caractere, de forma a identificar o tipo de cada palavra. Durante a análise de cada palavra este tipo é gerado, e no fim, a palavra é inserida na hash table consoante o seu tipo. Este identificador (tipo) passa pelo método **HashCode()**, que gera um index. Esse index corresponde à posição onde o programa tentará inserir a palavra na hash table.

Nessa posição, existe uma lista, que pode estar vazia, ou já conter palavras. Caso a lista se encontre vazia, a palavra é inserida diretamente, sinal de que ainda não existem palavras daquele tipo guardadas na hash table. Caso a lista não esteja vazia, duas situações podem ocorrer: a lista guarda palavras daquele tipo; a lista guarda palavras de um tipo diferente. Ou seja, sempre que se insere uma palavra e a lista naquela posição já não se encontre vazia, é necessário verificar se o tipo da palavra que pretendemos inserir e os tipos das palavras que já se encontram guardadas naquela lista, são iguais. Se forem iguais, a palavra é inserida na lista, caso contrário, estamos perante uma colisão.

Como já foi referido anteriormente, em casos de colisão aplica-se um tratamento linear, e assim, todas as posições seguintes da hash table serão visitadas até que seja encontrada a posição em que a lista guarde palavras daquele tipo (da palavra que queremos inserir), ou em que a lista ainda esteja vazia, para que se possa então proceder à inserção da palavra na hash table.

Este circuito é efetuado para todas as palavras até ao fim do dicionário, obtendo, por fim, uma hash table com todas as palavras guardadas e divididas em listas, pelos seus tipos.

Nota: Nas palavras que contêm hífen(-) ou apóstrofe('), estes caracteres são ignorados, não sendo contabilizados para o tipo da palavra. No entanto, as palavras são guardadas e apresentadas ao utilizador com a sua forma correta, por exemplo: "deixa-me"contém um hífen, mas o seu tipo é "3349263", não sendo, por isso, o hífen aqui contabilizado.

2.2.2 Pesquisa e Sugestão de palavras

A pesquisa das palavras na hash table é feita no momento em que o utilizador digita a sequência de algarismos representante do tipo de palavra que pretende inserir na mensagem.

Essa sequência representa o tipo da palavra, e o programa tem de pesquisar as palavras daquele tipo, para que possam ser sugeridas ao utilizador. Para isso, é realizado um processo similar ao realizado quando se dá a inserção de uma palavra na hash table. O tipo da palavra inserido pelo utilizador vai servir como parâmetro do método **HashCode()** que devolve o index onde o programa vai aceder na hash table para sugerir palavras ao utilizador. Uma vez mais, a lista que se encontra nessa posição da hash table pode-se encontrar vazia, ou pode conter palavras. No caso de estar vazia, significa que não existem palavras daquele tipo, e, por isso, o utilizador insere uma nova sequência de algarismos (este processo repetir-se-á até que o utilizador digite uma sequência que corresponda ao tipo de alguma palavra existente). Caso a lista não esteja vazia, é necessário verificar se aquela lista contém palavras do tipo pretendido, uma vez que, se não forem do mesmo tipo, significa que existiram colisões quando as palavras foram guardadas na hash table, e como o tratamento realizado para as colisões foi linear, é necessário visitar as listas seguintes até encontrar aquela que contenha palavras do tipo digitado pelo utilizador.

Uma vez encontrada a lista certa, a primeira palavra, a que se encontra à cabeça da lista, é sugerida ao utilizador. Aqui, este decide se quer aceitar ou não a palavra sugerida. Se aceitar, a palavra é automaticamente adicionada à mensagem e o utilizador pode digitar o tipo da próxima palavra que pretende; se não aceitar, o programa vai avançando pela lista, posição a posição, sugerindo palavra a palavra daquela lista ao utilizador, até que este aceite, ou até que não existam mais sugestões para dar.

Neste momento, se o utilizador não aceitar nenhuma das palavras que lhe foi sugerida, este pode introduzir a palavra exata que pretende a partir do teclado, a qual será automaticamente adicionada à mensagem final e guardada na hash table, passando então a pertencer ao conjunto de sugestões posteriores, mas apenas para a própria sessão.

2.2.3 Menú e Ações do Utilizador

Durante o decorrer do funcionamento do programa e da construção da mensagem, algumas informações vão sendo fornecidas ao utilizador, bem como o poder de decisão em alguns momentos, escolhendo, dentro das opções disponíveis, aquela que mais é do seu agrado para a situação em que se encontra. Assim:

- No início do programa o utilizador é notificado com o tempo que este demorou a processar e guardar os dados (dicionário) fornecidos;
- É apresentada ao utilizador uma tabela com a codificação de cada algarismo, para que este possa digitar corretamente o tipo de palavra que pretende;
- O utilizador pode, digitando uma sequência de algarismos válida, dizer ao programa qual o tipo de palavra que pretende;
- O utilizador pode aceitar ou recusar as sugestões de palavras do programa;

- O utilizador pode inserir diretamente do teclado a palavra que pretende inserir na mensagem, após recusar todas as sugestões fornecidas pelo programa (passando esta nova palavra inserida pelo utilizador a estar disponível para futuras sugestões, naquela sessão);
- Sempre que o utilizador tiver que responder "sim"ou "não"ao programa, caso não o faça corretamente, isto é, utilizando os caracteres reservados para este efeito, é notificado de que tem de responder corretamente;
- O utilizador pode premir a qualquer altura o caractere "1"e ser-lhe-á apresentada a mensagem atualizada;
- O utilizador pode premir o caractere "0"em qualquer altura para sair da aplicação.

Nota: A aplicação criada cumpre todos os requisitos exigidos, à exceção das funcionalidades adicionais que não foram implementadas.

2.3 Métodos Implementados

Linked List
Métodos implementados para a criação e utilização de uma Linked List:

- **new_nodeWord(wchar_t* value, char* id)** - este método cria e retorna um nó, com uma palavra e o seu tipo;
- **new_linkedListWord()** - este método cria uma lista vazia;
- **list_insertWord(LinkedListWord* list, wchar_t* value, char* id)** - este método é responsável por inserir numa lista específica, uma palavra e o seu tipo, utilizando como seu auxiliar o método **new_nodeWord()**;
- **checkIfIdExists(LinkedListWord* list, char* id)** - este método verifica se um certo tipo de palavras corresponde ao tipo de palavras guardadas na lista; este método é utilizado nos processos de alocar ou pesquisar palavras;
- **find_valueWord(LinkedListWord* list, int position)** - este método é responsável por retornar uma palavra que se encontre numa posição específica numa determinada lista; este método é utilizado no processo de sugestão de palavras ao utilizador;
- **is_emptyWord(LinkedListWord* list)** - este método verifica se uma determinada lista está vazia ou não;
- **list_lengthWord(LinkedListWord* list);** - este método retorna o tamanho de uma detrminada lista, isto é, o número de palavras que contém.

Hash Table
Métodos implementados para a criação e utilização de uma Hash Table:

- **InitializeTable()** - este método cria uma hash table com uma lista vazia em cada posição, utilizando o método **new_linkedListWord()** como auxiliar;
- **HashCode(char* id)** - recebendo como parâmetro o tipo de uma palavra, este método gera um index (posição) na hash table, dependendo do tipo que recebe; este método é utilizado para o programa descobrir onde vai tentar guardar uma palavra, ou por onde deve começar a procurá-la.

- **Find(HashTable* H, char* id, int position)** - Este método, com o auxílio do método **find_valueWord()** da Linked List, retorna a palavra que se encontre numa certa posição, numa certa lista da hash table.

- **Insert(HashTable* H, wchar_t* word, char* id)** - método utilizado para inserir uma palavra na hash table, utilizando o método **list_insertWord()** da Linked List como auxiliar para inserir a palavra na lista pretendida.

Main

Implementação dos métodos utilizados no processamento inicial dos dados e do Menú da aplicação:

- **createTypeOfWord(char* typeOfWord, wchar_t* word)** - método que gera o tipo de uma palavra, analisando-a caractere a caractere e atribuindo a cada um o algarismo correspondente, retornando por fim o tipo da palavra;

- **processData(HashTable* hashtable, char* argv)** - este método trata de todo o processamento dos dados, lendo o dicionário e utilizando como auxiliares os métodos **createTypeOfWord()** e **Insert()** para gerar o tipo de cada palavra e inseri-la na Hash Table, respetivamente. Por fim, apresenta o tempo que levou este processo descrito, deixando também todo o dicionário gravado na Hash Table;

- **main(int argc, char* argv[])** - onde se encontra deifinido todo o menú com que o utilizador interage e onde vários métodos anteriormente definidos são invocados para que o programa tenha o comportamento esperado.

3 Conclusão

Tendo sido este trabalho prático uma forma de pôr em prática alguns dos conhecimentos obtidos durante as aulas de Estruturas de Dados e Algoritmos I, foi-nos possível aprimorar conhecimentos sobre as estruturas abordadas e aqui utilizadas, nomeadamente Linked Lists e Hash Tables, bem como a forma como podemos resolver um problema, não de uma maneira qualquer, mas sim de uma maneira mais eficiente. Olhando para o exemplo deste problema em concreto, esta aplicação poder-se-ia basear num simples array com uma palavra em cada posição, e de cada vez que fosse necessário dar uma sugestão ao utilizador, fazer uma pesquisa linear, partindo do início do array, até encontrar uma possível palavra para sugerir. Tal não seria a abordagem mais eficiente, daí a utilização de uma hash table, onde podemos realizar através de chaves, acessos diretos, sabendo à priori, que o que pretendemos encontrar é obtido de forma direta, ou quase direta (casos de colisões, que aqui foram resolvidas com tratamento linear, uma vez que é o suficiente tendo em conta as colisões que aqui possam ocorrer).

Além disso, esta foi também uma oportunidade para aumentar-mos o nosso conhecimento e a nossa familiarização com a linguagem de programação utilizada, a linguagem C, especialmente no que toca à utilização e domínio de "wide strings"(caracteres especiais, um dos obstáculos com que nos depará-mos no início do desenvolvimento deste trabalho).

Por fim, foi-nos possível adquirir mais experiência no que toca à arte da programação em geral, além da linguagem em si, como já foi referido, e no entendimento das estruturas de dados e do seu funcionamento, utilizando conhecimentos obtidos nas aulas da disciplina e de pesquisa realizada, para obter como resultado final, um programa funcional e eficiente, um serviço de mensagens curtas inteligente.