# Trabalho Prático EDA1 – 20/21

- Serviço de mensagens curtas inteligente -

T9 (Text on 9 keys)

40090 / 45491


UNIVERSIDADE DE ÉVORA

# Objetivo

Desenvolver uma aplicação que implementa a tecnologia de escrita de mensagens curtas inteligente T9 (*Text on 9 Keys* ).

Esta tecnologia permite escrever palavras com as teclas numéricas do telemóvel: cada tecla corresponde a 3 ou 4 letras do alfabeto e a utilização de um dicionário permite introduzir uma palavras através dos algarismos correspondentes a cada uma das suas letras.

# Estrutura de Dados

Para resolver este problema, decidimos utilizar a estrutura de dados *Tabela de Dispersão* com cadeias separadas para resolver as colisões. Logo obtemos uma complexidade temporal de inserção constante O(1), e uma complexidade temporal de acesso que varia conforme o tamanho da tabela, caso o tamanho da tabela seja superior ao numero de elementos a inserir, obtemos sempre complexidade constante O(1), caso seja inferior, obtemos complexidade no pior dos casos O(n / tamanho da tabela).

# Funções Implementadas

I**nt wordToNum (char * s) –** Esta função recebe uma string correspondente a uma palavra, e retorna o valor numérico que representa a forma como se escreveria a palavra segundo a tecnologia T9. Esta função é fundamental pois foi a partir dela que desenvolvemos a nossa função de *Hash* e serve-nos de base para comparação de input com os valores da tabela.

**Node* new_node(char str[MAX_WORDSZ]) –** Esta é a função constructora dos nós da lista ligada que será usada no encadeamento separado. Recebe uma string com os dados a serem guardados (neste caso uma string de tamanho predefinido) e devolve um apontador para o nó que tem como dados: a string, a chave (string passada para numero através da função wordToNum) e um apontador para o nó seguinte.

**List* new_list(struct node* N) –** Esta é a função constructora da lista ligada, recebe um nó (pode ser NULL) e define-o como *Head e Tail* da lista simultaneamente. Devolve um apontador para a nova lista.

**Void add(struct list* L, struct node* N) –** Esta função acrescenta um nó no fim da lista. Tem complexidade temporal constante.

**Hash_Table* new_table(int size_t) –** Esta é a função constructora da tabela de dispersão. Recebe o tamanho da lista e devolve uma tabela de dispersão que contém como dados o seu tamanho e um array de apontadores para listas ligadas de tamanho size_t.

**Int hash(int key, int t_size) –** Esta é a função de *Hashing* utilizada para definir os indices de inserção e acesso na tabela. O seu valor de retorno varia com o valor inserido na chave e também com o tamanho da tabela, utilizámos a fórmula mais simples que existe para esta função:

Hash(x, tamanho) = x mod tamanho

**Void insert(hash_table* table, node* item) –** Esta função insere um nó na tabela, após calcular o indice a partir do tamanho da tabela e do valor contido no nó a inserir, adiciona o nó a lista ligada indicada a partir da função *add.*

**Void t9_predict(hash_table* table) –** Esta é a função principal do programa, recebe uma tabela de dispersão preenchida com palavras. Durante a execução pede input numérico ao utilizador e determina um indice de pesquisa a partir da função *Hash*, ao aceder a esse indice no array da tabela já sabemos se existe uma palavra que se escreve da mesma forma que foi dado o input, caso exista, o programa irá sugerir a palavra contida na *Head da lista ligada* e pede ao utilizador para indicar se é a palavra desejada ou não, caso não seja a palavra desejada ou caso não exista nenhuma palavra que se escreva a partir do input dado, então o programa pede ao utilizador para escrever a palavra que ele desejava, essa palavra será então adicionada à tabela e também à mensagem que está a ser escrita a cada iteração.

**Main –** Na main o programa carrega um dicionário em formato .txt ,insere todas as suas palavras na tabela de dispersão e indica-nos quanto tempo demorou a inserir. De seguida faz print das teclas do programa e também das instruções. Depois chama a função *T9_predict* e só para de correr quando o input for “0”.

# Conclusão

O trabalho cumpre os requisitos básicos indicados no enunciado, no entanto não conseguimos implementar nenhuma das funcionalidades recomendadas de otimização baseada em estatística.

Após a realização concluímos que a tabela de dispersão é uma estrutura de dados bastante apropriada quando queremos fazer inserções e acessos com complexidade constante, no entanto poderíamos ter feito a sua implementação de forma mais abstracta.

Durante a realização do trabalho não implementámos todas as funções normalmente implementadas para as estruturas dados que utilizámos (lista e tabela), só implementámos aquelas que iriamos realmente utilizar (para as listas ligadas só temos a função add porque só adicicionamos à lista sempre no fim, etc..) , isto permitiu-nos implementar o programa com muito mais simplicidade e tornou-o mais “reader friendly”.

Trabalho realizado por:

Duarte Anastácio nº40090

Beatriz Girão nº 45491