

# Word Ladder

**Relatório 08/12/2022**

## Tiago Sousa Fonseca (107266) – 37.5%

**Tomás Sousa Fonseca (107245) – 37.5%**

## Beatriz Ferreira (107214) – 25%

**Universidade de Aveiro Algoritmos e Estruturas de Dados**

**Índice**

[Introdução 3](#_TOC_250005)

Funções Modificadas 4

hash\_table\_create [4](#_TOC_250003)

hash\_table\_grow 4

hash\_table\_free 5

find\_word 5

find\_representative 5

add\_edge 6

breadh\_first\_search 6

list\_connected\_component 6

path\_finder 7

connected\_component\_diameter 7

graph\_info 7

[Comentários 8](#_TOC_250001)

[Apêndice 9](#_TOC_250000)

# Introdução

Uma “word ladder” é uma sequência de palavras em que duas palavras adjacentes diferem por uma letra. Por exemplo:

* Inglês:

head —> heal —> teal —> tell —> tall —> tail

* Português:

tudo —> todo —> nodo —> nado —> nada

Seguindo o princípio estabelecido para a diferenciação de duas palavras, podemos assim estabelecer um componente conectado, ao qual uma palavra pertence.

A partir de vários componentes conectados, é possível estabelecer um caminho entre duas palavras.

# Funções Modificadas

## hash\_table\_create:

## Este código é uma função em C chamada "hash\_table\_create" que cria e inicializa uma estrutura de dados de tabela hash. A função começa por alocar memória para a tabela hash usando o malloc e atribui o endereço de memória a um ponteiro chamado "hash\_table". Se a alocação de memória falhar, a função irá imprimir uma mensagem de erro "create\_hash\_table: out of memory" para o stream de erro padrão e sair do programa com o status 1. Em seguida, a função define o tamanho da tabela hash para 50, o número de entradas e arestas para 0. Em seguida, a função usa o malloc novamente para alocar memória para um array de ponteiros chamado "heads" com o tamanho da tabela hash. Se a alocação de memória falhar novamente, a função irá imprimir a mesma mensagem de erro e sair do programa. Por fim, ele usa um loop for para inicializar todos os ponteiros no array com NULL, e depois a função retorna a tabela hash criada.

## hash\_table\_grow:

## Este código é uma função em C chamada "hash\_table\_grow" que aumenta o tamanho da tabela hash. A função começa por criar array de ponteiros para as cabeças dos nós da tabela hash com o tamanho da tabela atual multiplicado por dois. Se não for possível alocar memória para o novo array, a função imprimirá uma mensagem de erro "create\_hash\_table: out of memory" para o fluxo de erro padrão e sairá do programa com o status 1. Em seguida, o loop for inicializa todos os ponteiros no novo array com NULL. Depois, outro loop for percorre cada nó na tabela antiga e usa uma função de hash (crc32) para calcular o novo índice para cada nó e adicioná-lo ao novo array. Por fim, a função libera a memória do array antigo, atualiza a tabela hash com o novo array e atualiza o tamanho da tabela.

## hash\_table\_free:

## Este código é uma função em C chamada "hash\_table\_free" que tem como objetivo libertar a memória alocada para uma tabela hash. A função usa um loop for para percorrer cada ponteiro no array de cabeças da tabela hash e liberta a memória de cada nó da tabela usando a função free\_hash\_table\_node. Depois, a função usa a função free para libertar a memória do array de cabeças. A função tem como objetivo garantir que toda a memória alocada para a tabela hash seja libertada, evitando vazamentos de memória e garantindo que a memória possa ser reutilizada pelo sistema.

## find\_word:

## Este código é uma função em C chamada "find\_word" que encontra um nó específico em uma tabela hash. A função usa uma função de hash (crc32) para calcular o índice do nó com base na palavra dada. Em seguida, usa um loop para percorrer a lista encadeada no índice calculado e verifica se a palavra no nó atual corresponde à palavra dada. Se encontrar o nó, a função o retorna. Se o parâmetro "insert\_if\_not\_found" for definido como 1 e o nó não for encontrado, a função aloca e inicializa um novo nó com a palavra dada e o insere na tabela hash. A função também verifica se a tabela está quase cheia e, se estiver, chama a função "hash\_table\_grow" para aumentar o tamanho da tabela. Se o nó não for encontrado e o parâmetro "insert\_if\_not\_found" não for definido como 1, a função retorna NULL.

## find\_representative:

Este código é uma função em C chamada "find\_representative" que encontra o representante de um nó em uma tabela hash. O representante é o nó de cima na estrutura de conjunto de cada grupo de palavras relacionadas. A função usa dois loops while para percorrer a estrutura de conjunto até chegar ao representante. O primeiro loop começa com o nó dado e segue o ponteiro "representative" até encontrar o representante. O segundo loop começa com o nó dado e segue o ponteiro "representative" até o representante, alterando cada nó visitado para apontar diretamente para o representante. Isso é conhecido como compressão de caminho e é usado para acelerar a busca de representante em futuras chamadas. A função retorna o representante encontrado.

## add\_edge:

Esta função, chamada "add\_edge", é usada para adicionar uma aresta entre dois nós na tabela hash. O nó "from" e o nó "word" são passados como argumentos, com a tabela hash também sendo passada. A função usa primeiro a função "find\_word" para localizar o nó "to" na tabela hash que corresponde à palavra passada. Se este nó não for encontrado, a função sai. Caso contrário, o número de arestas na tabela hash é incrementado e um nó de adjacência é alocado para ligar os nós "from" e "to". A função, então, realiza uma operação de conjunto-união nos nós representativos do "from" e "to" para combiná-los se eles não estiverem já no mesmo conjunto. A função termina retornando.

## breadh\_first\_search:

Esta função, chamada "breadth\_first\_search", é um algoritmo de busca em largura usado para encontrar o caminho mais curto entre dois nós em um grafo representado por uma tabela hash. A função leva como argumentos o número máximo de vértices, uma lista de todos os vértices, o nó de origem e o nó de destino. A função usa uma fila para controlar os nós a serem visitados e começa inicializando o nó de origem como o primeiro elemento da fila, marcando-o como visitado e definindo o nó anterior como nulo. A função então itera através da fila até que ela esteja vazia, retirando o nó da frente, verificando se é o nó de destino, se for, retorna o número de vértices visitados. Se o nó de destino não for encontrado, a função itera através dos vizinhos do nó retirado e os adiciona à fila se eles ainda não foram visitados. A função termina retornando o número de vértices visitados e redefinindo o status visitado de todos os vértices visitados na busca para 0.

## list\_connected\_component:

Esta função, chamada "list\_connected\_component", é usada para listar todos os nós que estão conectados a um nó específico em uma tabela hash. A função leva como argumentos a tabela hash e uma palavra, e usa a função "find\_word" para localizar o nó específico na tabela hash. Se o nó não for encontrado, a função sai. Caso contrário, a função usa a função "find\_representative" para encontrar o nó representativo do componente conectado ao qual o nó específico pertence. Em seguida, usa a função "breadth\_first\_search" para realizar uma travessia em largura do componente conectado e armazena os nós visitados em uma lista. A função, então, imprime as palavras dos nós na lista e libera a lista de vértices.

## path\_finder:

Esta função, chamada "path\_finder", é usada para encontrar e imprimir o caminho mais curto entre duas palavras em uma tabela hash. A função recebe como argumentos a tabela hash e as duas palavras, "from\_word" e "to\_word". Em primeiro lugar, a função utiliza a função "find\_word" para localizar os nós correspondentes na hash\_table para as palavras "from\_word" e "to\_word" e os atribui às variáveis "from" e "to". Se uma das palavras não for encontrada na hash\_table, a função sai e imprime uma mensagem de erro. Se ambas as palavras forem encontradas, a função utiliza a função "breadth\_first\_search" para encontrar o caminho mais curto entre os nós "from" e "to" e armazena os nós visitados em uma lista. Em seguida, a função itera através dos nós na lista, começando pelo nó "to" e imprime as palavras de cada nó no caminho, com uma seta "->" entre cada palavra e uma nova linha no final.

## connected\_component\_diameter:

A função, chamada "connected\_component\_diameter", é usada para encontrar o caminho mais longo dentro de um componente conectado de um grafo representado em uma tabela hash. A função leva como argumento um nó da tabela hash. A função primeiro encontra o nó representativo do componente conectado ao qual o nó de entrada pertence. Em seguida, aloca uma lista de vértices e usa a função "breadth\_first\_search" para encontrar todos os nós no componente conectado. A função, então, itera através dos nós na lista e usa a função "breadth\_first\_search" novamente para encontrar o caminho mais longo entre cada nó e todos os outros nós no componente conectado. O caminho mais longo encontrado é armazenado como o diâmetro do componente conectado. A função, então, libera a lista de vértices alocada e retorna o diâmetro do componente conectado.

## graph\_info:

A função "graph\_info" é usada para imprimir informações sobre um gráfico armazenado em uma tabela hash. A função recebe a tabela hash como argumento.

A função primeiro usa um laço for para percorrer o array de listas ligadas na tabela hash e chama a função "breadth\_first\_search" no primeiro elemento não nulo no array. O número de nós visitados pela busca em largura é armazenado na variável "num\_visited".

Em seguida, a função imprime o número de nós no gráfico e o número de componentes conectados no gráfico.

A função então usa outro loop for para percorrer a lista de vértices visitados pela busca em largura e chama a função "connected\_component\_diameter" em cada nó não visitado, a fim de encontrar o número de nós no maior componente conectado e o diâmetro do maior componente conectado. A função também imprime o exemplo do maior componente conectado, que é o caminho encontrado pelo algoritmo de busca em largura e armazenado na variável largest\_diameter\_example. Finalmente, a função libera a lista de vértices