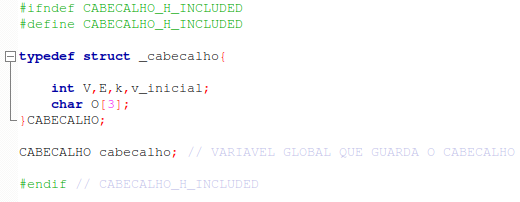
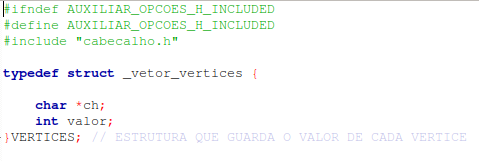
Estruturas:

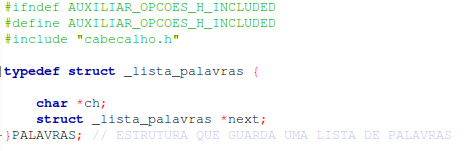
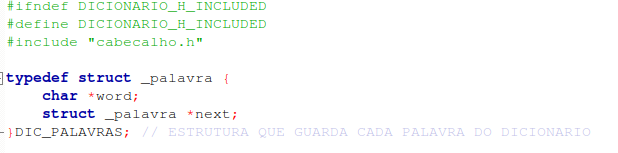


Esta estrutura, visa guardar a informação do cabeçalho lido apartir do ficheiro dos grafos. Estando lá presentes os 5 elementos: V -> número de vertices, E -> número de arestas, O[3] -> opção a realizer, k -> valor objetivo e, por fim, v\_inicial -> vértice de partida. Relativamente a esta estrutura, ainda é de verificar que declaramos uma variável global, de nome cabecalho, de forma a termos acesso ao dito cabeçalho, ao longo do programa.

Esta estrutura, serve para guardar informação obtida apartir do ficheiro dos grafos. O valor e as letras correspondentes a cada um dos vertices, vai ser guardada num vetor alocado dinamicamente, de tamanho igual ao número de vertices existentes. Desta forma, os dados correspondentes ao vértice 1, vão estar guardadas na posição 0 do vetor, as do vértice 2, na posição 1, e por aí em diante. Esta estrutura é deveras importante, uma vez que é apartir dos dados que se encontram no vetor de VERTICES, que vamos construindo palavras, e posteriormente comparar as palavras obtidas com um dicionário, para resolver cada uma das opções.

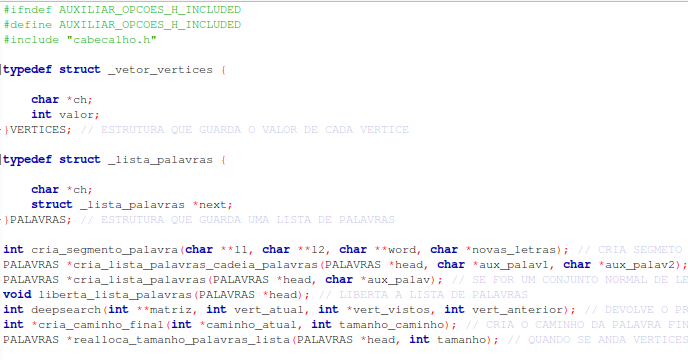


Esta estrutura, tem como objetivo, guardar as palavras que vão sendo formadas ao longo do programa, isto é, à medida que se vão lendo as letras que estão nos vertices, vão se introduzindo-as numa lista de PALAVRAS, e posteriormente, estes conjuntos de letras vão ser comparadas com o dicionário, para ver se algum pode ser realmente chamado de palavra (explicado com mais detalhe posteriormente).



Esta estrutura, permite guardar as palavras de um dicionário. Em si a estrutura é exatamente igual à estrutura PALAVRAS, mas como possui outro objetivo, decidimos criar uma nova estrutura, apenas para não nos confundirmos. Vamos criar então uma hash table com as palavras do dicionário, que são lidas de um ficheiro.

FUNÇÕES DO AUXILIARES\_OPCOES.H



Estas funções vão como o nome indica, auxiliar na resolução do pretendido pelas opções, elas vão ter então um carácter mais geral, o que faz com que estas sejam usadas, independentemente, da opção imposta pelo enunciado. Nós de seguida, vamos explicar o funcionamento das funções, que consideramos serem as mais complexas.

C:\Users\linha\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ssss.png

Esta função vai receber como argumento 3 variáveis do tipo char, passadas por endereço e uma que não é enviada por endereço, mas que são alocadas dinamicamente. A variável novas\_letras, possui as letras presentes no vértice em que estamos atualmente. É então dentro desta função que se descobre se o conjunto de letras é uma cadeia de letras, um prefixo, um sufixo ou um conjunto normal de letras. Consoante o que seja o conjunto de letras seja, a função retorna um inteiro, que vai de 1 a 4 o que permite, que apartir daí se tome as decisões necessária.

Se as letras corresponderem a uma cadeia, a primeira parte é enviada pela variável l1 e a segunda pela variável l2. Qualquer uma das outras é enviada pela variável word. Estas variáveis vão ser posteriormente introduzidas na lista de palavras.

C:\Users\linha\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ssss.png

Ambas as funções vão servir para introduzir as letras retiradas e já posteriormente tratadas, isto é de pois de se retirar o: - ou a /, dos vértices, numa lista.

1. Funcionamento geral:

* As funções vão receber um conjunto de letras;
* Se a lista estiver vazia vão alocar memória, para introduzir as letras, caso esta não esteja vão tartar de introduzir estas letras, nas palavras já lá existentes;

1. Funcionamento específico:

* A função **cria\_lista\_palavras,** vai introduzir as novas letras nas palavras já lá existentes, ou caso não existir nenhuma palavras criar uma nova, com essas letras;
* A função **cria\_lista\_palavras\_cadeia\_palavras,** vai copiar a palavra que já lá está e depois introduzir-lhe a primeira letra da cadeia, e depois vai introduzir a segunda letra, na palavra copiada e introduzi-la no final da lista, seguindo para a próxima palavra da lista. Caso não exista nenhuma palavra na lista, vão ser criadas duas novas (uma com cada element da cadeia);

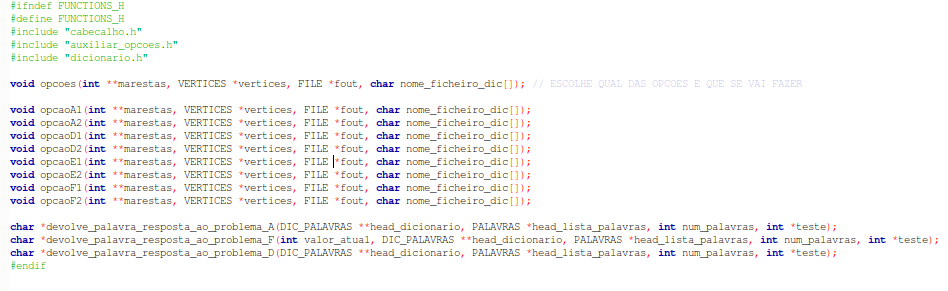
C:\Users\linha\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ssss.png

É nesta função que se vai saber qual é o vértice seguinte. Esta função recebe o vértice anterior. Caso este seja == -1 (o que significa que na volta anterior o vértice já não tinha mais vertices em comum), vai- se fazer um for{} que vai desde o (vértice anterior + 1), até ao número de vertices máximo. Caso exista uma aresta e esse vértice não se encontre no vert\_vistos (explicado com mais detalhe posteriormente), vetor correspondente ao caminho, esse vai-se tornar o vertice seguinte. Se não for encontrado , retorna-se -1.

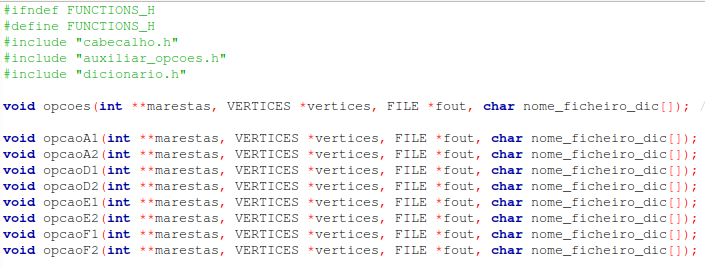
Se o vértice anterior for diferente de -1, faz-se exatamente o mesmo que anteriormente, mas o ciclo for{}, vai ser feito desde o vértice 0.

C:\Users\linha\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ssss.pngEsta função é responsável, por atualizar a lista de palavras, caso se volte um vértice para trás. Se o tamanho for != 1, apenas se diminui o tamanho das palavras na lista. Isto é , supondo que estamos no vértice 7 e adicionámos 3 letras, então, para voltar para trás vamos realocar as palavras para ficarem com menos 3 letras. Caso o tamanho seja == 1, como não sabemos se vamos retirar uma letra que correspondeu anteriormente a uma cadeia de letras, vamos fazer exatamente o mesmo que anteriormente, mas depois vamos copiar as palavras para uma nova lista, e caso essa palavra ainda não exista na nova lista, é introduzida, caso já exista, não a introduzimos.

FUNÇÔES DO FUNCTIONS.H



Estas funções, visam responder ao pretendido pelo enunciado, e por isso, apesar de serem banstante parecidas, vão possuir diferenças, de acordo com as condições impostas, vão ter assim um carácter mais específico. De seguida passamos, a uma explicação mais detalhada do seu funcionamento.



A função **opcoes** serve apenas de meio transitório. Nesta função compara-se qual é a opção que se vai efetuar. Isto é feito através de um conjunto de if{} else{}, onde se compara a variável O[3], presente na variável global cabecalho, com as diversas hipóteses de opção. Caso não exista a opção pretendida, faz-se fprintf de -1 para o ficheiro de saída e dá-se esse cabeçalho por terminado. Caso exista, segue-se para uma das funções que se apresenta na figura em cima.

As funções que se apresentam vão ser explicadas mais adiante, com mais cuidado, mas fica de salientar, que é nestas funções que se vão correr as funções, que permitem correr os diversos vértices, criar o dicionário, a lista de palavras, entre outros. É então realmente nestas funções que se encontra as respostas aos cabeçalhos.

C:\Users\linha\AppData\Local\Microsoft\Windows\INetCache\Content.Word\ssss.png

Estas funções são um elemento fundamental para se encontrar palavras. Estas funções vão receber como argumento, uma lista de palavras (que vai ser explicada posteriormente), que consiste no conjunto atual de palavras que se podem formar de acordo com o caminho de vertices atual, o dicionário com o qual se vão comparar as palavras da lista, o numero de palavras do dicionário (como consiste numa hash table é necessário saber o seu comprimento) e uma variável teste que é passada por endereço

1. Funcionamento geral:

.

* Estes conjuntos de letras vão ser comparados com o dicionário;
* Caso exista alguma palavra que tenho o início igual a alguma palavra que se encontre na lista, o programa segue para o vértice seguinte e a variável teste não é atualizada, caso não exista, a variável teste é atualizada e transforma-se como se fosse um sufixo, o que obriga o programa a voltar para o vértice anterior;
* Caso se encontre alguma palavra que exista no dicionário, esta é criada e enviada para a função que a chamou;

1. Funcionamento específico:

* A função **devolve\_palavra\_resposta\_ao\_problema\_A**, vai comparar as palavras da lista de palavras com o dicionário consuante os seguintes critérios:
  + Se existe alguma palavra no dicionário que tenha até agora as mesmas letras que a palavra da lista;
  + Se o strlen da palavra for igual ao do apresentado no cabeçalho;
  + Se a palavra é igual à palavra do dicionário;

Se estes 3 critérios se verificarem a palavra devolve a palavra formada ao dicionário e segue para o vértice seguinte;

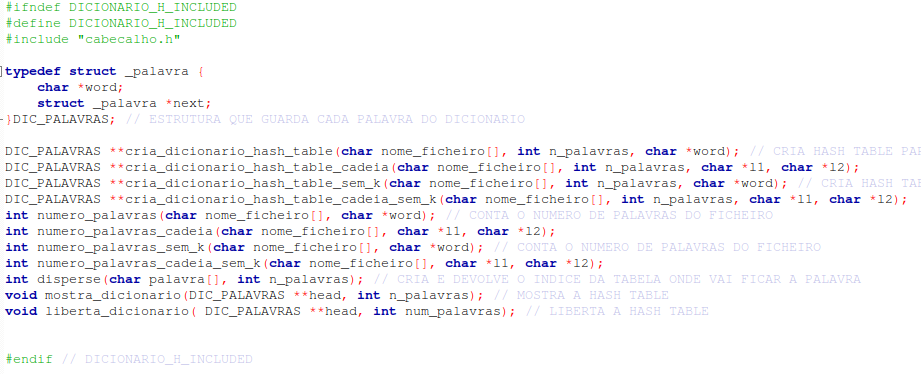
* A função **devolve\_palavra\_resposta\_ao\_problema\_D**, vai comparar as palavras da lista de palavras com o dicionário consuante os seguintes critérios:
  + Se existe alguma palavra no dicionário que tenha até agora as mesmas letras que a palavra da lista;
  + Se a palavra é igual à palavra do dicionário;

Se estes 2 critérios se verificarem a palavra devolve a palavra formada ao dicionário e segue para o vértice seguinte;

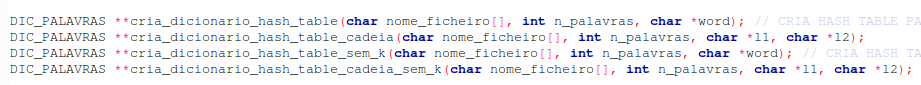
* A função **devolve\_palavra\_resposta\_ao\_problema\_F**, vai comparar as palavras da lista de palavras com o dicionário consuante os seguintes critérios:
  + Se existe alguma palavra no dicionário que tenha até agora as mesmas letras que a palavra da lista;
  + Se o valor da palavra for igual ao do apresentado no cabeçalho;
  + Se a palavra é igual à palavra do dicionário;

Se estes 3 critérios se verificarem a palavra devolve a palavra formada ao dicionário e segue para o vértice seguinte;

FUNÇÕES DE DICIONARIO.H:



Estas funções, servem para criar o dicionário, que vai servir de base para a resolução das opções. De seguida , vamos exemplificar as funções que achamos mais complexas, com mais detalhe.



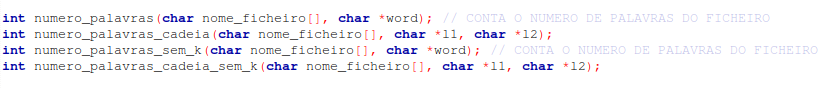
Todas estas funções, funcionam praticamente da mesma forma, o que apenas muda são pequenas alterações nos if(), que têm que ser feitas, devido à natureza do que é pretendido.

1. Funcionemento geral:

* As funções vão receber como argumentos, o ficheiro do dicionário, o número total de palavras que estas vão possuir (muda conforme as condições: vértice de partida e o valor k) e a/ou letra/as que se encontra no vértice de partida;
* Vai se alocar então uma hash table com índice == n\_palavras. Como se sabe que cada palavra é única, assim que se encontra uma palavra para introduzir na hash table, o índice aumenta, ficando assim as palavras dispersas um por índice, o que leva à melhor performance;
* De seguida, vai se correr o programa, de forma a que quando se encontra uma palavra que verifique as condições, esta vai ser introduzida na hash table e por fim, retorna-se essa dita hash table;

1. Funcionamento específico:

* A função cria\_hash\_table e cria\_hash\_table\_sem\_k, são idênticas, só vão receber um conjunto de letras, isto é o vértice inicial não possui uma cadeia de letras. A única diferença , é que a primeira, tem em conta que só vai buscar as palavras que começam por aquelas determinadas letras e que possuem tamanho k, ao passo que a segunda, apenas vai buscar as palavras que começam por aquelas determinadas letras;
* Relativamente à função cria\_hash\_table\_cadeia e cria\_hash\_table\_cadeia\_sem\_k, estas também são idênticas, só que vão receber dois conjunto de letras, isto é o vértice inicial possui uma cadeia de letras, logo as palavras podem começar de duas formas distintas. A única diferença entre elas , é que a primeira, tem em conta que só vai buscar as palavras que começam por aquelas determinadas letras e que possuem tamanho k, ao passo que a segunda, apenas vai buscar as palavras que começam por aquelas determinadas letras;



Todas estas funções, funcionam praticamente da mesma forma, o que apenas muda são pequenas alterações nos if(), que têm que ser feitas, devido à natureza do que é pretendido.

1. Funcionamento geral:

* As funções vão receber como argumentos o ficheiro do dicionário e a/ou letra/as que se encontra no vértice de partida;
* Nestas funções, vai se correr o dicionário e, sempre que se encontram palavras que verifiquem as condições necessárias, aumenta-se o contador;
* Por fim, retorna-se o contador, que corresponde ao número total de palavras que verificam as condições impostas e que vão ser usadas para criar a hash table do DICIONARIO;

1. Funcionamento específico:

* A função numero\_palavras e numero\_palavras\_sem\_k, são praticamente idênticas, estas vão receber um conjunto de letras, isto é o vértice inicial não possui uma cadeia de letras. A única diferença , é que a primeira, tem em conta que só vai contar as palavras que começam por aquelas determinadas letras e que possuem tamanho k, ao passo que a segunda, apenas vai contar as palavras que começam por aquelas determinadas letras;
* A função numero\_palavras\_cadeia e numero\_palavras\_cadeia\_sem\_k, são praticamente idênticas, estas vão receber dois conjuntos de letras, isto é o vértice inicial possui uma cadeia de letras. A única diferença , é que a primeira, tem em conta que só vai contar as palavras que começam por aquelas determinadas letras e que possuem tamanho k, ao passo que a segunda, apenas vai contar as palavras que começam por aquelas determinadas letras;

FUNCIONAMENTO DO PROGRAMA:

O nosso programa, começa no ficheiro wordz.c. Aqui, lemos, do terminal as instruções que vão permitir correr o programa. Primeiramente, verificamos se o número de argumentos, do terminal está correto. Se sim segue-se o programa, se não aborta-se logo. De seguida, vamos abrir o ficheiro dos grafos, e criar as estruturas que permitem resolver o problema em mão. Para guardar os vertices e os dados que lá estão, vamos alocar um vetor de estruturas VERTICES,de tamanho V (número de vertices retirado do cabeçalho). De seguida, alocamos uma matriz de adjacências, de tamanho V\*V, de forma a guardarmos as arestas do problema. Optamos por fazer desta forma, uma vez que para problemas de grande tamanho , compensa em termos de memória alocarmos esta matriz, visto que para verificarmos a existência de uma aresta, apenas tem uma complexidade de 1, e visto que esse vai ser o objetivo central deste programa, é o ideal. Uma vez as estruturas feitas, estas são enviadas para a função **opcoes.**

Na função **opcoes,** apenas se vai direcionar, consoante a opção retirada do cabeçalho, para uma das funções que visa resolver essa dada opção ( opcaoA1, opcaoF2, … ). Caso não exista a opção que se leu do ficheiro dos grafos, o programa termina, e o resultado final é -1.

Por sua vez, é numa destas funções que se vai tentar encontrar a resposta ao dado cabeçalho. Como todas as funções são banstante parecidas, apenas muda umas quantas condições.

Vamos começar por inicializar a 0 um vetor de nome vert\_vistos, de tamanho V (número total de vertices). Este vértice, vai servir, para sabermos, num dado ponto do programa, quais vertices é que já visitamos e quais é que ainda vamos visitar, ou seja, os vertices presentes no caminho atual. Desta forma sempre que visitamos um novo vértice, introduzimos um 1 na sua posição neste vetor, e assim que voltamos atrás desse vértice, voltamos a meter 0, para sabermos que nesse novo caminho, ainda não visitamos esse vértice. Optamos por fazer desta forma, pois para verificarmos numa condição se o vértice já está visto ou não, apenas temos que aceder à sua posição neste vetor. De seguida, vamos alocar uma variável de nome caminho\_vert. Este vetor que vai servir como uma espécie de stack, vai ser o nosso caminho concreto. Neste vetor, encontra-se por ordem os vetores do caminho, assim se se visitar um novo vértice, realloca-se o vetor e introduz-se, se quisermos andar um vértice para trás, já sabemos, para qual temos que ir, devemos de ir para o vértice que se encontra na posição anterior. Também vamos ao criar um vetor de nome caminho\_char, que vai funcionar da mesma forma, mas este tem o intuito de guardar o número de letras que se introduz em cada vértice, para podermos saber quando voltamos um vértice para trás, quantas letras é que introduzimos, para podermos realocar as palavras da lista de palavras.

Tendo agora ideia de quais as estruturas presentes no programa, bem como o funcionamento das funções usadas, vamos fazer uma breve explicação. Para cada vértice, vamos retirar as letras que lá se encontram . Vamos então juntar essas novas letras à lista de palavras e compara-las com o dicionário. Caso se encontrem as condições necessárias para se proseguir com o dado caminho, atualiza-se todas as estruturas e continua-se para o vértice seguinte. Caso as condições não se verifiquem, volta-se para os vertices anteriores, de forma a encontrarmos uma nova aresta. Se não se encontrar aresta nenhuma dá-se o programa por terminado e escreve-se no ficheiro de saída o resultado, caso se encontre segue-se novamente para esse vértice, volta-se a atualizar as estruturas e comparar novamente com o dicionário. Seguindo-se desta forma, até se acabarem as arestas.

No caso específico para as opçoes X2, faz –se um ciclo for{}, que corra o programa de forma a que o vértice inicial seja o 1, depois o 2,3,…, V.