Universidade de Brasília
Disciplina: Métodos de Programação
Professor: Jan Correa

Aluno: Rafael Lourenço de Lima Chehab
Matrícula: 15/0045123

Será feito uma estrutura de dados grafo que contêm vértices, origens e arestas.

Deverá ser possível ler o grafo de um arquivo, escrever no mesmo formato em um arquivo. Ademais, deve-se poder inserir e remover vértices, origens e arestas, achar o menor caminho de um vertice inicial e um final e verificar se, a partir das origens, pode se chegar a todos os vértices do grafo.

Observações: toda utilização de um grafo nas funções serão de ponteiros de grafo, ou seja, sempre que for passado ou recebido um grafo, entenda-se um ponteiro para a estrutura de dados grafo.
Todas as funções passaram em todos os testes feitos/mencionados.

Estrutura de Dados: grafo

Será criado uma estrutura de dados grafos que possua uma cabeça de estrutura de dados. A partir dessa cabeça deve-se poder fazer tudo que foi especificado acima.

Função: criar_grafo

Aloca a memória necessária para o grafo e o inicializa. Um cliente **sempre** deve chamar essa função logo após declarar o grafo, e ela deverá criar um grafo vazio, sem vértices, arestas, nem origens.

A função não terá parâmetros. Um ponteiro para esse grafo vazio será o retorno da função

Teste 1: criacao

Será testado se na criação o grafo aponta para um endereço não vazio que é um grafo, sem nenhum vértice, origem ou arestas.

Se executará a função criar_grafo e se verificará o valor de retorno e o número de vértices e arestas do grafo que deverão ser 0.

Função: deletar_grafo

Desaloca a memória necessária para o grafo. Um cliente deve chamar essa função logo após o último uso do grafo, e ela deverá deletar todos os vértices, origens e arestas, caso existam. O usuário poderá usar a função independente da existência de vértices, origens ou arestas, contudo assume-se que o grafo tenha sido inicializado com criar_grafo.

Essa função receberá como parâmetro somente o grafo do usuário. Ela retornará um ponteiro para o grafo após a deleção, que será **sempre nulo**. É de suma importância a atribuição desse valor para o grafo para evitar que o usuário tente utilizar um grafo deletado.

```
meu_grafo = deletar_grafo(meu_grafo);
```

, tal que meu_grafo é um ponteiro para o grafo.

Caso se queira utilizar novamente a variável que foi passada para essa função, ela deverá receber criar_grafo.

Teste 1: delecao_grafo

No teste se criará um grafo e se deletará, logo se verificará se o valor do grafo é nulo. Observação: todos os testes criarão e deletarão os grafos, assim cada um deles equivale à um teste da deleção

Função: existe_vert

Essa função verifica a existência de um vértice em um grafo. Estes serão os parâmetros recebidos: um grafo já inicializado com criar_grafo, e um vértice. Os vértices do grafo serão strings de tamanho até 100, nenhum nome poderá ser iniciado por espaço, e não poderá haver vírgulas ou nova linha no nome do vértice.

Assume-se que o nome dado respeita as condições acima. Caso esse vértice esteja no grafo se retornará TRUE, caso contrário se retorna FALSE.

Teste 1: vertices_nao_existentes

Será testado se, ao se perguntar da existência de vértices que não foram adicionados ao grafo, a resposta será FALSE, valor esperado

Teste 2: vertices_existentes

Similar ao teste anterior adiciona mesmos vértices usados no exemplo anterior e em seguida usa existe_vert. Nesse caso a resposta deverá ser TRUE para todos eles.

Função: existe_origem

Essa função verifica a existência de uma origem em um grafo. Os seus parâmetros serão iguais aos da função existe_vert. Caso esse vértice esteja no grafo e seja uma origem se retornará TRUE, caso contrário se retorna FALSE.

Teste 1: origens_inexistentes

Será testado se, ao se perguntar da existência de origens que não foram adicionados ao grafo, a resposta será FALSE, valor esperado. Contudo todos os nomes serão vértices do grafo.

Teste 2: vertices_inexistentes

Será testado se, ao se perguntar da existência de origens que não foram adicionados ao grafo, a resposta será FALSE, valor esperado. Nenhum dos nomes passados às perguntas serão nem vértices nem origem.

Teste 3: origens_existentes

Similar ao teste anterior adiciona os mesmos vértices e origens usados no exemplo anterior e em seguida usa existe_origem. Nesse caso a resposta deverá ser TRUE para todos eles.

Teste 4: origens_existentes_e_inexistentes

Teste tudo testado nos três anteriores, porém em um mesmo grafo.

Função: existe_aresta

Essa função verifica a existência de uma aresta em um grafo. Estes serão os parâmetros recebidos: um grafo já inicializado com criar_grafo, vértice inicial da aresta e vértice final da aresta. Será procurada uma aresta direcionada iniciando no primeiro vértice e terminando no segundo.

Assume-se que os nomes dos vértices dados respeitam as normas de nomeação de um vértice descritas em existe_vert. Caso esses vértice estejam no grafo e haja uma aresta entre eles se retornará

TRUE, caso contrário se retorna FALSE.

Teste 1: vertices_inexistentes

Será testado se, ao se perguntar da existência de arestas que não foram adicionados ao grafo, a resposta será FALSE, valor esperado. Nenhum dos nomes passados às perguntas serão vértices.

Teste 2: arestas_inexistentes

Será testado se, ao se perguntar da existência de origens que não foram adicionados ao grafo, a resposta será FALSE, valor esperado. Contudo os nomes passados às perguntas serão vértices, que não estarão ligados por arestas.

Teste 3: arestas_existentes

Similar ao teste anterior adiciona arestas à aquilo usado no exemplo anterior e em seguida usa existe_aresta. Nesse caso a resposta deverá ser TRUE para eles. Ademais, ao final se testa se o inverso de uma aresta é verdade, ou seja, se existe uma aresta de C para D verifica se há uma de D para C, como esperado esse resultado é falso, uma vez que as arestas são direcionais.

Teste 4: peso invalido

Similar ao teste anterior porém as arestas adicionadas tem peso negativo ou nulo. Assim espera-se que a resposta seja FALSE, e que essas arestas não tenham sido adicionadas.

Função: inserir_vert

Essa função irá inserir um vértice em um grafo. Estes serão os parâmetros recebidos: um grafo já inicializado com criar_grafo, e um vértice que segue as regras de nomeação de um vértice descritas em existe_vert. Caso o vértice já esteja no grafo, ele não será adicionado, isto é não haverão dois vértices com nomes iguais, poderá ser escrita uma mensagem de erro ao arquivo padrão de erro stderr, caso DEBUG seja setado na compilação, para mais detalhes veja o código. Isso possibilita que um cliente da biblioteca perceba os erros e que também se for desejado escreva-os em algum laudo ou arquivo sem atrapalhar um usuário do serviço que ele oferece.

Assume-se que o nome dado respeita as condições acima e que o grafo já tenha sido alocado e inicializado, outra assertiva de entrada é que a lista de vértices é consistente. O grafo modificado pela função sairá também consistente e a única modificação é, possivelmente, a inserção de um vértice.
A função não retorna diretamente nenhum valor.

Teste 1: insercao

Será testado se vértices foram realmente adicionados. Para isso a função existe_vert verificará. Espera-se que vértices adicionados retornem TRUE em existe_vert e qualquer outra string que se passe retorne FALSE.

Função: inserir_origem

Essa função irá inserir uma origem em um grafo. Estes serão os parâmetros recebidos: um grafo já inicializado com criar_grafo, e um vértice que segue as regras de nomeação de um vértice descritas em existe_vert. Caso o vértice não esteja no grafo ou caso ele já seja uma origem, ele não será adicionado, isto é não haverão duas origens com nomes iguais, poderá ser escrita uma mensagem de erro ao arquivo padrão de erro stderr, caso DEBUG seja setado na compilação, para mais detalhes veja o código.

Se não se sabe se já há um vértice com o nome que se quer inserir nas origens, deve-se usar inserir_vert antes de inserir_origem. Assim, se já existir não ocorrerá nenhum problema, possivelmente uma mensagem de erro que pode ser ignorada, e se não existir é garantido que inserir_origem

adicionará a origem.

Assume-se que o nome dado respeita as condições acima e que o grafo já tenha sido alocado e inicializado, outra assertiva de entrada é que a lista de vértices é consistente. O grafo modificado pela função sairá também consistente e a única modificação é, possivelmente, a inserção de uma origem.

A função não retorna diretamente nenhum valor.

Teste 1: vertice_existentes

Se tentará adicionar origens que já foram adicionadas como vértices. Se espera que, ao se utilizar existe_origem, o retorno seja TRUE, para todos os outros vértices do grafo a resposta deve ser FALSE, e para tudo aquilo que não for nem vértice seja FALSE.

Teste 2: vertice_inexistentes

Tenta-se adicionar origens que não são vértices do grafo. Se espera que, ao se utilizar existe_origem, o retorno seja FALSE.

Função: inserir_aresta

Essa função irá inserir uma aresta direcionada em um grafo. Estes serão os parâmetros recebidos: um grafo já inicializado com criar_grafo, o vértice inicial e o final da aresta quem segue as regras de nomeação já descritas em existe_vert e o peso da aresta que deve ser um valor maior que 0.

Caso pelo menos um dos vértices não esteja no grafo ou caso já haja uma aresta entre eles, ele nada será adicionado, isto é não haverão duas arestas saindo de um vértice e chegando ao mesmo, não será possível criar um multigrafo. Nada impede que haja um loop, ou seja uma aresta de A para A, por exemplo. Se não se adicionado poderá ser escrita uma mensagem de erro ao arquivo padrão de erro stderr, caso DEBUG seja setado na compilação, para mais detalhes veja o código.

Se não se sabe se já há os vértices existem, vale utilizar a técnica mencionada em inserir_aresta para ambos os vértices.

Assume-se que os nomes dado respeitam as condições acima, que o grafo já tenha sido alocado e inicializado, outra assertiva de entrada é que a lista de vértices é consistente e que o peso seja maior que 0. O grafo modificado pela função sairá também consistente e a única modificação é, possivelmente, a inserção de uma aresta.

A função não retorna diretamente nenhum valor.

Teste 1: vertice_inexistentes

Tenta-se adicionar arestas cujos início e fim não são vértices do grafo. Se espera que, ao se utilizar existe_aresta, o retorno seja FALSE.

Teste 2: vertice_existentes

Tenta-se adicionar arestas cujos início e fim são vértices do grafo. Se espera que, ao se utilizar existe_aresta, o retorno seja TRUE, e para tudo aquilo que não for nem vértice seja FALSE, ou para o que seja vértice porém sem um inserir_aresta seja FALSE.

Função: ler_grafo

A função deverá ler o grafo de um arquivo escrito no formato a seguir:

A primeira linha contem os vértices do grafo que são strings de tamanho Maximo 100, separados por vírgulas
A segunda linha contem as origens (strings de tamanho Maximo 100), separados por vírgulas

As linhas seguintes são as arestas direcionadas com vértice de origem, destino e o peso que é um número real.
Por exemplo:

A, B, C
A, B
A, B, 3.0
B, C, 4.0

Para isso o trabalho da função será dividida em outras três que lerão, respectivamente, vértices, origens e arestas, ou, de outra maneira, primeira linha, segunda linha e outras linha.

A função receberá como parâmetros um grafo, já inicializado com criar_grafo, podendo ou não já ter vértices, origens e arestas e o nome do arquivo que contenha os dados, o arquivo deve existir e seguir o formato dado acima. ler_grafo adicionará dados porém não eliminará nenhum dado do grafo antes da leitura, fica à cargo do cliente decidir se o grafo passado será vazio. Isso permite por exemplo que um grafo grande seja dividido em arquivos, cuidado porém que se em um arquivo houver uma aresta com um vértice definido em outro arquivo.

Caso o arquivo não esteja no formato dito o resultado será imprevisível, o mesmo vale para um arquivo sem nenhum vértice e nenhuma origem, embora não haja nenhum problema em um arquivo sem arestas. Para que a função ler_grafo funcione como esperada deve-se dar pelo menos um vértice e uma origem, eles podem já pertencer ao grafo, assim nada será adicionado, ou pode ser no caso da origem um nome inválido que não seja vértice e que portanto não será adicionado.[

Garante-se que a saída será um grafo consistente, se a entrada também o for.

As funções ler_vert, ler_origem e ler_aresta serão as divisões de ler_grafo. Uma será executada onde a outra parou. Eles farão a leitura e usarão inserir_vert, inserir_origem e inserir_aresta internamente.

Teste 1: grafo_do_exemplo

Mostra o uso dessa função para o grafo que foi utilizado para explicar a formatação. Como escrever_grafo, função explicada posteriormente, escreve em um arquivo, não há um teste mesmo sendo feito. Deve-se olhar os arquivos e verificar que geram o mesmo grafo.

Teste 2: outras_aplicacoes

Utiliza ler_grafo mais de uma vez e mostra como os grafos dos diferentes foram “somados”.

Função: remover_vert

Essa função irá remover um vértice em um grafo. Estes serão os parâmetros recebidos: um grafo já inicializado com criar_grafo, e um vértice que segue as regras de nomeação de um vértice descritas em existe_vert.

Caso o vértice não esteja no grafo ele não será removido.

Se ele for origem, a origem será retirada, e todas as arestas chegando ou partindo dele serão retiradas.

Assume-se que o nome dado respeita as condições acima e que o grafo já tenha sido alocado e inicializado, outra assertiva de entrada é que a lista de vértices é consistente. A saída será um grafo consistente, onde serão retiradas todas as referências à esse vértice.
A função não retorna diretamente nenhum valor.

Teste 1: vertice_existentes

Testa o remoção de um vértice inserido no grafo, a resposta à existe_vert deve ser FALSE, valor esperado.

Teste 2: vertice_inexistentes

Testa o remoção de um vértice que não fora inserido no grafo, a resposta à existe_vert deve ser FALSE, valor esperado e não deve interromper a execução do programa.

Teste 3: vertice_existente_com_origem_e_arestas

Similar ao primeiro teste porém alguns vértices são origens e possuem arestas saindo ou chegando neles. Nesse caso qualquer pergunta que possua referência à um vértice deletado deve retornar FALSE.

Função: remover_origem

Essa função irá remover uma origem em um grafo. Estes serão os parâmetros recebidos: um grafo já inicializado com criar_grafo, e um vértice que segue as regras de nomeação de um vértice descritas em existe_vert.

Caso a origem não esteja no grafo ou não seja um vértice, nada será feito.

Essa origem continuará sendo vértice e suas arestas não serão modificadas.

Assume-se que o nome dado respeita as condições acima e que o grafo já tenha sido alocado e inicializado. A saída será um grafo consistente.

A função não retorna diretamente nenhum valor.

Teste 1: vertice_inexistente

Testa o remoção de uma origem que não fora inserida no grafo, e nem era vértice, a resposta à existe_origem deve ser FALSE, valor esperado e não deve interromper a execução do programa.

Teste 2: vertice_existente

Testa o remoção de uma origem inserida no grafo, a resposta à existe_origem deve ser FALSE, valor esperado e a resposta à existe_vert deve ser TRUE, visto que a remoção de uma origem não remove um vértice.

Função: remover_aresta

Essa função irá inserir uma aresta direcionada em um grafo. Estes serão os parâmetros recebidos: um grafo já inicializado com criar_grafo, o vértice inicial e o final da aresta quem segue as regras de nomeação já descritas em existe_vert. **Não** se recebe o peso, perceba que para todas as outras funções aquilo que se passa na hora de inserir é igual à remoção, nessa função esse argumento não será igual, uma vez que o usuário pode desconhecer o peso, que não influencia na remoção.

Caso pelo menos um dos vértices não esteja no grafo ou caso não haja uma aresta entre eles, nada será removido.

Assume-se que os nomes dado respeitam as condições acima, que o grafo já tenha sido alocado e inicializado. O grafo modificado pela função sairá também consistente e a única modificação é, possivelmente, a remoção de uma aresta.

A função não retorna diretamente nenhum valor.

Teste 1: vertice_inexistente

Testa o remoção de uma aresta cujos início e fim não foram inseridos no grafo, a

resposta à existe_aresta deve ser FALSE, valor esperado e não deve interromper a execução do programa.

Teste 2: aresta_existente

Testa o remoção de arestas inseridas no grafo, a resposta à existe_aresta deve ser FALSE, valor esperado e a resposta à existe_vert deve ser TRUE, visto que a remoção de arestas não remove vértices.

Função: escrever_grafo

A função deverá escrever o grafo em um arquivo no formato definido em ler_grafo.

Para isso o trabalho da função será dividida em outras três que escreverão, respectivamente, vértices, origens e arestas, ou, de outra maneira, primeira linha, segunda linha e outras linha.

A função receberá como parâmetros um grafo, já inicializado com criar_grafo, podendo ou não já ter vértices, origens e arestas, não havendo muita utilidade para grafos vazios, se o grafo não tiver vértices (vazio), ou origens a função não gerará problema, porém se isso for posteriormente lido (ler_grafo) podem haver problemas.

Garante-se que a saída não alterará o grafo e o arquivo terá o mesmo formato de ler_grafo, e que se houver pelo menos um vértice e uma origem, ler_grafo gerará outro grafo de mesmo conteúdo.

As funções escrever_vert, escrever_origem e escrever_aresta serão as divisões de escrever_grafo. Uma será executada onde a outra parou.

Teste 1: grafo_do_exemplo

Mostra o uso dessa função para o grafo que foi utilizado para explicar a formatação em ler_grafo. Como essa função escreve em um arquivo, não há um teste mesmo sendo feito. Deve-se olhar os arquivos e verificar que geram o mesmo grafo.

Teste 2: outras_aplicacoes

Utiliza ler_grafo mais de uma vez e mostra como os grafos dos diferentes foram “somados”.

Função: num_vert

Retorna o número de vértices do grafo.

A função receberá um ponteiro para um grafo já alocado e inicializado e retornará um número maior ou igual à zero.

Função: num_arestas

Aloca a memória necessária para o grafo e o inicializa. Um cliente **sempre** deve chamar essa função logo após declarar o grafo, e ela deverá criar um grafo vazio, sem vértices, arestas, nem origens.

A função não terá parâmetros. Um ponteiro para esse grafo vazio será o retorno da função

Função: menor_caminho

Estes são os parâmetros: grafo já alocado e inicializado, nome do vértice onde se inicia o caminho que se deve procurar e nome do vértice onde deve terminar o caminho. Retorna o tamanho de menor caminho entre os dois ou -1 caso não exista um caminho no grafo, ou se pelo menos um deles não for um vértice. Um caminho iniciando e terminando num mesmo vértice tem tamanho 0.

Assume-se que os nomes dado respeitam as condições acima, que o grafo já tenha sido alocado e inicializado. Assume-se internamente que todo caminho possui tamanho menor ou igual à 2 vezes (10 elevado à 40), embora esse parâmetro seja ajustável. Caso o caminho seja maior que esse valor, a resposta apontará erroneamente que não há um caminho entre eles.

O grafo não será modificado pela função.

Teste 1: caminho_existente

Verifica-se o caminho retornado é o menor possível, utiliza o grafo do exemplo de ler_grafo.

Teste 2: caminho_inexistente

Verifica-se retorna-se -1 para caminhos inexistentes, sendo que os vértices podem não pertencer ao grafo, ou caso existam, pode não haver caminho utiliza o grafo do exemplo de ler_grafo.

Função: eh_conexo

Passa-se um grafo já alocado e inicializado. A função retorna se é possível partindo das origens, chegar à todos os vértices. Define-se que um grafo sem nenhum vértice, e logo vazio (sem nenhuma origem nem aresta) é conexo.

Se o grafo for conexo retorna-se TRUE, caso contrário retorna-se FALSE.

Teste 1: conexo

Verifica que um grafo vazio é conexo, eu o grafo do exemplo é conexo e que ao se adicionar um vértice que também é origem à um grafo conexo esse também será conexo mesmo se nenhuma aresta apontar para esse novo vértice. O retorno esperado é TRUE.

Teste 2: nao_conexo

Verifica-se retorna-se FALSE para grafos não conexos, para isso adiciona-se um vértice inalcançável ao grafo do exemplo de ler_grafo.