Laboratório 12: Caminhos Mais Curtos

Entrega até domingo, 16/6, às 23:59h

Neste laboratório, vamos implementar dois métodos para calcular os caminhos mais curtos a partir de um vértice com todos os outros vértices do grafo. O arquivo **grafo.c**, que você deve completar, contém parte da implementação vista em aula e nos laboratórios anteriores, utilizando a representação de um grafo por lista de adjacências. O arquivo de teste lê cada um dos arquivos passados e chama os métodos que serão implementados. Nos arquivos do trabalho, o arquivo saida.txt mostra o resultado esperado para cada um dos grafos.

Cada método pode utilizar alguns arrays auxiliares importantes para que os caminhos mínimos sejam calculados corretamente:

a) O array **cmccusto** guarda o custo total para se chegar até um vértice a partir do nó inicial informado *no_inicial*. Sabemos que, no caso do nó inicial, cmccusto[*no_inicial*] = 0.
b) O array **cmcvant** contém o antecessor do caminho mais curto associado a um vértice. Por exemplo: se o caminho mais curto para se chegar até o vértice 6 é 0 – 1 – 4 – 6 (considerando *no_inicial* = 0), então cmcvant[6] = 4, cmcvant[4] = 1, e cmcvant[1] = 0. Por fim, cmcvant[*no_inicial*] = *no_inicial*.
c) O array **visitados** indica os vértices que já foram processados. Nesse caso, o seu estado muda após realizar a avaliação da sua respectiva lista de adjacentes. Lembre-se que, como o nó inicial já está na solução, pois seu custo é 0 e o seu caminho já é conhecido, então o array é inicializado com visitados[*no_inicial*] = 1.

Nos métodos a seguir, você irá perceber que o passo fundamental consiste no relaxamento de arestas, com o objetivo de atualizar, a cada iteração, o custo do caminho mais curto para cada vértice do grafo. Ao implementar os métodos das questões 2, 3 e 4, remova a primeira linha de retorno (**return NULL;**), que foi adicionada apenas para o teste não entrar em loop infinito. Cada método cria e retorna um array **cmcvant**, e recebe como parâmetro o ponteiro para o vetor **cmccusto** (já inicializado no teste para cada uma das chamadas).

Para compilar o programa de teste, utilize o comando **gcc -o t teste.c grafo.c heap.c**.

A partir disso, faça as seguintes questões:

1. (1.0) Termine a implementação da função **relaxa**, vista em sala de aula. Ao invés de receber um grafo, a função recebe os arrays **cmccusto** e **cmcvant**, já descritos anteriormente. Este método deve testar se o custo atual para chegar até o vértice **j** é maior do que a soma do custo de chegar até **i** com a aresta **(i,j)**. Em caso afirmativo, deve-se atualizar o custo do caminho mínimo para chegar até **j** pelo custo de **i** mais o custo da aresta **(i,j)**. É fundamental também atualizar o vetor **cmcvant**, para que o teste possa recuperar o caminho mais curto ao vértice **j**.

2. (2.0) Termine a implementação do método **cmcBellmanFord**. A cada iteração, cada vértice verifica se é possível atualizar o caminho mínimo de cada um dos seus vizinhos, a partir do processo de relaxamento de arestas. Ao repetir nv-1 vezes, temos o caminho mais curto para cada vértice a partir de um nó inicial (não se preocupe em verificar se ocorreu alguma relaxação durante uma iteração para terminar o algoritmo antes das nv-1 passadas). Neste laboratório não estamos considerando casos de ciclos com peso total negativo, por isso o processo é repetido apenas nv-1 vezes. Utilize o método **relaxa** da questão 1 para atualizar o caminho mais curto para cada vértice.

3. (3.0) Termine a implementação do método **cmcDijkstraBuscaLinear**. Nesse método, a cada iteração, todos os vizinhos do vértice corrente podem ter os seus custos de caminho mínimo atualizados a partir do processo de relaxação de arestas. Ao final, verifica-se qual é o vértice com o caminho de menor custo para a próxima iteração.
   Para achar o próximo vértice, uma busca linear é feita. A iteração é repetida até que todos os vértices com caminho possível sejam visitados. Utilize o array **visitados**, para indicar que o nó corrente foi avaliado, dessa forma ele não será revisitado no futuro. Utilize o método **relaxa** da questão 1 para atualizar o caminho mais curto para cada vértice.

4. (4.0) Por fim, termine a implementação do método **cmcDijkstraBuscaHeap**, que consiste no mesmo método da questão 3, porém utilizando um heap ao invés da busca linear para achar o próximo vértice que deve ser visitado. Nessa implementação, cada item do heap corresponde a um nó ainda não visitado, e o seu custo de caminho mínimo até então conhecido (inicialmente será INT_MAX para todos os vértices adicionados no heap). Lembre-se que o vértice inicial **não** deve ser adicionado no heap, pois já sabemos que seu custo é 0. Quando ocorre o relaxamento de alguma aresta (você pode utilizar o retorno da função **relaxa** para verificar isso), chame a função **heap_corrige** passando o novo custo total (armazenado em **cmccusto**) e o seu índice, que atualiza a prioridade e a posição do vértice no heap.

   Obs: A função heap_remove retorna -1 caso o heap esteja vazio, e o índice do próximo nó com menor custo mínimo conhecido caso o heap não esteja vazio. Use isso para verificar quando terminar o laço de repetição.

Faça upload do arquivo **grafo.c** no EAD até dia 16 de junho, domingo, às 23:59h. Lembre-se de fazer a entrega mesmo que não tenha chegado ao final do exercício.