Laboratório 11: Árvore Geradora Mínima (MST)

Entrega até domingo, 9/6, às 23:59h

Neste laboratório, vamos implementar dois métodos para calcular a árvore de custo mínimo. O arquivo de teste lê cada um dos arquivos passados (grafo1.dat, grafo2.dat, grafo3.dat) e chama ambos os métodos que serão implementados. 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.

Para implementar os dois algoritmos, utilize a estrutura de MinHeap implementada em heap.c e heap.h. Perceba que a interface foi modificada para que cada inserção receba um custo (que será a prioridade do item adicionado), e dois vértices. Na remoção, os dois vértices do item removido são repassados através de endereços passados como parâmetros da função. Lembre-se que o modo de funcionamento do Heap é diferente para os dois algoritmos que iremos implementar.

Para rodar o programa de teste, utilize o comando **gcc -o teste teste.c grafo.c heap.c ub.c**.

A partir disso, faça as seguintes questões:

1. (4.0) Termine implementação do método de Prim *primArvoreCustoMinimo*, que recebe como parâmetro um grafo de entrada e o vértice de onde a geração da árvore deve ser iniciada. O método retorna um ponteiro para um novo grafo representado a árvore de custo mínimo do grafo de entrada.

   Nesse método, precisamos de uma forma para modificar diretamente o custo de inserir um vértice na árvore geradora de custo mínimo. Para que isso seja possível, a estrutura de Heap implementada adiciona internamente um vetor auxiliar com o número de vértices do grafo (por isso, na sua inicialização, o Heap recebe 2 valores, o primeiro consistindo no tamanho máximo do heap, e o segundo definindo o tamanho do array auxiliar que guarda a posição de cada vértice no heap, para acesso em tempo constante). Use o método *heap_dimprioridade*, que modifica diretamente o custo de inserção de um vértice, e corrige o heap se for necessário (lembre-se de verificar antes se o vértice que será modificado existe no heap).

   Por fim, lembre-se de criar e usar um vetor auxiliar que verifica se um vértice já foi adicionado no heap, para que ele não seja adicionado novamente.

2. (6.0) Termine a implementação do método de Kruskal *kruskalArvoreCustoMinimo*, que recebe como parâmetro o grafo de entrada e retorna um novo grafo representando a árvore de custo mínimo do grafo de entrada.

Para implementar essa função, deve-se armazenar todas as arestas do grafo de entrada em um MinHeap, utilizando seu peso como prioridade. Lembre-se de não adicionar a mesma aresta duas vezes, pois estamos lidando com um grafo não direcionado. Você pode optar por adicionar uma aresta i-j apenas quando j > i.

Para verificar se uma aresta une duas componentes ainda não conexas, você deve utilizar a estrutura de união e busca. Inicialmente, essa estrutura representa nv conjuntos unitários (sendo nv o número de vértices). Quando uma nova aresta for adicionada no grafo (ou seja, quando os representantes de cada vértice de uma aresta forem diferentes), deve-se realizar a união das partições/conjuntos. A estrutura de união e busca está implementada nos arquivos ub.c e ub.h.

Faça upload do arquivo **grafo.c** no EAD até dia 9 de junho, domingo, às 23:59h. Lembre-se de fazer a entrega mesmo que não tenha chegado ao final do exercício.