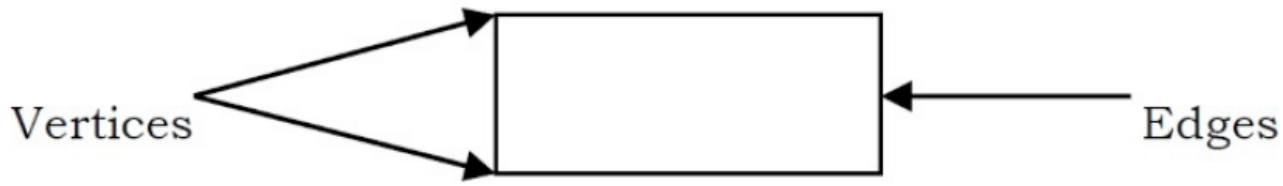
Universidade Federal do Maranhão
Departamento de Informática
Disciplina: ***Estrutura de Dados II 2019.1***
Prof.: ***João Dallyson Sousa de Almeida***

Atividade Prática 1
Valor: 30% da 1ª Avaliação
Entrega: 21/04/2019 (Individual)

1. DESCRIÇÃO

A árvore geradora (ou árvore de expansão) de grafo é um subgrafo que contém todos os vértices e também é uma árvore. Um grafo pode ter muitas árvores geradoras. Como exemplo, considere um grafo com 4 vértices como mostrado abaixo. Vamos supor que os cantos do grafo sejam vértices.



Para este grafo, nós podemos ter múltiplas árvores geradoras como ilustra a figura abaixo:

A árvore geradora mínima de um grafo não direcionado G é uma árvore formada por arestas do grafo que conectam todos os vértices de G com o menor custo (Somatório dos pesos). Existem dois algoritmos conhecidos que resolvem esse problema: Algoritmo de Prim e Algoritmo de Kruskal.

Neste trabalho será utilizado o algoritmo de Kruskal, por ter uma etapa que necessita da execução de um algoritmo de ordenação.

O algoritmo de Kruskal inicia com V diferentes árvores (V são vértices do grafo). Este algoritmo encontra a aresta segura (e com menor peso) procurando em todas as arestas que conectam quaisquer duas árvores na floresta A. A cada passo o algoritmo de Kruskal adiciona à floresta a aresta com menor peso possível.

O algoritmo de Kruskal utiliza conjuntos disjuntos, em cada conjunto contém os vértices de uma árvore na floresta atual. A operação FIND-SET(u) retorna um elemento representativo do conjunto que contém u . Assim, para determinar se dois vértices u e v pertencem à mesma árvore, basta testar se FIND-SET(u) é igual a FIND-SET(v). A combinação de árvores é executada com o procedimento UNION. A seguir temos o pseudoalgoritmo:

```
MST-KRUSKAL(G, w)
1  A ← ∅
2  for each vertex v ∈ V[G]
3       do MAKE-SET(v)
4  sort the edges of E into nondecreasing order by weight w
5  for each edge (u, v) ∈ E, taken in nondecreasing order by weight
6       do if FIND-SET(u) ≠ FIND-SET(v)
7            then A ← A ∪ {(u, v)}
8                 UNION(u, v)
9  return A
```

Para mais detalhes podem ser encontrados no capítulo 23 do livro, Algoritmos: teoria e prátrica (CORMEM et al., 2002) e nas notas de aula (slide 20, aula sobre Kruskal).

Será disponibilizada a implementação do algoritmo do Kruskal implementado em Python e Java.

Você deverá IMPLEMENTAR os algoritmos de ordenação estudados em sala de aula (**InsertSort, SelectSort, ShellSort, QuickSort, MergeSort HeapSort, CountSort**) para serem utilizados na etapa de ordenação das arestas realizada pelo algoritmo de Kruskal.. A especificação e desenvolvimento dos tipos abstratos de dados a serem utilizados no trabalho são de sua competência e fazem parte da avaliação.
Implemente o QuickSort usando como pivô: o primeiro elemento; o último elemento; elemento mediano entre o primeiro, central e último.

Você deverá carregar o vértices e arestas do grafo de através da leitura de arquivo.

Seu programa deverá ser executado passando-se opções de execução via linha de comando. Esse tipo de execução é bastante comum em sistemas Unix / Linux e no antigo DOS. Por exemplo, os parâmetros do programa podem ser definidos assim:

<Chamada ao Kruskal> <arquivo> <algoritmo ordenacao> <parâmetros ordenacao>

2. Descrição sobre os arquivos de dados

Serão utilizados arquivos de tamanho pequeno, média e grande definidos pela quantidade de arestas e vértices do grafo.

3. Análise dos resultados

A análise deve ser feita sobre o número de comparações, atribuições e tempo de execução dos algoritmos. Procure organizar inteligentemente os dados coletados em tabelas, e também construa gráficos a partir dos dados. Então, disserte sobre os dados nas tabelas e gráficos. Grande parte da avaliação será dedicada a análise dos resultados, ou seja, sobre o que você dissertar.

4. Entrega

- Código fonte do programa em C/C++ , Java ou Python (bem identado e comentado).
- Relatório do trabalho
- Upload no SIGAA.

O Relatório deve apresentar:

1. Introdução: descrição do problema a ser resolvido e justificativa apresentado exemplos de aplicações reais
2. Implementação: descrição sobre a implementação do programa. Devem ser detalhadas as estruturas de dados utilizadas (de preferência com diagramas ilustrativos), o funcionamento das principais funções e procedimentos utilizados, o formato de entrada e saída de dados, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado.
3. Análise de complexidade: apresentar o estudo de complexidade das funções implementadas e do programa como um todo (Execução do Kruskal) usando a notação O.
4. Testes: apresentação dos testes realizados.
5. Conclusão: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação
6. Referências: referências utilizadas no desenvolvimento do trabalho.