Atividade Prática A1 – Grafos (INE5413)

Ciências da Computação – Universidade Federal de Santa Catarina

Prof. Rafael de Santiago

Observações gerais:

- Trabalho deve ser executado em no máximo 3 estudantes da disciplina.
- Todas as codificações devem estar em uma das seguintes linguagens de programação: C/C++, Python ou Java.
- A biblioteca de grafos criada no primeiro item desse exercício deverá ser utilizada na codificação dos demais itens dessa atividade.
- A entrega do código-fonte deverá ser realizada no MOODLE[a] em um arquivo compactado no formato ZIP ou TAR.GZ.
- A atividade vale 12 pts. Se o grupo preferir, pode deixar de fazer um dos itens, exceto o item 4 (Relatório). As equipes que atingirem mais de 10 pts no trabalho, receberão nota 10 e o saldo será utilizado no próximo trabalho com nota inferior a 10 no semestre corrente.
- Duas ou mais equipes com trabalhos total ou parcialmente iguais receberão nota 0.

[a] A entrega deve ser realizada através do ambiente da turma no MOODLE.

1. [**Representação**] (2,0pts) Crie um tipo estruturado de dados ou uma classe que represente um grafo não-dirigido e ponderado $G(V, E, w)$, no qual V é o conjunto de vértices, E é o conjunto de arestas e $w : E \to \mathbb{R}$ é a função que mapeia o peso de cada aresta $\{u, v\} \in E$. As operações/métodos contemplados para o grafo deverão ser:

   - **qtdVertices()**: retornr a quantidade de vértices;
   - **qtdArestas()**: retorna a quantidade de arestas;
   - **grau(v)**: retorna o grau do vértice v;
   - **rotulo(v)**: retorna o rótulo do vértice v;
   - **vizinhos(v)**: retorna os vizinhos do vértice v;
   - **haAresta(u, v)**: se $\{u, v\} \in E$, retorna verdadeiro; se não existir, retorna falso;
   - **peso(u, v)**: se $\{u, v\} \in E$, retorna o peso da aresta $\{u, v\}$; se não existir, retorna um valor infinito positivo[1];
   - **ler(arquivo)**[2]: deve carregar um grafo a partir de um arquivo no formato especificado ao final deste documento.

   IMPORTANTE: As operações/métodos deverão ter complexidade de tempo computacional $O(1)$ quando possível. No caso de dúvidas, consulte o professor da disciplina.

2. [**Buscas**] (2,0pts) Crie um programa que receba um arquivo de grafo e o índice (de 1 a n) vértice s como argumentos. O programa deve fazer uma busca em largura[3] a partir de s e deverá imprimir a saída na tela, onde cada linha deverá conter o nível seguido de ":" e a listagem de vértices encontrados naquele nível. O exemplo abaixo trata de uma saída, na qual a busca se iniciou pelo vértice s no nível 0, depois prosseguiu nos vértices `3`, `4` e `5` para o próximo nível. No nível seguinte, a busca encontrou os vértices `1`, `2`, `6` e `7`.

```
0: 8
1: 3,4,5
2: 1,2,6,7
```

3. [**Ciclo Euleriano**] (2,0pts) Crie um programa que recebe um grafo como argumento. Ao final, o programa deverá determinar se há ou não um ciclo euleriano e exibí-lo na tela de acordo com o exemplo abaixo[4]. A primeira linha deverá conter o número `0` caso o grafo não contenha o ciclo euleriano. Caso contenha, deverá ser impresso `1` na primeira linha e em seguida, a sequência de vértices que corresponde ao ciclo deverá ser impressa.

[1] O valor infinito pode ser representado como o maior ponto flutuante positivo possível de ser caracterizado pelo tipo de dado selecionado.
[2] Para linguagens orientadas a objetos, a operação **ler(arquivo)** pode ser substituída por um construtor.
[3] Ignore os pesos do arquivo de entrada, pois a busca não precisará deles.
[4] Ignore os pesos do arquivo de entrada, pois não serão necessários.

```
1
2,4,3,1,5,6,2
```

4. [**Algoritmo de Bellman-Ford ou de Dijkstra**] (2,0pts) Crie um programa que recebe um arquivo de grafo como argumento e um vértice s. O programa deverá executar o algoritmo de Bellman-Ford ou de Dijkstra e informar o caminho percorrido de s até todos os outros vértices do grafo e a distância necessária. A saída deverá ser impressa na tela de acordo com o exemplo abaixo. Cada linha representa o caminho realizado de s para o vértice da respectiva linha. Em cada linha, antes dos símbolo ":" deverá estar o vértice destino. À direita de ":", encontra-se o caminho percorrido de s até o vértice destino. Mais à direita encontram-se os símbolos "d=" seguidos da distância necessária para percorrer o caminho.

```
1: 2,3,4,1; d=7
2: 2; d=0
3: 2,3; d=4
4: 2,3,4; d=6
5: 2,3,5; d=8
```

5. [**Algoritmo de Floyd-Warshall**] (2,0pts) Crie um programa que recebe um arquivo de grafo como argumento. O programa deverá exercutar o algoritmo de Floyd-Warshall e mostrar as distâncias para cada par de vértices na tela utilizando o formato do exemplo abaixo. Na saída, cada linha terá as distâncias para vértice na ordem crescente dos índices informados no arquivo de entrada.

```
1:0,10,3,5
2:10,0,9,8
3:3,9,0,11
4:5,8,11,0
```

6. [**Relatório**] (2,0pts) Elabore um relatório de uma página comentando para cada um dos exercícios quais as estruturas de dados selecionadas, justificando as escolhas. Não esqueça de informar os nomes dos integrantes da equipe.

Padrão de Arquivo de Entrada

O arquivo de entrada deve estar no formato abaixo. Na primeira linha, n é o número de vértices. Nas linhas seguintes e antes da palavra "*edges", há uma listagem de rótulos dos vértices. Note que cada vértice possui um índice de 1 à n. Esse índice é importante, pois ele é utilizado nas definições das arestas. Depois da palavra "*edges" cada linha conterá uma aresta. Por exemplo, na linha onde há "a b valor_do_peso", `a` e `b` são os vértices que a aresta conecta, `valor_do_peso` é o peso da aresta.

```
*vertices n
1 rotulo_de_1
2 rotulo_de_2
...
n label_de_n
*edges
a b valor_do_peso
a c valor_do_peso
...
```