Estrutura de Dados - Trabalho da disciplina

Prof. Ricardo Oliveira - 2020/2 - **APNP**

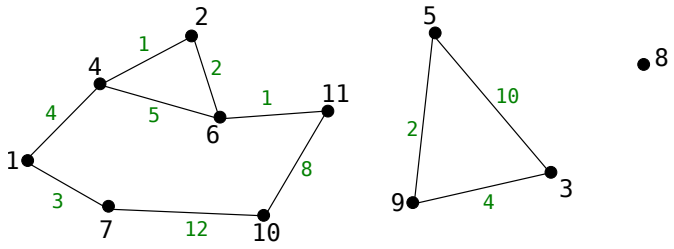
Enunciado

Escreva um programa em C que manipula um *grafo ponderado* (não direcionado), usando como estrutura a *lista de adjacência*.

Seu programa deve iniciar lendo um inteiro N, o número de vértices do grafo (considere que os vértices são identificados por inteiros entre 1 e N, inclusive). Inicialmente, o grafo não contém nenhuma aresta (isto é, o grau de todos os vértices é igual a 0). Após ler o valor de N, seu programa deve ler e processar uma sequência de comandos, onde cada comando pode ser:

- `+ A B w`: insere uma aresta de peso w entre os vértices A e B. Este comando não imprime nenhuma saída. Você pode assumir que $A \neq B$ e que esse comando nunca é chamado se já existe uma aresta entre A e B. Como exemplo, em um grafo com $N = 11$ vértices inicialmente sem arestas, os comandos dados abaixo (à esquerda) fazem do grafo o dado abaixo (à direita):

```
+ 1 4 4
+ 3 5 10
+ 11 6 1
+ 7 10 12
+ 6 2 2
+ 9 5 2
+ 4 6 5
+ 3 9 4
+ 10 11 8
+ 1 7 3
+ 4 2 1
```



- `w A B`: imprime o peso da aresta $\{A, B\}$. Caso a aresta não exista, imprime `aresta inexistente`. Como exemplo, no grafo dado anteriormente, o comando `w 5 3` deve imprimir `10`, enquanto o comando `w 6 1` deve imprimir `aresta inexistente`.

- `matriz`: imprime a *matriz de adjacência* do grafo. Note que a matriz deve ser gerada e impressa *apenas* quando este comando é chamado – o grafo em si deve ser armazenado em sua *lista de adjacência* durante toda a execução do programa. Imprima `-` para arestas ausentes/"infinito". Como exemplo, no grafo dado anteriormente, este comando deve imprimir:

```
- - - 4 - - 3 - - - -
- - - 1 - 2 - - - - -
- - - - 10 - - - 4 - -
4 1 - - - 5 - - - - -
- - 10 - - - - - 2 - -
- 2 - 5 - - - - - - 1
3 - - - - - - - - 12 -
- - - - - - - - - - -
- - 4 - 2 - - - - - -
- - - - - - 12 - - - 8
- - - - - 1 - - - 8 -
```

- `vizinhos A`: imprime em uma linha a lista de vizinhos do vértice A, separados por espaço. A lista pode ser dada em qualquer ordem. Caso o vértice A não tenha vizinhos, este comando deve imprimir `(nenhum)`. Como exemplo, no grafo dado anteriormente, o comando `vizinhos 6` deve imprimir
`11 2 4`
(em qualquer ordem), enquanto o comando `vizinhos 8` deve imprimir `(nenhum)`.

- `popular`: imprime `P (V vizinhos)`, onde P é vértice mais "popular" do grafo, e V é o seu grau. O vértice mais "popular" do grafo é aquele de maior grau. Se houver mais de um vértice com o maior grau, o mais "popular" é aquele de menor identificador. Como exemplo, no grafo dado anteriormente, o comando `popular` deve imprimir
`4 (3 vizinhos)` (note que o vértice 6 também tem 3 vizinhos, mas o vértice 4 é impresso porque seu identificador é menor).

- `dfs`: executa uma busca em profundidade (DFS) no grafo. Este comando deve imprimir uma linha para cada componente conexa do grafo, no formato `[A B C ...], peso total T`, onde `A B C ...` é a lista de vértices da componente (em qualquer ordem, separados por espaço) e `T` é o peso total de todas as arestas da componente. Ao final, este comando deve imprimir `C componente(s)`, sendo C o número de componentes conexas do grafo. Como exemplo, no grafo dado anteriormente, o comando `dfs` deve imprimir, por exemplo:

```
[1 4 6 11 10 7 2], peso total 36
[3 5 9], peso total 16
[8], peso total 0
3 componente(s)
```

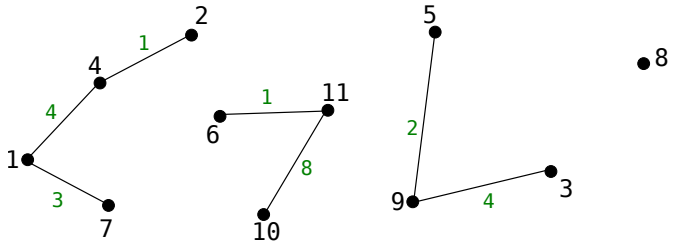
- `bfs A B`: executando uma busca em largura (BFS), determina o menor caminho (em *número de arestas*) entre o vértice A e o vértice B. Este comando deve imprimir em uma linha a sequência de vértices no caminho de A para B em ordem, separados por espaço, e, em outra linha, `M aresta(s), peso total T`, sendo M o número de arestas no caminho, e T o peso total das arestas no caminho. Se houver mais de um caminho mínimo (em número de arestas) entre A e B, imprima qualquer um (você não precisa minimizar a soma dos pesos das arestas). Se não houver um caminho entre A e B, imprima apenas `(sem caminho)`. Como exemplo, no grafo dado anteriormente, o comando `bfs 1 11` deve imprimir, por exemplo,

  ```
  1 4 6 11
  3 aresta(s), peso total 10
  ```

  enquanto o comando `bfs 9 2` deve imprimir `(sem caminho)` .

- `- A B`: remove a aresta $\{A, B\}$ do grafo. Este comando não imprime nenhuma saída. Você pode assumir que a aresta sempre existe antes da chamada deste comando. Como exemplo, no grafo dado anteriormente, os comandos dados abaixo (à esquerda) fazem do grafo o dado abaixo (à direita):

```
- 7 10
- 2 6
- 5 3
- 6 4
```



- `F`: libera toda a memória alocada e termina a execução do programa.

A última página deste documento contém um exemplo completo de entrada e saída com todos os comandos solicitados.

Observações

- Você pode assumir que o grafo terá no máximo 100 vértices;
- Note que é necessário usar uma fila para implementar a BFS. Além do arquivo `.c` principal, o trabalho deve conter (pelo menos) os seguintes quatro arquivos:
  - `FilaEstatica.h` e `FilaEstatica.c`: biblioteca que implementa uma fila usando como base um vetor (fila "estática");
  - `FilaDinamica.h` e `FilaDinamica.c`: biblioteca que implementa uma fila usando como base uma lista ligada (fila "dinâmica").

  Atenção: como a fila é um TAD, deve ser possível escolher entre usar uma fila estática ou uma fila dinâmica *apenas* alterando o `#include` do programa principal e recompilando-o, sem alterar nenhum outro trecho do código.

- Você pode escolher entre implementar a DFS recursivamente ou com uma pilha explicita. Caso opte pela pilha explicita, você também deve escrever e incluir os arquivos `PilhaEstatica.{h,c}` e `PilhaDinamica.{h,c}`, de maneira análoga à fila.

Orientações

- O trabalho pode ser feito por equipes de *até* 2 (dois) estudantes;
- Submeta, via *Moodle*, um pacote (zip/tar.gz) contendo todo o código fonte do trabalho, além de um arquivo de texto (txt) onde conste:
  - O nome de todos os integrantes da equipe;
  - Toda informação que a equipe julgar relevante para a correção.
- Submeta um pacote **zip** ou **tar.gz**. Pacotes que não puderem ser abertos pelo professor não serão corrigidos;
- Nomeie as variáveis e comente o código adequadamente para facilitar a correção;
- O programa será testado com outras entradas além das fornecidas como exemplo. Seu programa deve funcionar para qualquer entrada que esteja dentro da especificação;
- **Atenção**: a correção será parcialmente automatizada e, por isso, é necessário que tanto a entrada quanto a saída estejam no formato especificado. *Siga* ***fielmente*** *o formato de entrada e saída dado nos exemplos*, sob pena de grande redução da nota;
- O trabalho deve ser entregue até **12 de Maio (quarta-feira), 12:00 (meio-dia)**, via *Moodle*. É suficiente que o trabalho seja submetido por apenas um estudante da equipe;
- Trabalhos copiados ou plagiados receberão **todos** a nota 0 (**ZERO**).

Exemplo de entrada	**Saída esperada**
11 + 1 4 4 + 3 5 10 + 11 6 1 + 7 10 12 + 6 2 2 + 9 5 2 + 4 6 5 + 3 9 4 + 10 11 8 + 1 7 3 + 4 2 1 w 5 3 w 6 1 matriz vizinhos 6 vizinhos 8 popular dfs bfs 1 11 bfs 9 2 - 7 10 - 2 6 - 5 3 - 6 4 dfs popular bfs 1 11 bfs 2 8 + 7 6 1 + 10 5 40 + 3 8 12 dfs bfs 2 8 F	10 aresta inexistente - - - 4 - - 3 - - - - - - - 1 - 2 - - - - - - - - - 10 - - - 4 - - 4 1 - - - 5 - - - - - - - 10 - - - - - 2 - - - 2 - 5 - - - - - - 1 3 - - - - - - - - 12 - - - - - - - - - - - - - - 4 - 2 - - - - - - - - - - - - 12 - - - 8 - - - - - 1 - - - 8 - 11 2 4 (nenhum) 4 (3 vizinhos) [1 4 6 11 10 7 2], peso total 36 [3 5 9], peso total 16 [8], peso total 0 3 componente(s) 1 4 6 11 3 aresta(s), peso total 10 (sem caminho) [1 4 2 7], peso total 8 [3 9 5], peso total 6 [6 11 10], peso total 9 [8], peso total 0 4 componente(s) 1 (2 vizinhos) (sem caminho) (sem caminho) [1 4 2 7 6 11 10 5 9 3 8], peso total 76 1 componente(s) 2 4 1 7 6 11 10 5 9 3 8 10 aresta(s), peso total 76