Trabalho Prático 3: Graphs!

Entrega: 11/12/2017

Introdução

Um Grafo é uma estrutura de dados muito importante e que possui diversas aplicações. Diversos problemas são resolvidos com grafos e com os algoritmos para grafos. Nesse contexto, é muito importante que você saiba implementar um grafo e alguns algoritmos básicos para ele. Neste trabalho, você vai desenvolver um tipo abstrato de dados para um grafo. O seu tipo abstrato de dados realizará diversas ações a partir dos pedidos que um usuário realiza no terminal. O usuário pode realizar os seguintes pedidos/questões:

1. Crie um grafo direcionado/não-direcionado com N vértices
2. Adicione uma aresta: (u,v)
3. Adicione N arestas: (u_1,v_1),(u_2,v_2),...,(u_N,v_N)
4. Remova u
5. Remova (u,v)
6. Imprima a matriz de adjacência
7. Imprima as listas de adjacência
8. Existe o vértice u?
9. Existe a aresta (u,v)?
10. Existe um caminho de u a v?
11. Mostre um possível caminho de u a v caso exista

Algumas observações sobre os itens seguem abaixo. No item 1, o usuário pede para criar um grafo direcionado ou não. O grafo criado tem N vértices numerados de 0 a N-1, isto é, o índice do primeiro vértice criado é 0, do segundo é 1 e, assim por diante. O item 2 pede para adicionar uma aresta onde u é o vértice de origem e v é o vértice de destino. Se o grafo for não-direcionado, a aresta (v,u) também será adicionada. No item 3, o pedido é para adicionar N arestas. Para o

item 4, juntamente com o vértice u devem ser removidas todas as suas ligações. Para o item 5, se o grafo for não-direcionado, será removida também a aresta (v,u). Os itens 8, 9 e 10 devem ser respondidos com sim ou não.

Todas essas operações devem ser definidas e implementadas no seu tipo abstrato de dados (TAD). Crie um TAD que seja organizado e modularizado com funções bem definidas. O seu TAD deve representar internamente o grafo usando ambas as representações: matriz de adjacência e lista de adjacência. As duas representações devem estar atualizadas e consistentes após cada operação realizada.

Cada pedido/questão do usuário será digitada em uma linha. Então quando o usuário pressionar o enter, a operação deve ser executada e seu resultado impresso na tela.

O objetivo deste trabalho prático também é exercitar a leitura e manipulação de strings.

Existe uma biblioteca chamada SDS para manipulação de strings em C: https://github.com/antirez/sds. Para usar essa biblioteca basta baixar os arquivos .h e .c do site acima, compilar juntamente com seu código o arquivo sds.c e incluir, em um dos seus arquivos, os seguintes arquivos de cabeçalho: sdsalloc.h e sds.h. Essa biblioteca possui diversas funções úteis entre as quais destaca-se:

1. sdsnew: cria uma nova string;
2. sdssplitlen: faz o split da string usando um separador;
3. sdstrim: retira espaços em branco da string;
4. sdsrange: cria uma nova string com a substring do intervalo definido;
5. sdslen: retorna o comprimento de uma string.

Para quem fizer em C++, tem as funções substr e find da biblioteca string ($\#include < string >$), entre outras, que são úteis também.

Código

O código pode ser feito em C ou C++. Para quem já tiver feito POO e quiser, pode fazer o trabalho utilizando orientação a objetos em C++.

Entrada e Saída

Exemplo de entrada e saída:
Crie um grafo não-direcionado com 3 vértices
Adicione uma aresta (0,2)
Adicione uma aresta (1,2)
Existe o vértice 2?
sim

Existe a aresta (2,1)?
sim
Existe a aresta (0,1)?
não
Existe um caminho de 0 a 1?
sim
Mostre um possível caminho de 0 a 1 caso exista
0 - 2 - 1
Imprima a matriz de adjacência
0 0 1
0 0 1
1 1 0
Remova 2
Existe a aresta (0,2)?
não

Crie um grafo direcionado com 4 vértices
Adicione 3 arestas: (0,2),(1,2),(2,1),(1,3)
Imprima as listas de adjacência
0 -> 2 -> NULL
1 -> 2 -> 3 -> NULL
2 -> 1 -> NULL
3 -> NULL

O que entregar

Você deve submeter uma documentação de até 12 páginas contendo uma descrição de como cada uma das operações foi implementada. Além da documentação, você deve submeter um arquivo compactado contendo todos os arquivos de código (*.c* e *.h*) que foram implementados. Além dos arquivos de código, esse arquivo compactado deve incluir um *makefile*.

Avaliação

Seu trabalho será avaliado quanto a documentação escrita e à implementação. Eis uma lista **não exaustiva** de critérios de avaliação utilizados.

Documentação

Introdução Inclua uma breve explicação do problema que está sendo resolvido no seu trabalho e um resumo da sua solução.

Solução do Problema Você deve descrever a solução do problema de maneira clara e precisa. Para tal, artificios como pseudo-códigos, exemplos ou figuras podem ser úteis. Note que documentar uma solução não é o mesmo que documentar seu código. **Não** é preciso incluir trechos de código em sua documentação nem mostrar detalhes de sua implementação, exceto quando os mesmos influenciem o seu algoritmo principal, o que se torna interessante.

Avaliação Experimental Sua documentação deve incluir os resultados de experimentos que avaliem o tempo de execução de seu código em função de características da entrada. Cabe a você gerar entradas para esses experimentos. Por exemplo: se esse trabalho fosse sobre ordenação, seria interessante mostrar como o tempo de execução de cada algoritmo varia quando o número de items a serem ordenados aumenta. Para tal, um gráfico mostrando o tempo de execução em função do tamanho da entrada pode ser interessante. Você também deve interpretar os resultados obtidos. Comente sobre cada gráfico ou tabela que você apresentar mostrando o que é possível concluir a partir dele.

Valor

- O trabalho vale 20 pontos, sendo 17 pontos para o código e 3 pontos para a documentação. Ambos devem ser entregues. O aluno que entregar somente um dos dois terá sua nota zerada.
- O trabalho que não compilar receberá nota 0.
- O trabalho poderá ser feito em dupla.

Considerações Finais

- Essa especificação não é isenta de erros e ambiguidades. Portanto, se tiver problemas para entender o que está escrito aqui, me pergunte e pergunte ao monitor.