Questão **1**

Correto

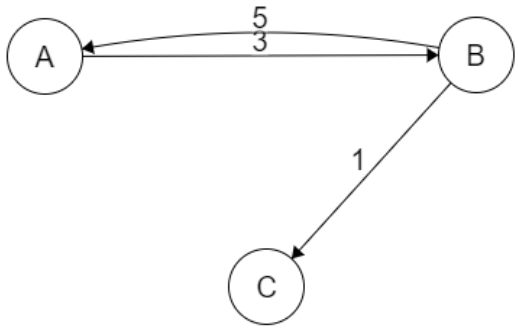
Atingiu 1,0 de 1,0

Desenhe um grafo direcionado valorado com três nós A, B e C, de forma que as seguintes propriedades sejam satisfeitas:

- o grafo possui apenas 3 arestas.
- O caminho mais curto de A para B tem comprimento 3.
- O caminho mais curto de A para C tem comprimento 4.
- O caminho mais curto de B para C tem comprimento 1.
- O caminho mais curto de B para A tem comprimento 5.
- Não há caminho de C para A nem para B.

**Answer:** (penalty regime: 10, 20, ... %)

Help Clear

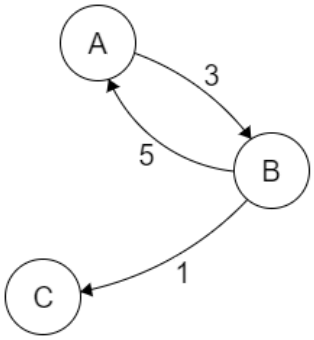



| | Test | Expected | Got | |
|---|---|---|---|---|
| ✔ | `# Check the set of nodes`<br>`print(sorted(graph.keys()))` | `['A', 'B', 'C']` | `['A', 'B', 'C']` | ✔ |
| ✔ | `# Check all edges have integer weights`<br>`for u, elist in graph.items():`<br>`  for v, label in elist:`<br>`  try:`<br>`  int(label)`<br>`  except ValueError:`<br>`  print("A aresta de {} para {} deveria ter um valor inteiro".format(u, v))` | | | ✔ |
| ✔ | `# Print connectivity (computed by`<br>`# hidden all-pairs shortest path code).`<br>`for src in 'ABC':`<br>`  for dest in 'ABC':`<br>`  if src != dest:`<br>`  d = distances[src, dest]`<br>`  print(src, dest, '-' if d == INF else d)` | `A B 3`<br>`A C 4`<br>`B A 5`<br>`B C 1`<br>`C A -`<br>`C B -` | `A B 3`<br>`A C 4`<br>`B A 5`<br>`B C 1`<br>`C A -`<br>`C B -` | ✔ |
| ✔ | `# Check there are just 3 edges`<br>`print(sum([len(edgelist) for edgelist in graph.values()]))` | `3` | `3` | ✔ |

Passou em todos os teste! ✔

## Question author's solution (Python3):

Help Clear




Correto

Notas para este envio: 1,0/1,0.

Questão **2**

Correto

Atingiu 1,0 de 1,0

Crie uma classe que servirá como base para todas as outras submissões relativas ao curso de Teoria dos Grafos.

Observe as seguintes convenções:

- o nome da classe é Graph

- o método **construtor** da classe receberá como parâmetro 1 valor inteiro N. N é a quantidade de vértices presentes nesse grafo. Os vértices serão identificados pelos valores [0... N-1]

- um método polimórfico chamado addEdge será membro dessa classe, da seguinte maneira:

addEdge(u,v) - adiciona uma aresta direcionada que sai do vértice u para o vértice v

addEdge(u,v,w) - adiciona uma aresta direcionada que sai do vértice u para o vértice v com peso w

- um método chamado getInDegree(V) retorna o grau de entrada do vértice V

- um método chamado getOutDegree(V) retorna o grau de saída do vértice V

- um método chamada getDegree(V) retorna o grau do vértice V.

A estrutura de dados utilizada para representar seu grafo pode ser à sua escolha.

Você pode criar métodos auxiliares na sua classe.

#Teste 1

g = Graph(1)

g.addEdge(0,0)

print(g.getInDegree(0), g.getOutDegree(0), g.getDegree(0))

Saída:

1 1 2

#Teste 2

g = Graph(2)

g.addEdge(0,1)

print(g.getInDegree(0), g.getOutDegree(1), g.getInDegree(1),g.getOutDegree(0))

Saída:

0 0 1 1

**Answer:** (penalty regime: 10, 20, ... %)

```
class Graph:
    def __init__(self, n):
        self.n = n
        self.adj = [[] for i in range(n)]

    """
    Adiciona uma aresta direcionada que sai do vértice u para o vértice v com peso w
    u: inteiro que representa o vértice de origem
    v: inteiro que representa o vértice de destino
    w: inteiro que representa o peso da aresta (opcional)
    """
    def addEdge(self, u, v, w=1):
        self.adj[u].append((v,w))

    """
    Retorna o grau de entrada do vértice V
    V: inteiro que representa o vértice
    """
    def getInDegree(self, v):
        sum = 0
        for i in self.adj:
            for j in i:
                if j[0] == v:
                    sum += 1
        return sum

    """
    Retorna o grau de saída do vértice V
    V: inteiro que representa o vértice
```

```
30      """
31      def getOutDegree(self, v):
32          return len(self.adj[v])
33
34      """
35      Retorna o grau do vértice V
36      V: inteiro que representa o vértice
37      """
38      def getDegree(self, v):
39          return self.getInDegree(v) + self.getOutDegree(v)
40
```

| | Test | Expected | Got | |
|---|---|---|---|---|
| ✔ | `g = Graph(1)`<br>`g.addEdge(0,0)`<br>`print(g.getInDegree(0), g.getOutDegree(0), g.getDegree(0))` | `1 1 2` | `1 1 2` | ✔ |
| ✔ | `g = Graph(2)`<br>`g.addEdge(0,1)`<br>`print(g.getInDegree(0), g.getOutDegree(1), g.getInDegree(1),g.getOutDegree(0))` | `0 0 1 1` | `0 0 1 1` | ✔ |

Passou em todos os teste! ✔

Correto

Notas para este envio: 1,0/1,0.

Questão **3**

Completo

Vale 1,0 ponto(s).

Baseado em sua experiência própria, sugira aplicações práticas (do mundo 'real') para Fecho Transitivo Direto e Fecho Transitivo Inverso de um grafo.

Formate a sua sugestão da seguinte maneira:

- Descreva a situação

- Descreva a maneira de representar a situação como um grafo (quem assume o papel de vértices, quem assume o papel de arestas)

- A aplicação.

Fecho Transitivo Direto:

Em relação ao fecho transitivo direto, que se relaciona com o conjunto de vértices que pode ser atingido a partir de um vértice V, podemos dizer que essa forma de analise poderia ser feita para analisar os descendentes deixados por um determinada pessoa em um arvore genealógica. Em outra palavras, poderíamos verificar se determinada pessoa deixou filhos e se sua linhagem foi continuada ate determinado período de tempo. Em um grafo, poderíamos tratar cada pessoa como sendo um vértice, sendo que as arestas nos mostrariam a conexão direta de determinada pessoa com seus descendentes, exatamente como funciona uma arvore genealógica.

Fecho Transitivo Inverso:

De forma antagônica com o anterior, o fecho transitivo inverso analisa o conjunto de vértices a partir dos quais é possível atingir um vértice v. Com isso, relacionado com a resposta do item anterior, podemos dizer que podemos usar esse tipo de analise para descobrir quem são os antepassados de terminadas pessoa em uma arvore genealógica. Em um grafo, poderíamos tratar cada pessoa como sendo um vértice, sendo que as arestas seriam conexão direta de determinada pessoa com outra, sendo possível, assim, analisar quem são os antepassados de alguém.

◄ Aula 3 - Conceitos e Representação em Memória

Seguir para...

Video - Busca em Largura ►