REPRESENTAÇÃO TEORICA DOS GRAFOS

Os grafos podem ser definidos matematicamente por meio de um par em que o primeiro elemento contém o conjunto de vértices; o segundo, o conjunto de arestas representadas por um par não ordenado (no caso de um grafo não dirigido) ou ordenado (no caso de um dígrafo). Dessa forma, formalmente tem-se:

$$G = \big(V(G), E(G)\big)$$

No entanto, tal modelo de abstração humana deve ser representado consoante a uma dada estrutura de dados em um contexto computacional. Dessa forma, pode-se recorrer a listas e matrizes.

Dentre estas formas, a mais simples de representar computacionalmente um grafo, bem como a mais próxima da definição formal, seria por meio de uma lista em que cada item conteria os vértices constituintes da aresta. Caso o grafo seja não orientado, a ordem em que os vértices são inseridos em cada item é irrelevante, no entanto não se deve ter redundâncias do tipo (a,b),(b,a). Todavia, se se tratar de um dígrafo a ordem dos vértices refletirá se são de partida ou de chegada. Em Python, para G = ({a,b,c},{(a,b), (b,c)}):

Conjunto = [[a,b],[c,b]]

Outra representação computacional de grafos por meio de listas encontra-se na lista de adjacência. Cada item dessa lista conterá os vértices adjacentes de um dado vértice. O índice dos itens identifica sobre qual vértice se trata, haja vista estes passarem por um processo de ordenação, por exemplo pode-se colocá-los em uma lista auxiliar. Assim, o primeiro item da lista de adjacência refere-se ao vértice referente ao primeiro item da lista auxiliar. Caso se trate de um dígrafo, o item conterá apenas os vértices que computam o grau de saída do vértice. Tal lista é utilizada, por exemplo nos algoritmos de busca em largura e de profundidade. Em Python, para G = ({a,b,c,d}, { (a,b), (b,c),(b,d) }) tem-se:

Lista_auxiliar = [a,b,c,d]

Lista_adjacencia = [[b],[c,d],[b],[b]]

Por outro lado, pode-se representar grafos por meio de matrizes de inteiros, em geral, por meio de 0's e 1's no caso de grafos não dirigidos ou de 0's, 1's e -1's no caso de dígrafos. Nesse contexto, há dois tipos de matrizes: de incidência e de adjacência. No primeiro caso, o grafo é representado por uma matriz cujas linhas representam os vértices e cujas colunas representam as arestas. Caso a aresta de uma dada coluna seja incidente aos dois vértices, o número 1 é inserido no cruzamento das linhas com a coluna. O restante da coluna é preenchido por 0's. Caso o grafo se trate de um dígrafo, o número -1 representa o vértice de chegada, enquanto que o 1; o vértice de partida. Para G = ({a,b,c}, { (a,b), (b,c) }):

Ao se adotar uma representação matemática, tem-se:

$$\begin{matrix} 1 & 0 \\ 1 & 1 \\ 0 & 1 \end{matrix}$$

Em Python:

Matriz_incidencia = [[1 0],[1 1], [0 1]]

No caso das matrizes de adjacência, a representação matricial ficaria a cargo de uma matriz quadrada em que ordem n é definida pelo número de vértices do grafo. Cada linha representaria os vértices, bem como as colunas na mesma ordem. Dessa forma, caso o vértice de uma dada linha seja adjacente ao vértice de uma dada coluna, o inteiro 1 é inserido no cruzamento de linhas com coluna. O restante da linha é então preenchido com 0's. Caso o grafo se trate de um dígrafo, o número -1 representa o vértice de chegada, enquanto que o 1; o vértice de partida. Para G = ({a,b,c}, { (a,b), (b,c) }):

Ao se adotar uma representação matemática, tem-se:

$$\begin{matrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{matrix}$$

Em Python:

```
Matriz_incidencia = [[0 1 0], [1 0 1], [0 1 0]]
```

ATENÇÃO: PARA MELHOR VISUALIZAÇÃO, ABRA ESTE ARQUIVO PELO GITHUB

Trabalhos referente as aulas de teoria os grafos incluindo atividades avaliativas referente a produção de algoritmos!

Segue abaixo o relatório de cada parte do algoritmo e como executar:

Todos os códigos produzidos foram feitos em linguagem de programação python, abaixo contém uma descrição de cada nível pasta:

OBSERVAÇÃO: todos os algoritmos foram feitos em python 3.7 e é preciso instalar a biblioteta numpy, para instalar, execute no terminal: ***pip install numpy***

Feito isso, vamos aos codigos:

```
|-----Avaliacao1
|   |
|   |-----interface
|   |     *Nessa atividade foi feita uma interface gráfica para melhor entendimento da execução
|   |      do código. A interface foi produzida na biblioteca nativa do python 3.7, Tkinter.
|   |
|   |-----representações
|   |   |     *Aqui contém todas as representações e manipulações solicitadas pela professora Patrícia, isso inclui as
|   |   |     representações de lista de adjacência, matriz de incidência e matriz de adjacência e em cada uma delas
|   |   |     vai ser gerada uma representação por conjuntos. As representações estão separadas por classes em
|   |   |     diferentes arquivos .py.
|   |   |
|   |   |--Grafo.py
|   |   |      *Classe pai abstrata que contém as propriedades básicas de cada representação de grafo.
|   |   |
|   |   |--listaAdjacecia.py
|   |   |     *Representação de grafo por lista de adjacência e seus métodos de manipulação!
|   |   |      Definição: Dado um grafo G(V,E) Cada lista A(v) é denominada lista de adjacência de v, e contém todos os
|   |   |      os vértices que são adjacentes a v.
|   |   |--matrizAdjacencia.py
|   |   |    *Representação de grafo por matriz de adjacência e seus métodos de manipulação
|   |   |      Definição: A matriz de adjacência é uma matriz n x n, onde n é dado por n vertices, onde o valor de cada
|   |   |      elemento E(jk) da matriz é determinado da seguinte maneira:
|   |   |            E(jk) = 1, se os vértices v(j) e v(k) são ligados por uma aresta, senão, E(jk) = 0.
|   |   |
|   |   |--matrizIncidencia.py
|   |   |    *Representação de grafo por matriz de incidência e seus métodos de manipulação
|   |   |       Definição: Seja G um grafo de n vértices v1, v2,…,vn, e m arestas a1, a2, …, am, e nenhum laço.
|   |   |       É uma matriz n x m, onde o valor de cada elemento da matriz é determinado da seguinte maneira:
|   |   |            E(jk) = 1, se a aresta a(k) é incidente ao vértice v(j), senão, = 0!
|   |   |       Sendo J a linha e K a coluna do elemento E.
|   |   |       Se G for um grafo dirigido, então:
|   |   |            E(jk) = 1, se a aresta a(k) sai do vértice v(j)
|   |   |            E(jk) = -1, se a aresta a(k) chega no vértice v(j), senão, = 0
|   |
|   |--main.py
|   |     *Arquivo principal de execução dos métodos de representações, basta executar no terminal esse arquivo!
|
|-----Busca-Largura
|   |
```

```
|  |--busca.py
|  |     *Arquivo principal de execução do método, nele está declarado estaticamente o grafo representado na
|  |     imagem Grafo1.png na mesma pasta. Para testar outro grafo basta declarar o grafo desejado por meio da
|  |     lista de adjacência.
|  |
|-----Busca-Largura
|  |
|  |--Main.py
|  |     *Arquivo principal de execução do método, nele está declarado estaticamente o grafo representado
|  |     por lista de adjacência. Para testar outro grafo basta declarar o grafo desejado por meio da
|  |     lista de adjacência.
|  |
|  |--Grafo1.png
|  |     *Representação do grafo utilizado para teste no algoritmo de busca!
|
|-----Fleury
|  |
|  |--Fleury.py
|  |     *Arquivo principal de executarão do método.
|  |
|  |--GrafosEulerianosFleuryEx1e2.pdf
|  |     *Este pdf contém os exemplos utilizados no algoritmo.
|  |
|  |--README.md
|  |     *Este é um arquivo importante, para mais detalhes do algoritmo de Fleury, abra e leia este arquivo.
|
|-----Hierholzer
|  |
|  |--main.py
|  |     *Arquivo principal, da atividade de grafos eulerianos e trilhas eulerianas.
|  |
|
|---Kruskal
|  |
|  |--checkCicle.py
|  |     *Este arquivo irá armazenar a função que verifica a possibilidade de existência de ciclos
|  |     dentro da minha arvore atual.
|  |
|  |--grafo.py
|  |     *Este arquivo contém a função de retorna a lista de arestas, ordenada por peso, de um grafo,
|  |     e esse grafo deve ser passado por representação de lista de adjacência.
|  |
|  |--Kruskal.py
|  |     *Este é o arquivo principal para execução, nele contém o algoritmo de kruskal. Também contém
|  |     4 grafos declarados estaticamente com o fim de verificar a veracidade do algoritmo. Cada grafo
|  |     é tem sua imagem correspondente na mesma pasta, por exemplo, a variável grafo1 no algoritmo
|  |     de kruskal tem sua imagem corresponde que é grafo1.png.
|  |
|  |--grafo1.png
|  |     *Este é o grafo correspondente a variável grafo1 no algoritmo de kruskal.
|  |
|  |--grafo2.png
```

| | *Este é o grafo correspondente a variável grafo2 no algoritmo de kruskal.
| |
| |--grafo3.png
| | *Este é o grafo correspondente a variável grafo3 no algoritmo de kruskal.
| |
| |--grafo4.png
| | *Este é o grafo correspondente a variável grafo4 no algoritmo de kruskal.
|
|-----Prim
| |
| |--checkCicle.py
| | *Este arquivo irá armazenar a função que verifica a possibilidade de existência
| | de ciclos dentro da minha arvore
| |
| |--prim.py
| | *Este é o arquivo principal para execução, nele contém o algoritmo de prim. Também contém
| | 4 grafos declarados estaticamente com o fim de verificar a veracidade do algoritmo. Cada grafo
| | tem sua imagem correspondente na mesma pasta, por exemplo, a variável grafo1 no algoritmo
| | de Prim tem sua imagem corresponde que é grafo1.png.
| |
| |--grafo1.png
| | *Este é o grafo correspondente a variavel grafo1 no algoritmo de Prim
| |
| |--grafo2.png
| | *Este é o grafo correspondente a variavel grafo2 no algoritmo de Prim
| |
| |--grafo3.png
| | *Este é o grafo correspondente a variavel grafo3 no algoritmo de Prim
| |
| |--grafo4.png
| | *Este é o grafo correspondente a variavel grafo4 no algoritmo de Prim
|
|-----Relatorio Group
| | *Este relatório criado pelo grupo
|
|

RELATORIO PESSOAL DE CADA MEMBRO DO GRUPO

1. Manoel Gomes Borges

Este trabalho com certeza foi um dos mais trabalhosos já feitos, nossa equipe iniciou os trabalhos bem cedo e ainda sim pareceu que não ia dar tempo, no fim deu tudo certo e todos os membros conseguiram entregar suas partes a tempo. Além disso, a parte em que eu tive mais dificuldade para achar uma solução lógica, foi a implementação das representações de grafos e suas manipulações!

2. Carlos Mateus Sena de Oliveira

Mesmo dividindo as tarefas para os membros do grupo, ainda sim obtive dificuldade e entender algumas partes do conteúdo e da implementação, porém, por ser uma equipe nos sentamos e começamos a debater os conteúdos da disciplina e como cada regra de cada algoritmo ia ser aplicada na linguagem, assim, conseguimos obter transparência nas implementações. Ainda assim, existem partes dos algoritmos que trabalham com recursividade e estou tentando entender isso, mas, não implica no entendimento do conteúdo passado em sala.

3. Lucas Ferro Zampar

Embora tenha compreendido os processos dos algoritmos apresentados em sala e eles tenham sido bem elucidados pela explicação da professora, minha maior dificuldade consistiu na implementação. Isso se deve pois durante esta etapa, cada linha de raciocínio humano em relação ao algoritmo deve ser traduzida em código de forma consistente com a lógica computacional. Por exemplo, embora seja intuitivo pensar em um grafo, este deve ser a representado por meio de uma estrutura de dados e as etapas de busca sobre tal estrutura devem ser coerentes com lógica de programação, além das sequências do próprio algoritmo. Isso nos leva a não somente compreender o algoritmo, mas a pensar sobre este, a raciocinar de acordo com as perspectivas deste e com a de uma máquina. No entanto, o pensamento humano muitas vezes não é tão mecânico, o que leva a uma eventual dificuldade na implementação. O projeto envolveu muitos desafios, mas houve uma satisfação interna em cada bug resolvido, em cada momento que o terminal devolvia o resultado esperado depois de dezenas de fracassos e que pouco a pouco nossa lógica melhorava e nossos resultados ficavam mais próximos do ideal. Por fim, o esforço valeu a pena e foi gratificante chegar até o final.