Lista de exercícios

1. Escreva um programa que aceita como **entrada** algum dos itens a seguir, e dá como **saída** os outros dois:
    - Uma lista de arestas de um grafo dadas como pares de inteiros positivos;
    - A matriz de adjacência;
    - A matriz de incidência.

   NOTA::Utilizaremos recursos *online* para especificação de grafos com a finalidade de gerar grafos de forma simples. São diversas as interfaces gráficas disponíveis, sugiro duas: GraphOnline e GraphRel. Com esses recursos podemos especificar grafos através de interfaces gráficas e exportar os dados, que servirão de entrada para os programas que vocês fizerem.

   NOTA:: Veja nesta referência algumas operações tipicamente disponíveis em bibliotecas para trabalhar com grafos.

2. Escreva um programa que determina se um grafo contém um ciclo euleriano.

   NOTA:: Discutimos duas estratégias para abordar esse problema. Lançamos mão do teorema que estabelece que se um grafo contém ciclo euleriano então o grafo é conexo e todos os vértices têm grau par. Separamos então o problema em duas partes. (a) Verificar o grau dos vértices; (b) Verificar se o grafo é conexo. Para a parte (b) comentamos sobre as matrizes potência e o fato de serem uma implementação ineficiente. Uma opção eficiente é o algoritmo *depth-first-search (DFS)*. Neste link temos uma implementação recursiva do (DFS). O programa pode ser facilmente adaptado para responder se um grafo é ou não é conexo.

```
# Using a Python dictionary to act as an adjacency list
graph = {
    'A' : ['B','C'],
    'B' : ['D', 'E'],
    'C' : ['F'],
    'D' : [],
    'E' : ['F'],
    'F' : []
}

visited = set() # Set to keep track of visited nodes.

def dfs(visited, graph, node):
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)

# Driver Code
dfs(visited, graph, 'A')
```

3. Escreva uma programa que gera aleatoriamente uma matriz de adjacência n × n. Seu programa tem que imprimir a matriz de adjacência, o número de arestas, o número de loops (laços) e o grau de cada vértice.
4. Escreva um programa que determina se um grafo é bipartido. Se o grafo for bipartido, o grafo deve listar os conjuntos disjuntos de vértices.

5. Escreva um programa que liste todos os caminhos simples entre dois vértices dados.
6. Implemente o algoritmo de Dijkstra como um programa. O programa deveencontrar o menor caminho e seu comprimento.

   NOTA: Veja uma visualização do algoritmo de Dijkstra sendo executado neste link. Neste outro link temos visualizações de diversos algoritmos em grafos em execução, vale a visita.

7. Implementar o algoritmo de Palmer para construir um ciclo Hamiltoniano.