Implementação de um grafo

Gustavo Zambonin
Grafos (UFSC-INE5413)

A estrutura utilizada para a implementação de um grafo assemelha-se a um mapa. Utiliza como chave um vértice e como valor um conjunto de outros vértices aos quais este se conecta. Na linguagem escolhida, Python, estas estruturas são chamadas de `dict` e `set`. A fonte utilizada para verificar a complexidade de recursos da linguagem pode ser acessada aqui.

- `add_vertex`
  Um mapa adiciona chaves em tempo constante. $\mathbf{O(1)}$
- `remove_vertex`
  A remoção de um vértice sem conexões acontece em $O(1)$. Caso existam arestas conectadas a ele, então é necessário iterar sobre todos os vértices existentes e verificar se o vértice está presente no conjunto de conexões. A iteração leva tempo linear, e a operação de presença tem caso médio $O(1)$. $\mathbf{O(n)}$
- `connect_two_vertices`
  Mapear valores para chaves leva tempo constante. $\mathbf{O(1)}$
- `disconnect_two_vertices`
  Remover valores de chaves leva tempo constante. $\mathbf{O(1)}$
- `graph_order`
  A operação de tamanho de lista leva tempo constante. $\mathbf{O(1)}$
- `get_vertices`
  A iteração sobre as chaves de um mapa leva tempo linear. $\mathbf{O(n)}$
- `get_random_vertex`
  A complexidade temporal para gerar um número aleatório no módulo `random` em Python consegue ser constante. O método chamado na implementação presente, `choice`, pode ser consultado aqui. $\mathbf{O(1)}$
- `get_vertex_predecessors`
  Checar se um vértice está presente em n conjuntos leva tempo linear. $\mathbf{O(n)}$
- `get_vertex_sucessors`
  Acessar o valor de uma chave leva tempo constante. $\mathbf{O(1)}$
- `get_adjacent_vertices`
  Caso o grafo seja direcionado, unir dois conjuntos leva tempo linear. Do contrário, acessar o valor de uma chave leva tempo constante. $\mathbf{O(n)}$ ou $\mathbf{O(1)}$
- `get_vertex_indegree`
  Tamanho do retorno de `get_vertex_precedessors`. $\mathbf{O(n)}$
- `get_vertex_outdegree`
  Tamanho do retorno de `get_vertex_sucessors`. $\mathbf{O(1)}$
- `get_vertex_degree`
  Tamanho do retorno de `get_adjacent_vertices`. $\mathbf{O(n)}$ ou $\mathbf{O(1)}$
- `graph_regularity`
  Pode ser necessário iterar sobre todos os vértices. $\mathbf{O(n)}$
- `graph_completeness`
  Novamente, pode-se mostrar necessário iterar sobre a lista inteira de vértices. $\mathbf{O(n)}$
- `transitive_closure`
  O número de testes será proporcional ao somatório dos graus de saída de cada vértice, ou seja, o número de arestas do grafo. $\mathbf{O(m)}$

gustavo.zambonin@grad.ufsc.br src/

- `graph_connectivity`
  Utiliza o retorno de `transitive_closure`. $\mathbf{O(m)}$

- `is_tree`
  O método interno `check_cycles` calcula os vértices adjacentes de todos os vértices do grafo, e o retorno chama `graph_connectivity`. $\mathbf{O(n^2)}$