Relatório - REO 7

Arthur Henrique Sousa Cruz

Universidade Federal de Lavras

1 Execução

Para executar o código utilize os seguinte comando:

python3 main.py <arquivo-de-entrada>

2 Organização de arquivos

Os arquivos deste projeto foram organizados da seguinte forma:

- **main.py**: arquivo principal;
- **gerador.py**: contém a implementação de um gerador de arquivos de entrada. O gerador não garante conexidade;
- **gerenciador_arquivo.py**: contém a implementação da leitura do arquivo de entrada;
- **vertice.py**: arquivo com a classe que representa um vértice;
- **aresta.py**: arquivo com a classe que representa uma aresta;
- **grafo.py**: arquivo com as funções de um grafo;
- **lista_adj.py**: arquivo que implementa a lista de adjacência;
- **entradas**: pasta com arquivos de exemplo de entrada;

3 Implementação

A implementação foi feita em *Python*3. Foi utilizada uma lista de adjacência para representar o grafo, visto que busca-se pelos vizinhos de um vértice durante a busca em largura (usada para encontrar os caminhos aumentantes). As arestas também foram armazenadas em estruturas secundárias, para facilitar a implementação e agilizar os acessos. Foi utilizado o algoritmo de Edmonds-Karp, que utiliza o algoritmo de busca em largura para encontrar os caminhos aumentantes do algoritmo de Ford-Fulkerson.

Como a entrada permitia mais de uma aresta para cada par de vértices, foi criada uma aresta artificial com a soma das capacidades das arestas do par. Na lista de adjacência, as arestas foram inseridas duas vezes, de forma a representar o grafo como não direcionado. Assim, foi possível utilizar o algoritmo de Edmonds-Karp para solução do fluxo máximo.

Neste algoritmo, uma busca em largura é repetida, para o vértice inicial, até que não haja mais caminhos aumentantes. Considerando um grafo com mais de $\mid V \mid /2$ vértices, temos que a complexidade para encontrar os caminhos aumentantes é de $O(\mid V \mid \times \mid E \mid)$, já que é realizada uma busca em largura para encontrá-lo. As buscas são executadas $O(\mid E \mid)$ vezes e por isso temos que a complexidade total do algoritmo é $O(\mid V \mid \times \mid E^2 \mid)$.

4 Análise do algoritmo

O algoritmo foi testado com instâncias geradas pelo programa *gerador.py*. Foram criados arquivos com 5, 50, 100 vértices e 10, 10000, 20000 arestas, respectivamente. Para a instância de 5 vértices foram gastas 29 comparações, para a de 50 , 44247, e para a de 100, 479993. Como os grafos gerados são esparsos (e, provavelmente, desconexos) o número de comparações reflete as características do algoritmo. A complexidade cresce, em maior grau, de acordo com o número de arestas e, por isso nas instâncias testadas o número de comparações se manteve baixo (comparado ao número que poderia alcançar em grafos completos com mesmo número de vértices).