Relatório - REO 7

Arthur Henrique Sousa Cruz

Universidade Federal de Lavras

1 Execução

Para compilar e executar o código utilize os seguinte comando:

python3 main.py <arquivo-de-entrada> <arquivo-saida>

2 Organização de arquivos

Os arquivos deste projeto foram organizados da seguinte forma:

- **main.py**: arquivo principal;
- **gerenciador_arquivo.py**: contém a implementação da leitura do arquivo de entrada;
- **grafo.py**: arquivo com as funções de um grafo;
- **bellman_ford.py**: arquivo que implementa o algoritmo de Bellman-Ford;
- **entradas**: pasta com arquivos de exemplo de entrada;
- **saidas**: pasta com arquivos de exemplo de saída (foram executados os arquivos.

3 Implementação

A implementação foi feita em *Python*3. Para o Bellman-Ford a utilização de uma lista de adjacência é uma opção melhor que a matriz de adjacência, visto que todas as arestas serão percorridas e que a lista ocupa menos memória. Porém, na implementação feita, não foi usada nem a lista nem a matriz de adjacência. As arestas foram armazenadas em uma lista de tuplas, cada uma contendo o vértice de origem, o vértice de destino e o peso. Os nomes dos vértices foram armazenados em uma lista auxiliar de forma a facilitar o controle de acesso aos vetores. Foi escolhida essa forma de representação pois o Bellman-Ford pode percorrer as arestas fora de ordem.

4 Análise do algoritmo

O algoritmo foi testado com os arquivos disponibilizados pelo professor. Para os grafos *grafociclonegativo* e *grafoteste10vertices_2* foram encontrados ciclos negativos. Para todos os outros foi possível calcular os caminhos mínimos. Para o

grafopequeno foram necessárias 288 relaxações, enquanto que para o *grafomaior* foram necessários 38610 relaxações.

Os resultados refletem a complexidade do algoritmo de Bellman-Ford de $O(n \times m)$. No pior dos casos o grafo seria completo e a complexidade seria de $O(n^3)$, mas os arquivos *grafomaior* e *grafopequeno* provavelmente continham grafos esparsos e por isso a a quantidade de iterações reais foi baixa.

A complexidade se deve ao fato de que todas as arestas são percorridas $\mid V \mid -1$ vezes para fazer a busca dos caminhos. O algoritmo de Dijkstra diminui essa complexidade ao usar uma lista de prioridades, completando o algoritmo em $O(\mid E \mid \times \mid V \mid \log \mid V \mid)$. O problema é que a abordagem gulosa torna possível *loops* infinitos quando se tem arestas negativas, algo identificável para o Bellman-Ford.