Teoria dos Grafos

Trabalho da Disciplina - Parte 3

Engenharia de Computação e Informação UFRJ

Carolina Santiago de Medeiros 122053305
Zuilho Rodrigues Castro Segundo 122064877

Link do Repositório: https://www.github.com/ZuilhoSe/TeoGrafo_3

1 Introdução

Neste relatório, apresentamos a terceira parte do trabalho de algoritmos em grafos, que consiste na implementação do algoritmo de Ford-Fulkerson para encontrar o fluxo máximo em uma rede. Diferentemente das partes anteriores, que foram feitas em C++, optamos por usar a linguagem Python nesta parte, pois encontramos muitas dificuldades e complicações com a implementação em C++. Uma das principais dificuldades foi a alocação de memória para criar o grafo residual, que é necessário para aplicar o algoritmo de Ford-Fulkerson. Além disso, não conseguimos definir uma estrutura de classes para representar as arestas e os grafos que fosse coerente com o que já tínhamos construído nas partes anteriores. Por isso, decidimos usar Python, que é uma linguagem mais flexível e dinâmica, e que nos permitiu implementar o algoritmo de forma mais simples e eficiente.

2 Análise do Código

2.1 Representação dos Grafos

Em vez de usar matrizes de adjacência, vamos usar listas de adjacência, que são mais eficientes em termos de espaço e tempo. As listas de adjacência são estruturas de dados que armazenam os vértices adjacentes a cada vértice do grafo. Para implementar as listas de adjacência, usamos um dicionário que mapeia cada vértice a uma lista de objetos da classe Edge, que contém as informações sobre a aresta que liga os dois vértices.

A classe Edge foi utilizada para implementar fluxos nos grafos, representando uma aresta no grafo, com atributos para vértices, capacidade, fluxo, e um indicador para aresta reversa. Ela também possui métodos para atualizar o fluxo (`update`) e verificar se a capacidade residual é maior que um valor delta (`elegible`).

2.2 Busca em Grafos

A biblioteca implementa apenas a Busca em Largura (BFS). Ela é usada para percorrer o grafo e explorar seus vértices e arestas através de sua implementação com filas, e serve como ferramenta auxiliar indispensável na resolução do problema de fluxo máximo.

2.3 Algoritmo de Ford-Fulkerson

O algoritmo de Ford-Fulkerson é um método para encontrar o fluxo máximo em uma rede de fluxo. O algoritmo consiste em repetir os seguintes passos até que não haja mais caminhos aumentantes na rede:

1. Encontrar um caminho aumentante, ou seja, um caminho da origem ao destino que tenha capacidade residual positiva em todas as arestas.

2. Aumentar o fluxo ao longo desse caminho, reduzindo a capacidade residual das arestas do caminho e aumentando a capacidade residual das arestas reversas.

O código implementa duas variantes do algoritmo de Ford-Fulkerson: uma sem usar o critério de delta e outra usando o critério de delta. O critério de delta consiste em escolher um valor inicial de delta igual à metade da soma das capacidades das arestas que saem da origem, e ir dividindo esse valor por dois até que seja igual a um. Em cada iteração do algoritmo, só são considerados os caminhos aumentantes que tenham capacidade residual maior ou igual a delta. Isso acelera a convergência do algoritmo, pois evita escolher caminhos com capacidades residuais muito pequenas.

O código também implementa algumas funções auxiliares relevantes, como por exemplo:

- `get_bottleneck`: determina o gargalo em um caminho de fluxo, ou seja, o quanto de fluxo adicional pode ser enviado ao longo do caminho sem violar as restrições de capacidade. A função recebe um caminho como argumento e retorna a aresta de menor capacidade residual nesse caminho, iterando sobre todas as arestas do caminho e comparando suas capacidades residuais.
- `get_path`: retorna um caminho aumentante da origem ao destino, usando uma busca em largura ou em profundidade. Se o critério de delta for usado, só considera as arestas com capacidade residual maior ou igual a delta.
- `augment`: aumenta o fluxo ao longo de um caminho aumentante, atualizando as capacidades residuais das arestas do caminho e das arestas reversas.
- `get_flow`: retorna o valor do fluxo máximo na rede, que é igual à soma dos fluxos das arestas que saem da origem.
- `flow_allocation`: escreve em um arquivo os dados das arestas da rede, incluindo os vértices e os fluxos. Retorna o tempo gasto para escrever o arquivo.

3 Estudos de Caso

Grafo	***Tempo de Execução Médio [s]***	***Fluxo Máximo***
1	0.0660	1058
2	0.1711	11189
3	0.3288	2964
4	8.2726	13486
5	41.2808	26360
6	40.9873	26812

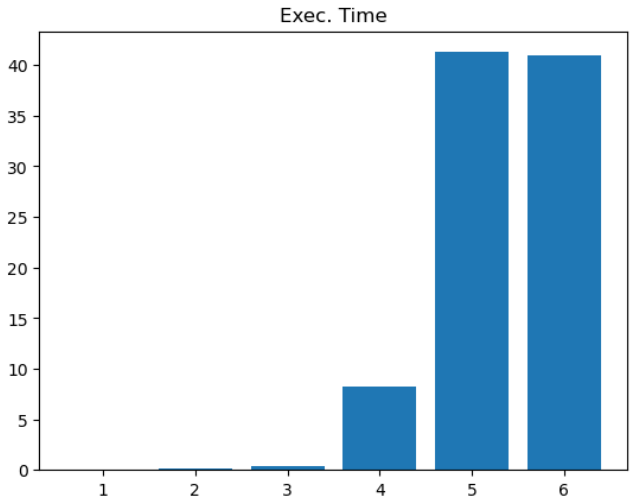

Figura 1: Execution Time

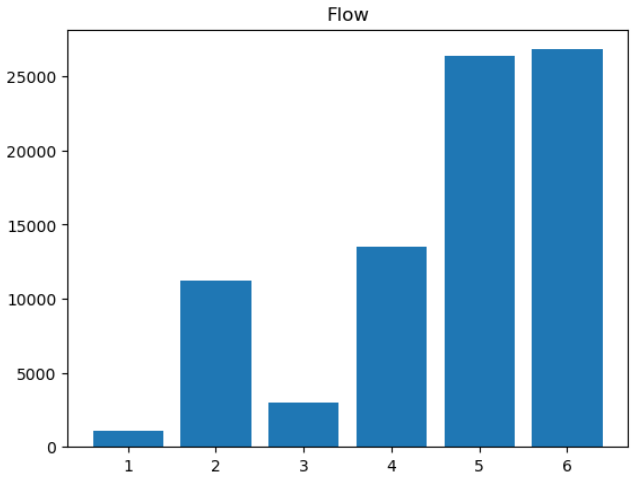

Figura 2: Flow