Relatório de Implementação Teoria dos Grafos e Computação

André Santos Alves, João Pedro Rosa de Moura, Giovanna Naves Ribeiro

[1]Ciência da Computação
Pontifícia Universidade Católica de Minas Gerais (PUC-MG)
Belo Horizonte – Minas Gerais – Brasil

Resumo. *Este relatório documenta as soluções implementadas para os trabalhos práticos da disciplina de Teoria dos Grafos e Computação no semestre 2024/2, cujo principal enfoque foi na implementação de três métodos para identificação de blocos em um grafo não direcionado e conexo. O documento detalhará as implementações, os experimentos e resultados obtidos.*

1. Introdução

Um grafo 2-conexo, também conhecido como biconexo, é crucial na computação por sua tolerância a falhas. São grafos onde cada par de vértices está conectado por dois caminhos disjuntos. Os componentes biconexos são blocos máximos desse tipo de grafo. Três abordagens serão exploradas para identificar esses blocos: verificação de caminhos disjuntos entre pares de vértices, identificação de articulações e o método proposto por Tarjan (1972). Serão conduzidos experimentos para avaliar o tempo médio requerido para aplicar essas estratégias em grafos aleatórios com 100, 1.000, 10.000 e 100.000 vértices.

A implementação de cada método está descrita abaixo.

2. Desenvolvimento

2.1. Verificação de Caminhos Internamente Disjuntos com Backtracking

Este método busca verificar se existem dois caminhos internamente disjuntos (ou um ciclo) entre cada par de vértices de um bloco no grafo. A técnica utilizada para isso é a busca em profundidade (DFS) combinada com backtracking[7].

O método é aplicado para cada par de vértices (origem, destino) do grafo. Para encontrar os caminhos disjuntos, utilizamos uma DFS com o auxílio de uma pilha.

Durante a busca, caso o caminho atual não leve ao destino ou retorno, ou encontra vértices que já foram percorridos, o algoritmo retorna à última decisão (vértice) e tenta outro caminho - backtracking. Esse retorno é possível graças ao empilhamento dos vértices do caminho, onde para cada vértice que precisamos aplicar o backtracking, desempilhamos e avançamos para outro da lista de adjacência.

Se forem encontrados dois caminhos internamente disjuntos entre os vértices origem e destino, o método retorna true, confirmando que o par de vértices pertence a um bloco biconexo. Caso contrário, retorna false.

Estruturas de Dados:

- A pilha (Stack) é utilizada para armazenar os vértices visitados na ordem de exploração.O Set visitedNodes armazena os vértices já visitados para evitar a repetição de vértices em um mesmo caminho.
- A variável booleana inbound monitora se o destino foi alcançado, indicando que estamos em busca de um segundo caminho para fechar um ciclo (a volta).
- A classe Node representa um vértice no grafo durante a busca. Além de armazenar o valor do vértice (value), ela mantém um índice (nextIndex) que aponta para o próximo vértice adjacente a ser explorado.

2.2. Método identifica articulações testando a conectividade após a remoção de cada vértice

Para desenvolver o método de verificação dos componentes a partir de suas articulações, primeiramente salvamos a vizinhança de cada articulação. Em seguida, removemos as arestas que se conectam a essas articulações. Posteriormente, criamos uma lista encadeada de listas encadeadas para armazenar os componentes conexos que não estão ligados a uma articulação.

Após isso, realizamos uma análise dos vizinhos de cada articulação da seguinte forma:

Para cada articulação, realizamos o procedimento de retirar o vértice e realizar uma busca para verificar se a conectividade do Grafo era mantida.

Após a execução do método, o retorno será de uma lista encadeada com todos os componentes do grafo.

O método que usamos para nos auxiliar foi do site GeeksForGeeks: https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/

2.3. Método proposto por Tarjan (1972)

O algoritmo é baseado nos tempo de descoberta e "Low Values" discutidos no artigo Strongly Connected Components, escrito pelo do cientista da computação e matemático Robert Tarjan[1] no ano de 1972 e a implementação foi adaptada do método descrito no portal GeeksForGeeks[2].

A ideia foi armazenar as arestas visitadas em uma pilha enquanto é feita uma DFS (Busca em Profundidade) em um grafo e continuar procurando por Pontos de Articulação. Assim que um Ponto de Articulação u for encontrado, todas as arestas visitadas durante a DFS a partir do nó u formarão um componente biconexo. Quando a DFS é concluída para um componente conectado, todas as arestas presentes na pilha formarão um componente biconexo.

Se não houver Ponto de Articulação no grafo, então o grafo é biconexo e haverá um componente biconexo que é o próprio grafo.

Complexidade de Tempo: O(N + E), onde N é o número de nós e E é o número de

arestas, como a complexidade de tempo da busca em profundidade. Complexidade de Espaço: O(N), onde N é o número de nós no grafo, foi necessário de um espaço de pilha de recursão O(N) para fazer chamadas DFS.

3. Classes auxiliares

Classes auxiliares são as classes que utilizamos de alguma forma para fazer a execução dos algoritmos: Tarjan, Caminhos Disjuntos e Articulação.

3.1. Geração de grafos aleatórios

A geração de grafos pseudoaleatórios é realizada usando a classe Random do Java. Para cada vértice v do grafo, um número pseudoaleatório é gerado para representar o número de arestas conectadas a esse vértice. Em seguida, para cada inserção de aresta (v,w), outro número pseudoaleatório é gerado para determinar o destino da aresta. Além disso, o método também insere a aresta (w,v), e verifica se a aresta (v,w) já existe; caso exista, essa aresta é ignorada.

3.2. Representação de grafos

Para representar os grafos, foi adotada uma abordagem utilizando uma lista de adjacência, que consiste em um vetor de tamanho m para os vértices e uma lista dinâmica para as n arestas.

4. Testes e Resultados

Os resultados foram obtidos através de uma média de quatro execuções em grafos gerados aleatoriamente.

4.1. Método que verifica a existência de dois caminhos internamente disjuntos (ou um ciclo) entre cada par de vértices do bloco

Qtd. vértices	Tempo de exec.
10	2ms
25	14s
100	Overflow
1.000	Overflow

.

4.2. Método identifica componentes biconexos a partir das articulações do grafo

Os tempos de conclus ao de cada caso para o m etodo est ao descritos abaixo:

Qtd. vértices	Tempo de exec.
100	2ms
1.000	4ms
10.000	8ms
100.000	100ms

.

4.3. Método proposto por Tarjan (1972)

Os tempos de conclusão de cada caso para o método estão descritos abaixo:

Qtd. vértices	Tempo de exec.
100	1,25ms
1.000	37ms
10.000	2.574,75ms
100.000	535.321ms

.

5. Conclusão

Com base na análise dos testes realizados, podemos chegar a algumas conclusões significativas sobre os diferentes métodos de identificação de componentes em grafos:

Melhor Desempenho: O método que identifica os componentes a partir das articulações demonstrou consistentemente o melhor desempenho em termos de tempo de execução. Mesmo com o aumento do número de vértices, o tempo de execução permaneceu relativamente baixo, indicando uma eficiência considerável na identificação dos componentes conexos.

Segunda Melhor Opção: O algoritmo de Tarjan apresentou um desempenho razoável, com tempos de execução mais elevados em comparação com o método das articulações, mas ainda assim sendo uma opção viável para a identificação de componentes em grafos.

Pior Desempenho: O método de verificação de caminhos disjuntos entre pares de vértices internamente apresentou um desempenho muito ruim, com tempos de execução tão altos que alguns testes não puderam ser concluídos devido a estouro de memória. Isso sugere que esse método não é prático para grafos de tamanho significativo.

Além disso, ao analisar as médias de tempo de execução para diferentes tamanhos de grafos, podemos observar que o método das articulações e o algoritmo de Tarjan mantiveram um crescimento assintótico razoável em relação ao aumento do número de vértices, enquanto o método de caminhos disjuntos apresentou um crescimento exponencial, tornando-se rapidamente inviável para grafos maiores.

Portanto, a conclusão geral é que o método de identificação de componentes a partir das articulações é a melhor escolha em termos de desempenho e eficiência, seguido pelo algoritmo de Tarjan, enquanto o método de caminhos disjuntos é impraticável para grafos de tamanho significativo.

6. Referências

[1]Tarjan, R. Depth-First Search and Linear Graph Algorithms. SIAM Journal on Computing 1:2, 146-160, 1972.
[2]Hasija, A. (2023, 13 de fevereiro). Biconnected Components. GeeksforGeeks. Recuperado de https://www.geeksforgeeks.org/biconnected-components/ em 14 de abril de 2024.
[3]Hasija, A. (2024, 16 de fevereiro). Depth First Search or DFS for a Graph. Recuperado de https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/ em 14 de abril de

2024.
[4]https://algs4.cs.princeton.edu/41graph/Biconnected.java.html
[5]https://algs4.cs.princeton.edu/40graphs/
[6]Codigo também em: https://github.com/andresalves01/graphs
[7]Frost, Daniel. Implementing A Heuristic Solution To The Knight's Tour Problem. Medium, 23 de julho de 2020, https://medium.com/@danielfrost_3076/implementing-a-heuristic-solution-to-the-knights-tour-problem-513a73cc7e20.