MAC0323 - EP1: Relatório

Leonardo Heidi Almeida Murakami
11260186

Maio 2025

1 Análise da Complexidade de Tempo para a Detecção de Ciclos

A detecção de ciclos no grafo direcionado é implementada utilizando o algoritmo de busca em profundidade (DFS) na classe `CycleDetector`. Cada vértice é visitado exatamente uma vez pelo algoritmo DFS, e para cada vértice, exploramos todas as arestas adjacentes.

Seja V o número de vértices e E o número de arestas no grafo:

- Cada vértice é processado uma única vez devido à tabela hash `visited` que rastreia os vértices já visitados.
- Para cada vértice, examinamos todas as suas arestas adjacentes.
- As operações de consulta e inserção na tabela hash têm complexidade de tempo de $O(1)$ em caso médio.

Portanto, a complexidade de tempo para a detecção de ciclos é:

$$O(V + E) \tag{1}$$

Esta é a complexidade ótima para a detecção de ciclos em um grafo direcionado, pois precisamos examinar cada vértice e cada aresta pelo menos uma vez.

No pior caso, onde temos colisões frequentes na tabela hash, a complexidade poderia se aproximar de $O(V \cdot L + E \cdot L)$, onde L é o comprimento máximo das listas de colisão na tabela hash. No entanto, com uma função de hash bem distribuída e uma tabela de tamanho adequado (997 no nosso caso), o valor esperado de L é pequeno, mantendo a complexidade próxima de $O(V + E)$.

2 Análise da Complexidade de Tempo para a Listagem de Caminhos

A listagem de todos os caminhos entre dois vértices implementada na classe `PathFinder` também utiliza busca em profundidade, mas sua complexidade é significativamente maior que a detecção de ciclos.

No pior caso, para um grafo direcionado completo (onde cada vértice tem uma aresta para todos os outros vértices), o número de caminhos possíveis entre dois vértices pode ser exponencial.

Seja V o número de vértices:

- Para encontrar todos os caminhos simples (sem ciclos) entre dois vértices, o algoritmo potencialmente explora todas as combinações possíveis de vértices intermediários.
- No pior caso, em um grafo completo, cada vértice (exceto o destino) pode ter até $V - 1$ opções de próximo vértice.
- Para cada caminho, podemos ter de 0 até $V - 2$ vértices intermediários entre o vértice de origem e o destino.
- Isso resulta em uma complexidade de $O(2^V)$, pois cada vértice pode ser incluído ou não no caminho (exceto origem e destino).
- Cada caminho requer $O(V)$ para ser impresso.

Portanto, a complexidade de tempo para a listagem de todos os caminhos no pior caso é:

$$O(V \cdot 2^V) \tag{2}$$

Esta complexidade exponencial é inevitável, pois o número de caminhos possíveis entre dois vértices em um grafo denso pode ser exponencial em relação ao número de vértices.

Na implementação, a tabela hash `visited` é utilizada para evitar ciclos, garantindo que cada caminho seja acíclico, mas isso não reduz a complexidade exponencial no pior caso. No entanto, para grafos esparsos, o número de caminhos e, consequentemente, a complexidade, pode ser muito menor.

3 Uso de Assistentes de IA

3.1 Como utilizei assistentes de IA na resolução

Utilizei o assistente de IA Claude da Anthropic em raras ocasiões durante o desenvolvimento do EP1:

1. Para a criação do arquivo `Makefile`: Como o foco do exercício era na implementação dos algoritmos de grafos e estruturas de dados em Java, utilizei o Claude para gerar um Makefile básico que simplificasse a compilação e execução do projeto. O Makefile gerado contém comandos padrão para compilar os arquivos Java e executar o programa principal, sem nenhuma lógica complexa. Utilizado apenas por conveniência.

2. Para adicionar um comentário de depuração no arquivo `PathFinder.java`: No método `dfs` da classe `PathFinder`, há um comentário de depuração (marcado como "AI Generated Debug Print") que foi gerado pelo Claude. Este comentário sugere um código para imprimir o caminho atual sendo explorado durante a execução do algoritmo, o que seria útil para depuração. Foi utilizado para entender porque o algoritmo não estava imprimindo todos os caminhos possíveis, apenas o primeiro que encontrava.

3. Para a geração do template LaTeX deste relatório: Utilizei o Claude para gerar a estrutura básica do documento LaTeX para este relatório. O assistente criou o template com as seções necessárias (análise de complexidade para detecção de ciclos, análise de complexidade para listagem de caminhos e uso de assistentes de IA), mas todo o conteúdo e análises matemáticas foram desenvolvidos por mim, com base na minha compreensão dos algoritmos implementados.

4. Para a criação do arquivo de teste utilizado para testar a `Main.java`.

3.2 Quais partes pedi ajuda ou esclarecimentos

Não solicitei ajuda ou esclarecimentos sobre o funcionamento dos algoritmos ou estruturas de dados implementados. O código dos algoritmos de detecção de ciclos (DFS), busca de caminhos, implementação da tabela hash e listas de adjacência foram desenvolvidos por mim com base no meu conhecimento prévio e no conhecimento adquirido nas aulas e materiais de referência da disciplina, principalmente os resumos disponíveis no site da disciplina no moodle.

3.3 Se compreendi totalmente o que implementei

Sim, compreendi totalmente todas as implementações do EP1. Cada estrutura de dados e algoritmo implementado foi baseado nos conceitos estudados na disciplina:

- **MyHashTable**: Implementei uma tabela hash com encadeamento para resolução de colisões, compreendendo completamente o funcionamento da função de hash, inserção, busca e a estrutura de nós encadeados.

- **SimpleList**: Implementei uma lista ligada simples com operações básicas, compreendendo o funcionamento da estrutura de nós e os métodos para adicionar elementos, verificar existência e percorrer a lista.

- **DirectedGraph**: Implementei a representação de um grafo direcionado usando listas de adjacência armazenadas em uma tabela hash.

- **CycleDetector**: Implementei e compreendi completamente o algoritmo de detecção de ciclos usando DFS, incluindo o uso das estruturas auxiliares para rastrear vértices visitados e em processamento.

- **PathFinder**: Compreendi e implementei o algoritmo para encontrar todos os caminhos entre dois vértices usando DFS, incluindo o backtracking necessário para explorar todos os caminhos possíveis.

As únicas partes onde utilizei IA foram por pura conveniência, sempre em elementos que não afetariam o entendimento dos algoritmos centrais do exercício ou tornariam o exercício extremamente trivial, apenas auxiliam no bom desenvolvimento de todo o exercício.