UNIVERSIDADE FEDERAL DE LAVRAS

Nome: Alisson Miguel Grilo
Matrícula: 202210112
Disciplina: Algoritmos em Grafos - GCC218
Professor: Vinícius Vitor dos Santos Dias

A base de dados escolhida para estudo e modelagem em grafos foi a rede de co-compra de produtos da Amazon, extraída do site SNAP (*Stanford Network Analysis Project*). Essa base de dados foi obtida por meio de um *web crawler*[1] no site da Amazon, com base na funcionalidade *"Clientes que compraram este item também compraram...".* Em resumo, se um produto p é frequentemente co-adquirido com o produto j, o grafo contém uma aresta direcionada de p para j. Dessa forma, os vértices são caracterizados como os produtos e as arestas como a ação de co-adquirir. Além disso, o grafo é simples e não possui atributos. A representação quantitativa da base coletada é composta por 262.111 vértices e 1.234.877 arestas, sendo aproximadamente 4 o grau médio de cada vértice.

Com os dados coletados, é possível detectar itens de recomendação após uma compra. No entanto, não é trivial utilizar esses dados para a composição de kits de produtos, por exemplo, se um produto c é frequentemente co-adquirido com um produto d, e um produto d é frequentemente co-adquirido com um produto c, é possível montar conjuntos com c e d juntos. Isso pode aumentar o faturamento da organização, pois facilita a compra de kits.

Um bom exemplo é no mundo dos cosméticos, onde um shampoo é frequentemente adquirido com um condicionador e vice-versa. É importante ressaltar que nem sempre essa relação é recíproca. Por exemplo, a compra de um Playstation 4 está fortemente relacionada com a compra de um jogo físico, contudo a compra de um jogo físico raramente acarreta na compra do console. Além disso, essas relações não se limitam apenas a dois produtos.

Assim, o problema consiste em: dado um grafo G representado como descrito acima, identificar suas *componentes fortemente conectadas*, que serão os conjuntos de produtos a serem vendidos. Além de identificar as componentes, é importante exibi-las de forma amigável com a descrição do item e seu código, pois a ferramenta será utilizada por executivos para tomada de decisões estratégicas.

Dessa forma, a escolha recaiu sobre a implementação do algoritmo de Tarjan para a identificação dos Componentes Fortemente Conexos (CFCs), uma versão otimizada em tempo linear da abordagem clássica proposta pelo autor. É relevante destacar que não foram exigidas manipulações complexas de dados, uma vez que o arquivo de texto utilizado como base é simples, composto por pares de números separados por espaços. Esses números correspondem aos índices dos produtos, os quais estão vinculados a informações detalhadas disponíveis em outra fonte, nomeada por *Amazon product metadata*, fornecida no próprio site, possibilitando a obtenção das descrições correspondentes.

Quanto às modificações no próprio algoritmo, a mais notável reside na opção pela busca em profundidade interativa, fazendo uso de uma estrutura de dados Pilha em substituição à busca clássica recursiva. Essa adaptação foi necessária devido à elevada carga de dados contida no arquivo de texto, garantindo eficiência no processamento.

Sobre o algoritmo, é importante notar que ele começa considerando alguns pontos-chave:

1. A Busca em Profundidade (DFS) gera uma estrutura chamada árvore ou floresta DFS. Os Componentes Fortemente Conectados (CFCs) se manifestam como subárvores dentro dessa estrutura.
2. Identificar o começo dessas subárvores é crucial, pois permite imprimir ou armazenar os demais vértices, formando assim um CFC. Importante notar que não há arestas que conectam um CFC a outro.

No contexto do algoritmo de Tarjan Iterativo, as etapas podem ser simplificadas da seguinte maneira:

1. Para cada vértice, são inicializadas duas listas: uma para registrar o número de ordem da DFS (denominada "disc") e outra para armazenar o menor número de ordem DFS alcançável a partir do vértice em questão (chamada "lowest").
2. Inicia-se marcando um vértice como visitado e o colocando em uma pilha.
3. Para cada vizinho desse vértice:
    3.1. Se o vizinho ainda não foi visitado, a DFS é chamada iterativamente para ele.
    3.2. Caso contrário, se o vizinho não foi totalmente processado, atualiza-se o valor "lowest" do vértice atual para o menor entre o "lowest" atual e o "disc" do vizinho.
4. O vértice é marcado como totalmente processado.
5. Se o "lowest" do vértice atual for igual ao seu "disc", então temos um CFC:
    5.1. Retiram-se os vértices da pilha até chegar ao vértice atual.
    5.2. Esses vértices são adicionados ao CFC, e qualquer processamento adicional é realizado conforme necessário.

Em resumo, durante a visita de cada vértice, o algoritmo de Tarjan Interativo atualiza o valor "lowest" para refletir o menor número de ordem DFS alcançável a partir desse vértice. Quando "lowest" é igual a "disc", o vértice marca o início de um CFC, e o algoritmo identifica e trata esse CFC.

Para a implementação desta solução, optou-se pelo uso da linguagem de programação Ruby, na sua versão estável mais recente, a 3.2.2. O código fonte foi desenvolvido com uma estrutura simples, centrada em duas classes principais: a classe grafo, responsável por representar os comportamentos específicos de um grafo e utilizar os algoritmos selecionados, e uma classe principal que organiza a chamada de métodos para a execução adequada.

Além disso, o código inclui uma pasta dedicada à implementação de testes automatizados, uma prática crucial no desenvolvimento de software. Essa abordagem garante a qualidade do código, especialmente após grandes manutenções, contribuindo para a confiabilidade e robustez do sistema.

Para simplificar o processo de configuração e execução do código, foi fornecido um arquivo Makefile com comandos essenciais. Para instalar as dependências necessárias, basta executar o comando *make build*, que realizará a instalação dos pacotes especificados. Em seguida, para executar o código principal, utilize o comando *make run*, o qual iniciará a aplicação Ruby com base no arquivo principal.

A fim de garantir a integridade e funcionalidade do sistema, é altamente recomendável executar os testes automatizados. Isso pode ser feito por meio do comando *make test*, que utiliza a ferramenta de testes automatizados RSpec para validar a implementação correta do código. Ressalta-se a importância de ter a linguagem de

programação Ruby instalada na máquina. Todas essas medidas visam simplificar a avaliação e execução do código.

Os resultados da análise revelam que a base de dados em estudo compreende um total de 10,095 componentes fortemente conexos, cada um representando conjuntos de itens inter-relacionados. A média de aproximadamente 25 vértices por componente sugere uma complexidade significativa nas interconexões. A distribuição do tamanho dos componentes varia notavelmente, com um tamanho máximo identificado de 394.511, indicando conjuntos de itens comuns e altamente relacionados a atividades cotidianas. Por outro lado, observa-se um tamanho mínimo de 1, caracterizando itens com baixa saída e extrema pontualidade.

Além dessas métricas, destacamos conjuntos de dados de particular interesse, os quais podem ser apresentados à diretoria para uma avaliação mais aprofundada. Entre esses conjuntos, identificamos alguns itens notáveis que merecem atenção especial.

Primeiro kit:

- Item 1:
  - ASIN: 0070459363
  - Título: "3,000 Solved Problems in Electrical Circuits"
- Item 2:
  - ASIN: 0028619552
  - Título: "The Complete Idiot's Guide to Geography"

Segundo kit:

- Item 1:
  - ASIN: 1572304162
  - Título: "Therapeutic Communication: Knowing What to Say When"
- Item 2:
  - ASIN: 0831400757
  - Título: "Experiential Therapy for Co-Dependency Manual"

Esses achados promissores não apenas contribuem para a compreensão da extensão da base de dados, mas também oferecem oportunidades valiosas para discussão e análise por parte da diretoria. As imagens associadas a cada item podem fornecer uma representação visual complementar para enriquecer a apresentação desses resultados.

O estudo possui limitações importantes a serem consideradas. O tempo de execução do código pode variar conforme o tamanho da base de dados, impactando a eficiência, especialmente em conjuntos extensos. A representatividade dos Componentes Fortemente Conexos (CFCs) identificados é limitada devido à falta de uma análise detalhada, podendo resultar em uma visão parcial desses componentes. Além disso, o algoritmo fornece apenas o índice do produto associado a cada CFC, exigindo uma busca manual para informações detalhadas.

Apesar das limitações, o estudo fornece insights valiosos para orientar melhorias futuras e destaca a importância de superar esses desafios para aprimorar a utilidade prática do algoritmo em cenários mais complexos.

Referências:

Amazon product co-purchasing network, March 02 2003
Finding Strongly Connected Components: Tarjan’s Algorithm
Tarjan’s Algorithm to find Strongly Connected Components