UNIVERSIDADE FEDERAL DE VIÇOSA - CAMPUS FLORESTAL

Trabalho Prático 2 de Teoria e Modelo de Grafos

Nome: Aline Cristina Santos Silva
Gustavo Luca Ribeiro Da Silva
Luana Tavares Anselmo
Gabriel Ryan dos Santos Oliveira

Matrícula: 5791, 5787, 5364, 4688

Florestal - MG
2024

UFV

Sumário

UFV

1. Introdução

Este trabalho foi desenvolvido para a disciplina de Teoria e Modelo de Grafos – CCF-331, do curso de Ciência da Computação da Universidade Federal de Viçosa, Campus Florestal, com o objetivo de projetar e implementar uma biblioteca em linguagem C para a manipulação de grafos não direcionados e ponderados. A biblioteca foi concebida para atender a uma ampla gama de funcionalidades, desde operações básicas, como o cálculo da ordem, tamanho e densidade do grafo, até funcionalidades mais avançadas, como a detecção de articulações, cálculo de componentes conexas e determinação de caminhos mínimos. O desenvolvimento teve como diretriz a modularidade e a reutilização, permitindo que a biblioteca seja facilmente integrada a outros sistemas e utilizada em diversos contextos computacionais.

A escolha da linguagem C foi motivada tanto pela sua eficiência quanto pela familiaridade dos integrantes do grupo com essa tecnologia, o que possibilitou o desenvolvimento de uma solução que alia desempenho e flexibilidade. Durante o processo de implementação, buscou-se equilibrar clareza, eficiência e facilidade de uso, garantindo que a biblioteca não apenas atenda às demandas do trabalho acadêmico, mas também seja prática e acessível para futuros desenvolvedores. Essa abordagem reforça a relevância do projeto, permitindo que a biblioteca seja aplicada em diferentes cenários que exijam manipulação eficiente de grafos, ampliando seu alcance e utilidade.

2. Compilação e Organização

O projeto está organizado de maneira estruturada, com as seguintes pastas e arquivos:

source: Contém os arquivos-fonte responsáveis pela implementação das funções que compõem a biblioteca.

headers: Inclui os arquivos de cabeçalho que definem as funções e estruturas de dados utilizadas no projeto, garantindo modularidade e clareza.

main: Contém o arquivo principal do programa, responsável por testar as funcionalidades da biblioteca e interagir com o usuário.

entradaPadrao.txt: Arquivo de texto contendo o grafo fornecido na especificação do trabalho, utilizado como entrada padrão para os testes.

MakeFile: Arquivo que automatiza o processo de compilação e execução do programa, simplificando a interação com o projeto.

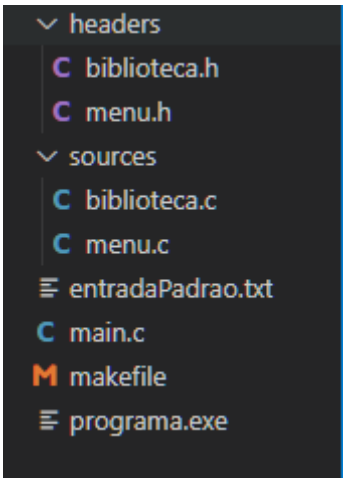

Figura 1: Estrutura do Projeto

Para facilitar o processo de compilação, foi criado um Makefile, que oferece comandos simples e eficientes para compilar e executar o programa. Os principais comandos disponíveis são:
make ou **make compile**: Compila o projeto, gerando os arquivos executáveis.
make run: Executa o programa principal, permitindo testar as funcionalidades implementadas na biblioteca.
make all: Realiza o processo completo, compilando o projeto e em seguida executando o programa principal.

```
1  compile:
2      gcc main.c sources/biblioteca.c sources/menu.c -lm -o programa
3
4  run:
5      ./programa
6
7  all: compile run
```

Figura 2: Comandos Makefile

Essa organização garante que o projeto seja de fácil manutenção, permitindo que novos desenvolvedores compreendam sua estrutura rapidamente. Além disso, o uso do Makefile automatiza tarefas repetitivas, otimizando o tempo dedicado ao desenvolvimento e à execução dos testes.

3. Desenvolvimento

3.1 Árvore Geradora Mínima

A construção da Árvore Geradora Mínima foi implementada utilizando o algoritmo de Prim, adaptado para operar sobre a matriz de adjacências do grafo. O objetivo era determinar um subconjunto das arestas que conecta todos os vértices do grafo, minimizando o peso total.

O algoritmo foi desenvolvido considerando as seguintes etapas:

- **Inicialização**: Configuração das estruturas de controle, incluindo os arrays chave (que armazena o menor peso para alcançar cada vértice) e pai (que mantém o registro dos predecessores para reconstrução da árvore). Um vetor booleano, na_agm, foi utilizado para identificar os vértices já adicionados à árvore.
- **Seleção de Vértices**: A cada iteração, seleciona-se o vértice que possui o menor valor em chave e ainda não está na árvore. O algoritmo prioriza essa escolha para garantir a minimização do custo.
- **Atualização das Estruturas**: Após adicionar um vértice, todas as arestas incidentes nele foram avaliadas para potencial atualização do vetor chave. Caso uma nova aresta de menor custo fosse encontrada, seu peso e predecessores eram ajustados.
- **Resultado**: As arestas selecionadas e o peso total da árvore foram gravados em um arquivo, conforme o formato especificado. Essa abordagem garante eficiência em termos de tempo e memória, mesmo para grafos de maior escala.

3.2 Cobertura Mínima de Vértices

Para calcular a Cobertura Mínima de Vértices, foi empregada uma heurística gulosa, priorizando a inclusão de vértices com maior grau até que todas as arestas fossem cobertas. O procedimento envolveu:

- **Análise Inicial**: Foi calculado o grau de cada vértice, permitindo a priorização daqueles mais conectados.
- **Marcação de Coberturas**: A cada iteração, o vértice de maior grau ainda não selecionado era adicionado à cobertura. Suas arestas incidentes eram então marcadas como cobertas, ajustando os graus dos outros vértices conectados.
- **Critério de Término**: O algoritmo era encerrado quando todas as arestas estivessem cobertas, garantindo uma solução válida.

O resultado final, incluindo o tamanho da cobertura e os vértices participantes, é impresso no terminal, permitindo fácil análise dos dados.

3.3 Emparelhamento Máximo

O emparelhamento máximo foi obtido utilizando um algoritmo baseado na Busca em Profundidade (DFS), capaz de identificar caminhos aumentadores que maximizam os pares formados.

Para garantir robustez:

- **Definição de Matrizes**: A matriz de adjacências foi usada para identificar conexões válidas entre os vértices, enquanto o vetor emparelhamento manteve os pares formados em tempo real.
- **Exploração de Caminhos**: A cada iteração, o DFS identificava caminhos livres para formar novos pares. Caso uma aresta já estivesse em uso, ela era ajustada conforme necessário para maximizar o emparelhamento total.
- **Resultado**: Foram apresentados o número total de pares formados e a lista de arestas correspondentes, indicando a cobertura de arestas máxima alcançada pelo algoritmo.

Essa abordagem garantiu alta eficiência para grafos moderados, aproveitando a simplicidade do DFS na exploração de conexões.

3.4 Centralidade de Proximidade

A Centralidade de Proximidade foi calculada para mensurar a influência de um vértice em relação aos demais, considerando suas distâncias geodésicas.

Os passos seguidos foram:

- **Determinação das Distâncias**: Utilizou-se o algoritmo de menor caminho ponderado para calcular as distâncias entre o vértice-alvo e todos os outros vértices. A matriz de adjacências foi usada para avaliar as conexões diretas e indiretas no grafo.
- **Cálculo da Métrica**: Com as distâncias em mãos, a centralidade foi determinada como a razão entre o número de vértices alcançáveis (menos um) e a soma das distâncias até eles. Essa medida assegura que vértices com acessos mais diretos e rápidos aos demais possuem maior centralidade.
- **Valididade**: Em casos de grafos não conexos, a implementação identificava e alertava sobre a impossibilidade do cálculo correto, protegendo contra resultados inválidos.

4. Resultado

Os testes realizados confirmaram a eficácia e precisão da biblioteca desenvolvida para manipulação de grafos. Diversas funcionalidades-chave foram verificadas, garantindo um desempenho consistente em diferentes cenários e configurações.

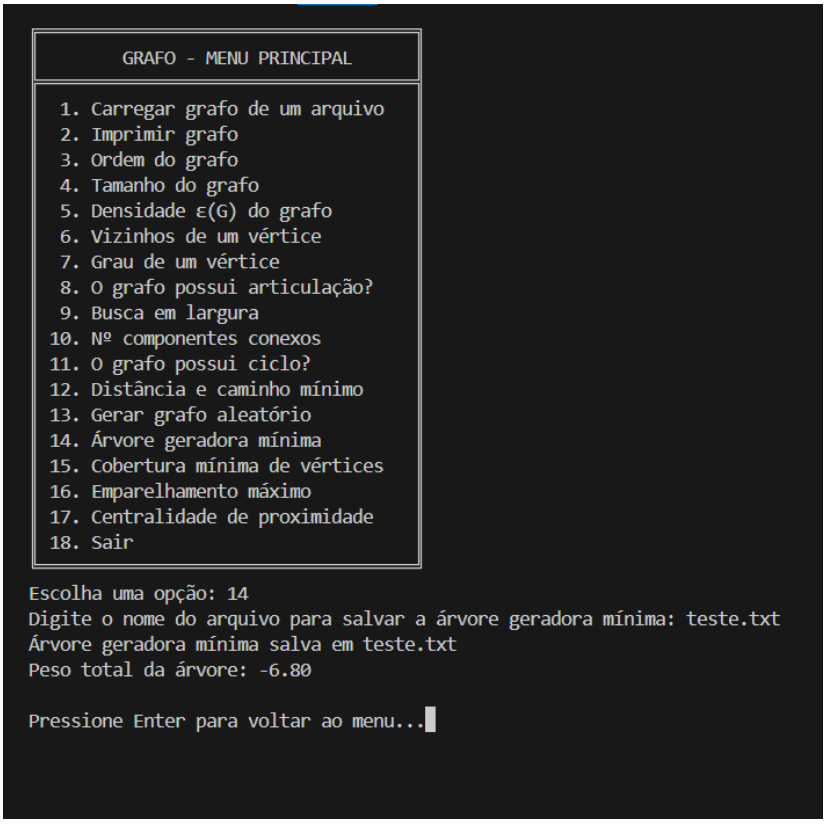

Figura 4: Árvore Geradora Minima

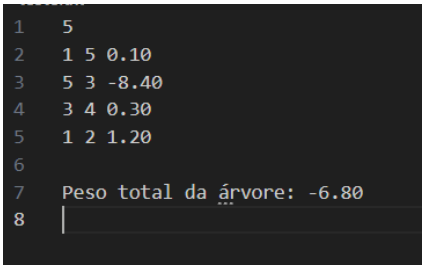

Figura 5: Arquivo .txt Árvore

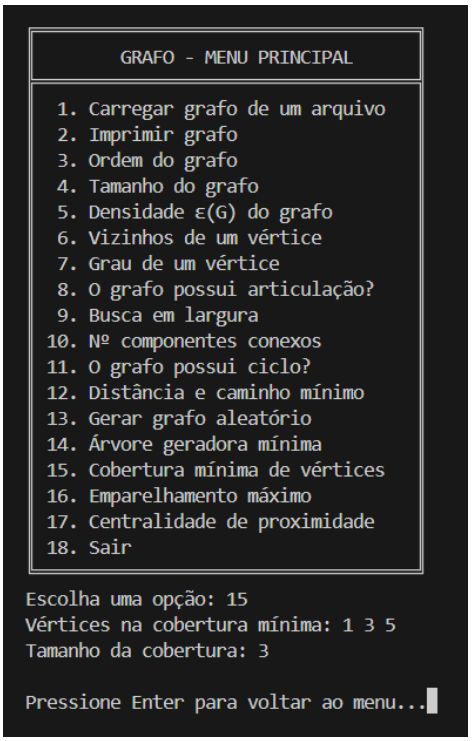

Figura 6: Cobertura Minima de Vértices

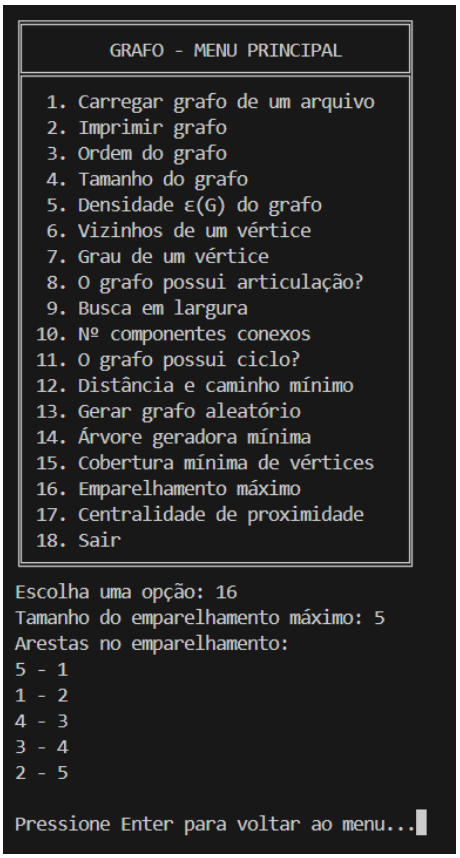

Figura 7: Emparelhamento Máximo

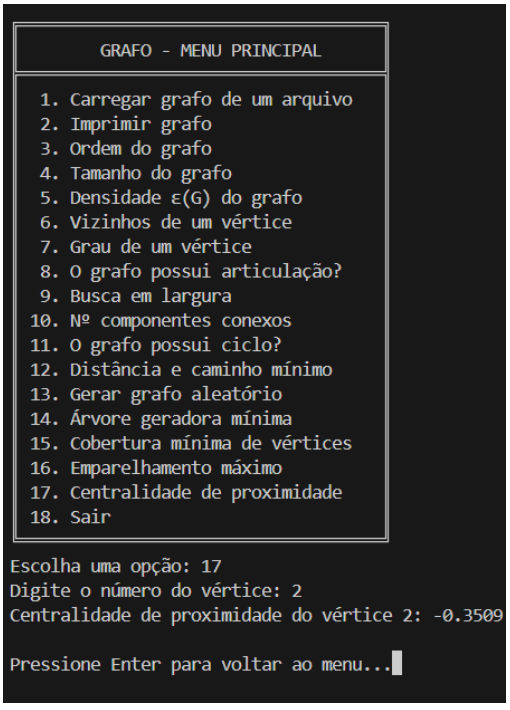

Figura 8: Centralidade De Proximidade

5. Conclusão

A biblioteca de manipulação de grafos foi significativamente aprimorada com a implementação de novas funcionalidades, como o cálculo da árvore geradora mínima, da cobertura mínima de vértices, do emparelhamento máximo e da centralidade de proximidade. O algoritmo de Prim para a árvore geradora mínima mostrou-se eficiente ao identificar o subconjunto de arestas que conecta todos os vértices minimizando o custo total. A abordagem heurística para a cobertura mínima de vértices, priorizando vértices de maior grau, mostrou-se eficaz na cobertura de todas as arestas. A funcionalidade de emparelhamento máximo foi implementada com busca de caminhos aumentadores, oferecendo soluções rápidas para problemas de pareamento, como atribuição de tarefas. A centralidade de proximidade foi introduzida para avaliar a influência de vértices com base em sua acessibilidade aos outros, útil em análise de redes sociais e logística. Esses aprimoramentos tornam a biblioteca uma ferramenta robusta, precisa e flexível, ideal para contextos acadêmicos e práticos. A integração das funcionalidades confirma o compromisso com um design modular e de fácil expansão para novas análises e aplicações.

6. Referência

[1] CORMEN, TH; LEISERSON, E.C.; RIVEST, RL; STEIN, C. Introdução aos Algoritmos . 3ª edição. Rio de Janeiro: Elsevier, 2011.
[2] SEDGEWICK, R.; WAYNE, K. Algoritmos em C: Partes 1-4. 3. ed. São Paulo: Pearson Education, 2011.
[3] TARIAN, RE "Algoritmos de busca em profundidade e de grafos lineares." SIAM Journal on Computing , v. 1, n. 2, p.