Relatório A2

INE5413 - Grafos

Alunas: Jéssica Regina dos Santos e Myllena da Conceição Correa

---

Neste relatório, descrevemos as estruturas de dados utilizadas na implementação da Atividade A2.

1. Componentes Fortemente Conexas - arquivo A2_1.

Para implementar o algoritmo de busca em profundidade (DFS) e identificar as componentes fortemente conexas (CFCs), utilizamos as seguintes estruturas de dados. Essas estruturas foram escolhidas com foco na facilidade de manipulação e eficiência de tempo.

- Dicionários C, T, Ancestrais e F:
  Objetivo: armazenar o estado de cada vértice durante a execução do DFS.
  - C armazena se o vértice foi visitado ou não.
  - T registra o tempo de descoberta de cada vértice.
  - Ancestrais mantém o vértice ancestral de cada vértice visitado.
  - F guarda o tempo de finalização de cada vértice.

  Justificativa: Usar dicionários permite acesso rápido aos estados de cada vértice por meio de sua chave (número do vértice). Como cada vértice é indexado diretamente, a busca e atualização do estado em cada etapa do DFS são eficientes, facilitando o rastreamento do processo em O(1) para cada consulta.
- Lista A_t para as arestas do grafo transposto:
  Objetivo: construir o grafo transposto para identificar as componentes fortemente conexas.
  Justificativa: Uma lista de arestas permite armazenar cada aresta com sua direção invertida, o que é muito importante para executar o DFS no grafo transposto e encontrar as CFCs de forma eficiente. Como os vértices e arestas já estão armazenados no grafo original, construir o transposto dessa forma é bem mais simples.
- Lista ‘componentes’ para armazenar componentes fortemente conectadas:
  Objetivo: armazenar os conjuntos de vértices que formam cada componente fortemente conexa.
  Justificativa: Usar uma lista de listas permite armazenar cada CFC como uma lista de vértices. À medida que cada CFC é identificada, ela é adicionada a ‘componentes’, que pode ser usada para exibir o resultado final de forma organizada.
- Variáveis auxiliares como ‘temp’ armazenam temporariamente os vértices de uma CFC enquanto percorremos o grafo transposto. Usar uma lista temporária para cada CFC facilita a organização dos vértices até que a componente esteja completa, permitindo a separação e armazenamento de CFC diferentes.

2. Ordenação Topológica - arquivo A2_2.

Nosso algoritmo para encontrar a OT foi implementado, analisado diversas vezes, porém não obtivemos o resultado final que gostaríamos. Mesmo assim, a dupla achou importante relatar o que desenvolvemos.

Neste código, utilizamos as seguintes estruturas de dados:

- Dicionário C:
  Objetivo: marcar se cada vértice foi visitado com valores booleanos True ou False.
  Justificativa: permite acesso rápido e direto pelo identificador do vértice, facilitando a verificação de visitação em tempo constante O(1).

- Dicionário T (Tempo de descoberta):
  Objetivo: armazenar o tempo em que cada vértice foi visitado pela primeira vez durante a execução do algoritmo.
  Justificativa: a estrutura de dicionário é novamente usada aqui para armazenar e acessar rapidamente os tempos associados a cada vértice, o que facilita o rastreamento do tempo de descoberta.
- Dicionário F (Tempo de finalização):
  Objetivo: armazenar o tempo de finalização para cada vértice, indicando quando todos os vértices adjacentes a ele foram processados.
  Justificativa: similar ao dicionário T, essa estrutura permite acessar rapidamente o tempo de finalização de cada vértice, necessário para a ordenação topológica
- Lista V:
  Objetivo: armazenar os vértices do grafo. Ela é inicializada com os índices dos vértices incrementados em 1 para corresponder ao índice baseado em 1.
  Justificativa: a lista é simples e eficiente para armazenar e iterar sobre vértices de maneira ordenada, permitindo fácil acesso a cada vértice pelo índice.
- Lista O (Ordenação Topológica):
  Objetivo: esta lista armazena os vértices na ordem topológica. Os vértices são inseridos no início da lista para que a ordenação resultante esteja correta ao final do algoritmo (no nosso caso, acabou falhando. Imagem abaixo).

```
PS C:\Users\jessi\OneDrive\Área de Trabalho\24-2\Trabalhos grafos\T2-Grafos> & C:/Users/jessi/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/jessi/OneDrive/Área de Trabalho/24-2/Trabalhos grafos/T2-Grafos/A2_2.py"
['Acordar', 'Tomar cafÃ© da manhÃ£', 'Ler o jornal', 'Tomar banho', 'Vestir camisa', 'CalÃ§ar meias', 'Vestir roupa de baixo', 'Vestir calÃ§as', 'CalÃ§ar tÃªnis', 'Sair', 'Escovar os dentes'] <-- 'Escovar os dentes' na posição errada
```

  Justificativa: a lista permitiria inserir elementos de forma eficiente no início e facilitaria a coleta dos vértices ordenados de maneira direta, pois é possível construir a lista à medida que o grafo é percorrido.
- Variável tempo:
  Objetivo: utilizada para acompanhar o tempo de descoberta e finalização dos vértices durante a execução da busca em profundidade.
  Justificativa: uma variável inteira é simples e eficiente para manter o tempo crescente enquanto os vértices são visitados, garantindo que cada vértice receba um tempo de descoberta e finalização únicos.

3. Algoritmo de **Kruskal** ou Prim - arquivo A2_3.
O algoritmo implementado pela dupla foi o de Kruskal para encontrar a árvore geradora mínima (AGM) de um grafo ponderado, onde as seguintes estruturas de dados foram utilizadas:

- Dicionário S para representar as componentes conectadas:
  Propósito: controlar quais vértices pertencem a qual componente conectada, evitando ciclos ao unir componentes.
  Justificativa: usar um dicionário onde cada chave representa um vértice e seu valor é uma lista de vértices pertencentes à mesma componente conectada. Isso facilitou a verificação de conectividade entre vértices em tempo constante. Para cada nova aresta, o dicionário é atualizado, o que permite um controle das conexões.
- Lista A para armazenar as arestas da AGM:
  Propósito: guardar as arestas que fazem parte da AGM.
  Justificativa: esta lista armazena as arestas selecionadas. Como o algoritmo de Kruskal adiciona arestas uma a uma, a lista permite uma inserção rápida e organizada das arestas finais.

- Lista E para armazenar as arestas com pesos ordenados:
  Propósito: guardar e organizar as arestas do grafo de forma que possam ser processadas em ordem crescente de peso.
  Justificativa: Uma lista é foi uma escolha “natural” para armazenar as arestas devido à facilidade de ordenação com o método sort().
- Lista X como auxiliar para unir componentes:
  Propósito: armazenar temporariamente os vértices das duas componentes conectadas para realizar a união delas.
  Justificativa: X serve para facilitar a atualização do dicionário S após a união de duas componentes. Ao combinar as listas de vértices das duas componentes conectadas, X permite atribuir a mesma lista a todos os vértices envolvidos, mantendo uma estrutura de conectividade coerente.
- A variável total_peso armazena a soma dos pesos das arestas da AGM. Esta variável permite fácil acompanhamento do custo total da AGM sem necessidade de estruturas muito complexas.