Análise de Complexidade Temporal No Pior Caso

USEI17

1. Inicialização

- **Criação do grafo e dos vértices "START" e "END"** Essas operações são de complexidade constante: O(1)

---

2. Leitura e criação dos vértices

- **Leitura do arquivo e adição de vértices ao grafo**

Essa etapa percorre todas as linhas do arquivo (exceto o cabeçalho) para processar cada atividade:

  - Leitura e separação dos valores por vírgulas: Se há linhas no arquivo, esta parte tem complexidade O(n).
  - Criação de vértices e inserção em um mapa: A operação de inserir no mapa () é em média, mas repetida vezes. **Total desta etapa:** O(n).

---

3. Leitura e criação das arestas

- **Leitura do arquivo novamente para criar as arestas**

Cada linha do arquivo é processada para identificar predecessores e criar conexões.

  - Para cada linha ():
    - **Extração dos predecessores:** Esta operação pode incluir até predecessores por linha (número máximo de predecessores em uma atividade). A extração e iteração pelos predecessores têm complexidade O(m).
    - **Adição de arestas ao grafo:** Para cada predecessor, a adição de uma aresta ao grafo tem complexidade , mas é repetida vezes. O(1)

Assim, no pior caso: O(n×m)O(n * m)

Onde:

- nn = número de atividades (linhas do arquivo)
- mm = número médio de predecessores por atividade

Total desta etapa: O(n×m) O(n * m).

---

4. Conexão de vértices sem predecessores ao "START"

- **Identificação de vértices sem predecessores:** Cada vértice é verificado (total) para determinar se possui predecessores.
- **Complexidade:** O(n)

---

5. Conexão de vértices sem sucessores ao "END"

- **Identificação de vértices sem sucessores:** Para cada vértice, são verificadas suas arestas de saída (chamada ao método outgoingEdges). No pior caso, isso resulta em , pois cada vértice é processado uma vez. O(n)

---

6. Tratamento de exceções

- **Bloco catch e retorno:** Essas operações têm complexidade constante , sendo irrelevantes para o cálculo da complexidade total.O(1)

Complexidade Total

Somando todas as etapas, temos:

O(1)+O(n)+O(n×m)+O(n)+O(n)= O(n×m) O(1) + O(n) + O(n * m) + O(n) + O(n) = O(n * m)

No pior caso, a complexidade temporal do método é: O(n×m)

Onde:

- n = número de atividades (linhas do arquivo)
- m = número médio de predecessores por atividade

USEI20

1. Método TopologicalSort

O método realiza a ordenação topológica de um grafo e pode ser dividido em partes:

1. **Inicialização de inDegree:**
    - Para cada vértice no grafo, o grau de entrada é calculado com . Se houver arestas no grafo, cada chamada a g.inDegree pode visitar todas as arestas incidentes. g.inDegree(vertex)g.inDegree(vertex)
    - **Complexidade:** O(n+m)
2. **Adição de vértices com grau de entrada zero à fila:**
    - Itera por todos os vértices para verificar . inDegreeinDegree
    - **Complexidade:** O(n)
3. **Processamento da fila:**
    - Para cada vértice removido da fila, percorremos os seus vértices adjacentes (arestas no total).
    - **Complexidade:** O(n+m)
4. **Verificação de ciclos:**
    - Comparação do tamanho de result com o número de vértices é O(1).

Complexidade total do TopologicalSort: O(n+m)

Onde:

- n é o número de vértices.
- m é o número de arestas.

---

2. Método calculateEarliestStartAndFinish

Este método calcula os tempos de início e término mais cedo de um grafo PERT.

1. **Ordenação topológica:**
    - Chamada ao TopologicalSort.
    - **Complexidade:** O(n+m)
2. **Inicialização dos tempos iniciais e finais:**

- Itera por todos os vértices.
- **Complexidade:** O(n)

3. **Atualização dos tempos para cada vértice:**
    - Para cada vértice, percorremos suas arestas de saída (no total).
    - Atualizar os tempos de início e término de cada vértice adjacente é O(1).
    - **Complexidade:** O(n+m)

Complexidade total do calculateEarliestStartAndFinish: O(n+m)

---

3. Método calculateLatestStartAndFinish

Este método calcula os tempos de início e término mais tarde de um grafo PERT.

1. **Ordenação topológica e reversão:**
    - Chamada ao TopologicalSort e reversão da lista (linear em).
    - **Complexidade:** O(n+m).
2. **Inicialização dos tempos mais tarde:**
    - Itera por todos os vértices.
    - **Complexidade:** O(n)
3. **Atualização dos tempos para cada vértice:**
    - Para cada vértice (), percorremos suas arestas de entrada (no total).
    - Atualizar os tempos de início e término é O(1).
    - **Complexidade:** O(n+m)

Complexidade total do calculateLatestStartAndFinish: O(n+m)

- **TopologicalSort:** O(n+m)
- **calculateEarliestStartAndFinish:** O(n+m)
- **calculateLatestStartAndFinish:** O(n+m)

Essas complexidades são lineares em relação ao número de vértices (n) e arestas (m), garantindo eficiência para grafos direcionados acíclicos típicos de PERT/CPM.
