Entrega 2 PI – Teoria da Computação e Linguagens Formais

Tema: Sistema de Monitoramento e Automação para Sala Maker

Aplicação de Técnicas e Algoritmos de Otimização

Aplicação de Técnicas e Algoritmos de Otimização na Lógica Embarcada (ESP32)

Objeto da Análise: Código-fonte C++ no ESP32 (lógica de controle dos atuadores e leitura de sensores)

1. Aplicação da Técnica de Otimização: Análise e Minimização de Autômatos Finitos

A lógica de controle de atuadores no ESP32, que responde às requisições do servidor Flask, é modelada como um Autômato Finito Determinístico (AFD). O foco da análise é a rota /ascensor, que controla os motores de subir/descer do equipamento, pois possui múltiplos estados de operação.

Modelagem da Máquina de Estado (AFD do Controle Ascensor)

A funcionalidade do ascensor possui três estados de operação que se alteram a partir de comandos recebidos via rede.

- **Estados (Q):**
  $$Q_{Ascensor} = \{E_{Parado}, E_{Subindo}, E_{Descendo}\}$$
- **Alfabeto de Entrada (Σ):** Comandos recebidos via parâmetro cmd na requisição HTTP.
  $$\Sigma = \{Comando_{Levantar}, Comando_{Descer}, Comando_{Parar}\}$$
- **Conjunto de Saídas** (Γ): Ações diretas nos pinos de saída do motor.
  $$\Gamma = \{Motor\_SOBE\ (HIGH/LOW), Motor\_DESCE\ (HIGH/LOW)\}$$

Função de Transição (δ): (Implementada no *handler* da rota /ascensor)

Estado Atual	**Entrada Σ**	**Próximo Estado Q**	**Saída Γ (Ação no Pino)**
$E_{Qualquer}$	$Comando_{Levantar}$	$E_{Subindo}$	$D(SOBE)=HIGH, D(DESCE)=LOW$
$E_{Qualquer}$	$Comando_{Descer}$	$E_{Descendo}$	$D(SOBE)=LOW, D(DESCE)=HIGH$

$E_{Qualquer}$	$Comando_{Parar}$	E_{Parado}	$D(SOBE)=LOW, D(DESCE)=LOW$

Otimização e Conclusão de Minimização
O objetivo da minimização é reduzir o número de estados, simplificando a implementação.

Análise:

O AFD do controle ascensor é, por natureza, um Autômato Finito Determinístico Mínimo.

Cada um dos 3 estados (E_{Parado}, $E_{Subindo}$, $E_{Descendo}$) corresponde a uma combinação única de saídas (ações físicas) nos pinos do motor. Portanto, nenhum par de estados é indistinguível; eles não podem ser fundidos, pois a transição para qualquer um deles resulta em uma saída física diferente.

Conclusão da Otimização: A modelagem da lógica do ascensor já representa a estrutura formal mais otimizada (minimal) possível, o que garante a máxima simplicidade e clareza da lógica embarcada.

2. Aplicação do Algoritmo de Otimização: Análise de Complexidade (Big O)

A análise do algoritmo de otimização se concentra na eficiência de tempo da lógica de decisão dentro do ESP32, fundamental para garantir o requisito de baixa latência do sistema de controle.

Análise da Função de Transição (δ) nas Rotas de Controle
A função de transição é realizada pelos *handler functions* das rotas HTTP no código C++ do ESP32.

Cenário de Teste: Análise da rota /ascensor e /solda_toggle.

C++
```
// Lógica de decisão do Ascensor e Solda
// É uma sequência de comparações com strings fixas:
if (cmd == "levantar") { /* ... */ }
else if (cmd == "descer") { /* ... */ }
else if (cmd == "parar") { /* ... */ }
// ...
```

A execução dessa lógica envolve um número **fixo e limitado** de operações de comparação de *string* (máximo 3 no ascensor e 2 na solda).

Complexidade de Tempo Formal (Notação Big O)
A complexidade é formalizada utilizando a notação Big O.

Conclusão Formal:

A complexidade de tempo do algoritmo de decisão (Função de Transição δ) para o controle de atuadores é:

$$\mathbf{O(1)}$$

Complexidade de Tempo Constante

Justificativa:

1. **Tempo Constante:** O tempo que o ESP32 leva para processar o comando e acionar o pino não varia com o volume de dados de entrada (o tamanho do comando 'levantar' ou 'ON' é fixo) ou com a escala do sistema.
2. **Validação de Eficiência:** A complexidade $O(1)$ demonstra que a lógica de controle implementada no hardware é **máxima em eficiência computacional**.
3. **Conexão com Latência:** Este resultado reforça que qualquer lentidão percebida (latência) reside na **camada de comunicação (WiFi/HTTP)** e não na performance do microcontrolador na execução do algoritmo de controle em si.

3. Aplicação do Algoritmo de Otimização: Análise de Sensores

A rota /status é responsável por ler os sensores (pinos FERRAMENTA_1 a FERRAMENTA_4) e formatar a resposta JSON.

C++
```
// Lógica de leitura de Sensores (C++)
bool status_ferramenta_1 = (digitalRead(FERRAMENTA_1) == HIGH);
// ... 4 leituras no total
// ... formatar string JSON de tamanho fixo
```

Complexidade:

O algoritmo executa um número fixo de leituras de pino (digitalRead()) e constrói uma string JSON com um número fixo de campos. Portanto, essa rotina de aquisição de dados também opera em tempo constante.

$$\mathbf{O(1)}$$
Conclusão Final:

A lógica embarcada do ESP32, que constitui a camada de interação com o mundo físico, utiliza algoritmos de controle e aquisição de dados com complexidade $O(1)$, confirmando que a otimização algorítmica foi alcançada em toda a arquitetura de software.

Aplicação de Técnicas e Algoritmos de Otimização na Lógica do Servidor

Foco da Análise: Lógica de Decisão e Controle do Ar-Condicionado (AC)

1. Aplicação da Técnica de Otimização: Minimização de Autômatos Finitos

A lógica de automação implementada no servidor do projeto (Flask/Python) para o controle de temperatura é modelada como um Autômato Finito Determinístico (AFD). A otimização se dá através da técnica de minimização de estados.

Modelagem da Máquina de Estado Original (AFD do Controle de Temperatura)

O sistema foi inicialmente modelado com uma granularidade maior, utilizando 5 estados para garantir a precisão nas transições do AC.

- **Conjunto de Estados (Q):**

  $$Q_{Original} = \{E_{Ocioso}, E_{Resfriamento\_Leve}, E_{Resfriamento\_Alto}, E_{Aquecimento}, E_{Alerta}\}$$

- **Alfabeto de Entrada (Σ):** Entradas do sensor de temperatura (ex: $T_{Ideal}: 22-24^\circ C$).

  $$\Sigma = \{T_{Ideal}, T_{Quente\_24}, T_{Quente\_26}, T_{Frio}, T_{Emergência}\}$$

Transição Crítica (Exemplo)	Estado Atual	Entrada	Próximo Estado
Passo 1	E_{Ocioso}	T_{Quente_24}	$E_{Resfriamento\_Leve}$
Passo 2	$E_{Resfriamento\_Leve}$	T_{Quente_26}	$E_{Resfriamento\_Alto}$
Retorno	$E_{Resfriamento\_Leve}$ ou $E_{Resfriamento\_Alto}$	T_{Ideal}	E_{Ocioso}

Aplicação da Minimização de Estados

A técnica de minimização busca identificar e fundir estados indistinguíveis – aqueles que, a partir de certo ponto, levam às mesmas saídas e aos mesmos estados subsequentes para as mesmas entradas.

Otimização Identificada:

Os estados $E_{Resfriamento\_Leve}$ e $E_{Resfriamento\_Alto}$ foram identificados como candidatos à fusão, pois, apesar de acionarem comandos de saída diferentes, as suas transições de retorno para o estado E_{Ocioso} sob a entrada T_{Ideal} são idênticas.

Resultado da Otimização (AFD Minimizada):

Redução de 5 estados para 4 estados no modelo final de controle.

- Conjunto de Estados Minimizados (Q_{Min}):
  $$Q_{Min} = \{E_{Ocioso}, E_{Resfriamento}, E_{Aquecimento}, E_{Alerta}\}$$
  (O estado $E_{Resfriamento}$ absorve a lógica de acionamento em duas fases, mas simplifica o grafo de transição principal).

Redução da Complexidade Lógica

A minimização resulta na simplificação da lógica combinacional do código-fonte, reduzindo o número de variáveis de estado e de *flags* necessários para rastrear a posição do autômato. Em um sistema embarcado (ESP32) e no servidor (Flask), a redução do número de estados:

1. Diminui a ocupação de memória (se a máquina for implementada via tabela de transição).
2. Simplifica a estrutura de controle (if/elif no Python), o que contribui para a manutenibilidade e confiabilidade do software, um dos objetivos dos Métodos Formais.

2. Aplicação do Algoritmo de Otimização: Análise de Complexidade (Big O)

O requisito de tempo próximo ao real (baixa latência) no projeto é diretamente validado através da análise de eficiência computacional do algoritmo de decisão central, utilizando a notação Big O.

Foco no Algoritmo de Transição (δ)

O algoritmo de otimização aqui analisado é a Função de Transição (δ) do AFD, que é a rotina executada pelo servidor (no código app.py) para:

1. Receber o dado do sensor.
2. Avaliar o estado atual.
3. Determinar o próximo estado e a ação do atuador.

Complexidade de Tempo da Decisão Lógica

O algoritmo de transição δ é implementado por uma sequência fixa e limitada de operações de comparação (Ex: if temperatura > X and estado == Y:).

Formalização da Complexidade (Notação Big O):

A eficiência de tempo para a Função de Transição (δ) é: $$\mathbf{O(1)}$$

Complexidade de Tempo Constante

Justificativa e Validação da Baixa Latência

1. **Independência de n:** A complexidade $O(1)$ significa que o tempo de execução da lógica de decisão é constante e não cresce com o volume de dados de entrada (n) ou com o número de estados do sistema (já que estes são finitos e fixos).
2. **Garantia de Desempenho:** A complexidade $O(1)$ atesta que o algoritmo de decisão é o mais eficiente possível em termos computacionais, provendo uma base sólida para cumprir o requisito de baixa latência. O fator limitante da latência total do sistema reside, portanto, no tempo de comunicação de rede (HTTP Request), e não na complexidade do algoritmo lógico.

Conclusão:

A aplicação da técnica de minimização de estados e a validação da complexidade $O(1)$ da função de transição demonstram a correta utilização dos conceitos de Teoria da Computação para otimizar a lógica, garantindo a eficiência e simplificação do sistema de automação.