Complexidade de Algoritmos

Depuração de Algoritmos Recursivos

1 Análise de Complexidade:

Algorithm 1: Cálculo da função recursiva x(n)

Input: Algoritmo para x(n) avaliado para $n = n_0$
Output: Tempo médio do algoritmo x(n) para $n = n_0$, N_{Max}
N = 0;
T = 0;
if $N \leq N_{Max}$ **then**
  return T/N_{Max};
end
else
  N = N + 1;
  T = T + tempo_execucao($x(n_0)$);
  Execute o algoritmo para calcular $x(n_0)$;
end

2 Função recursiva para análise:

Considere a seguinte função recursiva definida para valores inteiros positivos n:

$$x(n) = \begin{cases} 1 & \text{se } n \leq 1, \\ x(n-1) + x(n-2) & \text{caso contrário.} \end{cases}$$

3 Depure o código com o GDB

Para depurar o código com o GDB, siga os passos abaixo:

1. Compile o código com a flag `-g` para incluir informações de depuração:

```
gcc -g -o programa programa.c
```

2. Inicie o GDB com o executável gerado:

```
gdb ./programa
```

3. No GDB, defina um ponto de interrupção (breakpoint) na função desejada:

```
(gdb) break nome_da_funcao
```

4. Execute o programa no GDB:

```
(gdb) run
```

5. Quando o programa parar no ponto de interrupção, você pode inspecionar variáveis:

```
(gdb) print variavel
```

6. Para avançar linha a linha no código:

```
(gdb) next
```

7. Para sair do GDB:

```
(gdb) quit
```

8. Para reiniciar a execução do programa desde o início:

```
(gdb) run
```

3.1 Referências adicionais

Para mais instruções sobre o uso do GDB, acesse o arquivo anexado no SIGAA com as principais funções e exemplos práticos. Certifique-se de revisar os comandos básicos e avançados para uma depuração eficiente.

4 Cronometrando o algoritmo

1. Compile o código com a flag `-g` e execute o programa para diferentes valores de n.
2. Utilize a função `clock()` da biblioteca `time.h` para medir o tempo de execução do algoritmo.
3. Repita a execução do algoritmo N_{Max} vezes para calcular o tempo médio.
4. Exemplo de código em C para medir o tempo de execução:

```c
#include <stdio.h>
#include <time.h>

int x(int n) {
    if (n <= 1) return 1;
    return x(n - 1) + x(n - 2);
}

int main() {
    int n = 10; // Valor de n
    int N_Max = 1000; // Numero de repeticoes
    double total_time = 0;

    for (int i = 0; i < N_Max; i++) {
        clock_t start = clock();
        x(n);
        clock_t end = clock();
        total_time += (double)(end - start) / CLOCKS_PER_SEC;
```

```
    }

    printf("Tempo medio: %f segundos\n", total_time / N_Max);
}
```

(a) Compile e execute o programa para obter o tempo medio:

```
gcc -o tempo tempo.c
./tempo
```

(b) Aumente gradativamente o tamanho do problema até encontrar um valor de n onde o tempo de execução se torne proibitivo. Registre os tempos de execução para diferentes valores de n e analise o crescimento do tempo em função de n.

(c) Compare os resultados obtidos com a análise teórica da complexidade do algoritmo. Verifique se o comportamento observado experimentalmente está de acordo com a complexidade esperada.

5 Melhorando o algoritmo - Algoritmos Otimizados

1. Modifique o algoritmo para calcular valores maiores que os proibitivos, mantendo a recursão, mas otimizando seu desempenho.

2. Reescreva o algoritmo para calcular valores maiores que os proibitivos, eliminando a recursão e utilizando uma abordagem iterativa.

3. Desenvolva uma versão do algoritmo que calcula valores maiores que os proibitivos utilizando uma fórmula fechada.

4. Compare o tempo médio de execução dos algoritmos otimizados com o algoritmo original. Plote os resultados em um gráfico, utilizando qualquer ferramenta de sua preferência, e inclua os tempos do algoritmo original para os valores possíveis de calcular.

6 Apresentação dos resultados

Da seção anterior, três gráficos devem ser gerados. Apresento um exemplo de como apresentar esses gráficos logo abaixo:

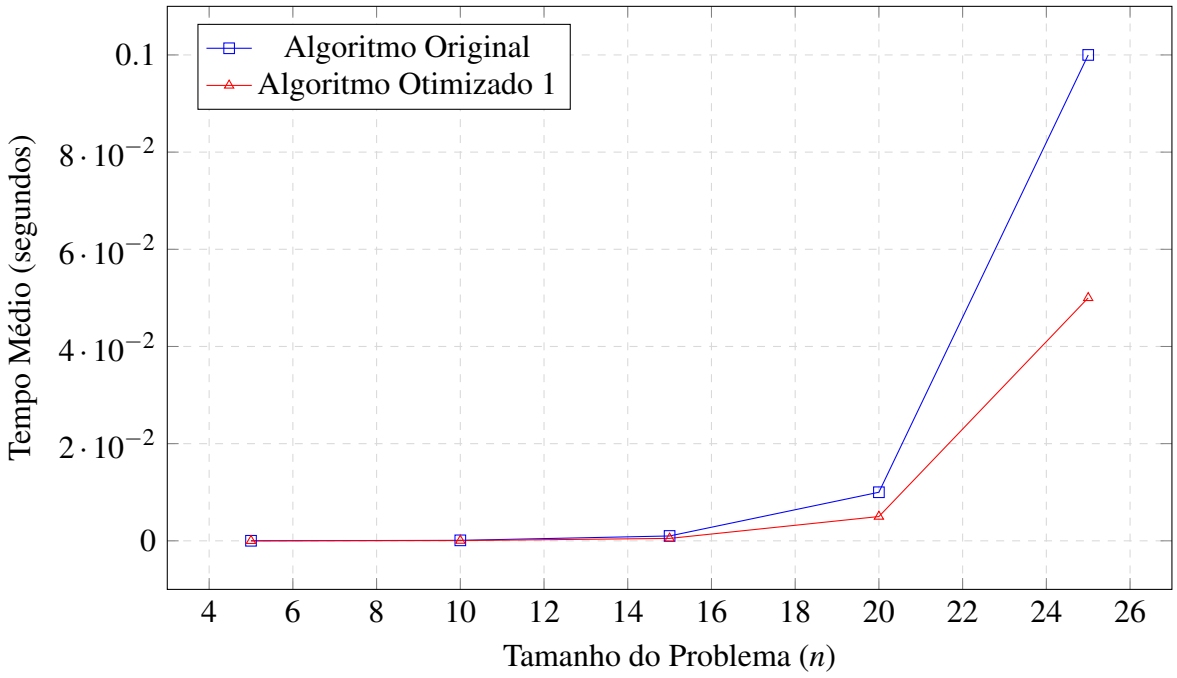

Figura 1: Comparação dos tempos médios de execução entre o algoritmo original e o otimizado.

A curva em azul representa o algoritmo original, enquanto a curva em vermelho corresponde ao Algoritmo Otimizado 1. Note que os dados apresentados são hipotéticos e podem não refletir os resultados reais. Gere gráficos semelhantes para os itens 5.2 e 5.3, comparando os tempos médios de execução das respectivas versões otimizadas com o algoritmo original.