INF 213 - Roteiro da Aula Pratica 1

Arquivos fonte e diagramas utilizados nesta aula:
https://drive.google.com/file/d/1127LdexMcuTr1H0Qc5zJ8WanNwrmLyaa/view?usp=sharing

Como nesta prática há algumas perguntas, faça uma cópia deste documento e acrescente as respostas nela.

**Etapa 1**

Considere o programa “complexidade1.cpp”. Ele possui 5 funções, sendo que cada uma delas realiza uma quantidade diferente de somas.

Estude brevemente o código, compile-o e, a seguir, faça as atividades abaixo (o programa deve ser executado usando a sintaxe “./a.out N”, onde N é o “tamanho da entrada”) :

a) Meça o tempo de execução para diferentes tamanhos de entrada: 1 , 2, 3 , 4, 10, 11, 12, 13, 14, 50, 100, 500, 1000 (apenas entenda e observe os tempos -- não é preciso anotá-los aqui)

b) Meça os tempos para n=1000 e adicione-os à tabela abaixo. A seguir, tente ADIVINHAR o tempo para n=2000 (adivinhe antes de testar!!!). Finalmente, acrescente na última coluna o tempo (medido pelo programa para n = 2000). (não se preocupe -- você não perderá pontos se errar agora….)

| N | 1000 | 2000 (adivinhada) | 2000 |
|---|---|---|---|
| Funcao0 | 0.0159570 | 0.0159570 | 0.0167530 |
| Funcao que executa n somas | 0.0029920 | 0.0119 | 0.0049870 |
| Funcao que executa n^2 somas | 2.5134140 | 10.053 | 9.8806290 |
| Funcao que executa 2n^2 somas | 2.5182020 | 10.072 | 9.5320850 |
| Funcao que executa 4n^2 + n somas | 2.3259640 | 9.303 | 8.3376210 |

c) O que podemos concluir sobre o tempo nas diferentes funções para valores pequenos de n ? (1, 2, 3, 10, 11, …)

Como os números são muitos pequenos, o tempo é tão rápido que chega ser irrelevante.

d) O que podemos concluir sobre as diferenças de tempo para valores maiores ? (na verdade esse experimento simples não é suficiente para garantir que essa conclusão é correta -- mas ela é! Com o tempo veremos que isso é algo que realmente ocorre na prática)

Para valores maiores, o tempo necessário para poder executar o programa cresce muito rapidamente , por isso é importante procurar modificar o código para deixá-lo da maneira mais eficiente possível.

e) Na função 1, por que a soma do ct é mais "importante" (crítica para analisar o algoritmo) do que o "i++"?

Porque o incremento i++ é um inteiro pequeno e que cresce de maneira linear em comparação a soma do ct, além de que a soma ct está na parte mais interna do laço.

**Etapa 2**

Considere o programa "complexidade2.cpp". Ele possui 4 funções, sendo que cada uma delas realiza uma quantidade diferente de operações.

a) Qual a operação básica em cada um delas?

| | |
|---|---|
| Funcao 1 | ct +=i |
| Funcao2 | ct += i+j |
| Funcao3 | ct+=i+j |
| Funcao4 | ct += i (dentro da funcao1) |

b) Quantas vezes a operação básica é realizada em cada um ? (no caso da funcao4, pode ser uma resposta aproximada)

| | |
|---|---|
| Funcao1 | n |
| Funcao2 | n^2 |

| | |
|---|---|
| Funcao3 | n*n+ n*n^2 = n^2 + n^3 = n^3 |
| Funcao4 | ~ 10n + n*(n-1) + n*(n-½) ~ 10n + n^2 - n + n^2 - n/2 ~ n^2 |

c) Meça o tempo de execução de cada função para os diferentes valores de N abaixo (coloque-os na tabela):

| N | 10 | 20 | 50 | 100 | 200 |
|---|---|---|---|---|---|
| funcao1 | 0 | 0 | 0 | 0 | 0.0009980 |
| funcao2 | 0 | 0 | 0.0156600 | 0.0156310 | 0.0910770 |
| funcao3 | 0 | 0 | 0.0811280 | 0.7362650 | 6.3436460 |
| funcao4 | 0.0312860 | 0.0469010 | 0.1337460 | 0.2728300 | 0.5614160 |

d) O que podemos concluir sobre os tempos ?

Para valores pequenos, o tempo é irrelevante pois a diferença entre eles é muito pequeno. A medida que os valores crescem numa escala maior o tempo vai aumentando bruscamente.

## Etapa 3

Compile o programa anterior utilizando a flag “-O3” do g++ (g++ -O3 complexidade2.cpp). Meça novamente os tempos para N = 200.

Essa flag ativa o nível máximo de otimização do compilador. Ela normalmente não reduz a complexidade dos algoritmos, mas acelera de forma considerável o tempo de processamento ao realizar diversos tipos de otimizações (nesta etapa não há nada a ser entregue/respondido)

## Etapa 4

Considere o programa complexidade3.cpp . Veja o código-fonte e entenda o que ele faz.

A seguir, compile o programa e teste-o com o arquivo de teste entrada_5.txt (./a.out < entrada_5.txt > saida.txt). Para ver a saída, basta digitar “cat saida.txt” (no Linux)

Concentre-se apenas na função “encontraPosicoes” (nesta prática NÃO altere nada em outras partes do programa -- especialmente a parte que executa sua função 1 milhão de vezes).

**Código original**

Qual a operação básica da função encontraPosicoes? A comparação (numeros[j] == i)
Quantas vezes essa operação é executada para uma entrada de tamanho N? n^2
Teste o desempenho do programa considerando os arquivos de teste disponibilizados (os números representam o tamanho da entrada)

**Primeira melhoria**

Note que, quando encontramos a posição de um determinado número “i”, não precisamos continuar procurando por esse número no array (podemos passar para o próximo número). Modifique seu programa utilizando essa estratégia para melhorar sua performance.

Considerando a nova versão do programa:
Quantas vezes a operação básica é executada para uma entrada de tamanho N?No melhor dos casos, ou seja, no caso em que cada número se encontre ao mesmo número da posição correspondida, é executada n vezes e no pior dos casos, ou seja, que os números estejam todos embaralhados continua sendo n^2 vezes. E no caso intermediário, n * n/2 vezes.

Teste o desempenho do programa considerando os arquivos de teste disponibilizados (os números representam o tamanho da entrada) Como esse tempo se compara com o obtido na versão original?

O tempo fica notavelmente menor com a melhoria em comparação com o tempo utilizado para rodar o código da versão original, é possível observar isso na tabela a seguir:

| arquivo | Tempo código original | Tempo código c/ primeira melhoria |
|---|---|---|
| entrada_5.txt | 0.0484 | 0.0279630 |
| entrada_10.txt | 0.1686 | 0.0970950 |
| entrada_40.txt | 3.7127 | 2.0055770 |
| entrada_50.txt | 6.3026680 | 3.2008880 |
| entrada_100.txt | 24.2528 | 11.8204970 |
| entrada_200.txt | 92.0963470 | 49.1055660 |
| entrada_300.txt | 220.3384430 | 111.5010640 |

**Segunda melhoria**

Dada uma entrada pequena (por exemplo, os números 2,0,1,3,4,5) encontre a saída (para o problema tratado neste exercício) utilizando uma folha de papel. Pense em uma forma mais

eficiente de resolver o problema e modifique “encontraPosicoes” para funcionar utilizando essa nova estratégia. Sua nova função deverá ficar MUITO mais eficiente.

Considerando a nova versão do programa:
Quantas vezes a operação básica é executada para uma entrada de tamanho N? N vezes.

Teste o desempenho do programa considerando os arquivos de teste disponibilizados (os números representam o tamanho da entrada). Como esse tempo se compara com o obtido na versão original?
Com a segunda melhoria o tempo diminui absurdamente comparado ao código da versão original e até mesmo o da segunda melhoria. É possível observar isso na tabela abaixo:

| arquivo | Tempo código original | Tempo código c/ primeira melhoria | Tempo código c/ segunda melhoria |
|---|---|---|---|
| entrada_5.txt | 0.0484 | 0.0279630 | 0.0149700 |
| entrada_10.txt | 0.1686 | 0.0970950 | 0.0387970 |
| entrada_40.txt | 3.7127 | 2.0055770 | 0.1117000 |
| entrada_50.txt | 6.3026680 | 3.2008880 | 0.1089110 |
| entrada_100.txt | 24.2528 | 11.8204970 | 0.2473360 |
| entrada_200.txt | 92.0963470 | 49.1055660 | 0.4654930 |
| entrada_300.txt | 220.3384430 | 111.5010640 | 0.6969040 |

**Submissao da aula pratica:**
A solucao deve ser submetida ate as 18 horas da proxima Segunda-Feira utilizando o sistema submitty (submitty.dpi.ufv.br).

Deverão ser enviados (não envie mais arquivos .cpp):
1) Um PDF deste documento (contendo suas respostas para as perguntas), com nome roteiro.pdf
2) A versão final do programa complexidade3.cpp (ou seja, a versão com a segunda melhoria).