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	0.0415760	0.0050314	0.0390030
Funcao que executa n somas	0.0032710	0.0065101	0.0061230
Funcao que executa n^2 somas	2.3662020	8.5412789	9.6590820
Funcao que executa 2n^2 somas	2.3395120	8.6472145	9.5941780
Funcao que executa 4n^2 + n somas	2.1563470	8.3214785	8.6326480

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

Pode-se dizer que os tempos de execução das funções entre cada intervalo de valor para N, aumenta insignificantemente ao passo do aumento de N.

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)

Pode-se dizer que ao tomarmos valores maiores para N, os intervalos de tempo para cada função aumenta arbitrariamente à medida que N aumenta de valor.

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

Pode-se dizer que a soma do ct é a operação mais importante pois ela contribui maior tempo de processamento para a função.

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?

funcao1	Incremento de ct (ct+=i)
funcao2	Incremento de ct (ct += i + j)
funcao3	Incremento de ct (ct += i +j)
funcao4	Incremento de ct (ct += i*sin(j)*cos(i) +j+100*i-547+10/(i+1.0+sqrt(j));)

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

funcao1	n
funcao2	$2N^2$
funcao3	$2N^2 + n$
funcao4	$11n^2$

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	0.0000680	0.0001770	0.0004940	0.0011650	0.0021750
funcao2	0.0006900	0.0031950	0.0201850	0.0436990	0.1167590
funcao3	0.0025770	0.0197680	0.1013720	0.7548920	6.2967170
funcao4	0.0245300	0.0227320	0.0460520	0.1113650	0.2934930

d) O que podemos concluir sobre os tempos ?

Pode-se dizer que há um aumento significante dos tempos ao passo que o valor de n também aumenta. Detalhe importante para a função3 que aumentou quase que exponencialmente os valores de tempo entre os intervalos.

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?
R: if(numeros[j]==i)

Quantas vezes essa operação é executada para uma entrada de tamanho N?
R: $2n^2$
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.

R: Break;

Considerando a nova versão do programa:
Quantas vezes a operação básica é executada para uma entrada de tamanho N?
R: n^2

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?

R: Pode-se dizer que houve uma diminuição em relação ao tempo original, por mais que irrelevante.

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?
R: N

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?

R: Pode-se dizer que após a segunda melhoria, o tempo de processamento da função caiu consideravelmente.

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).