INF 213 - Roteiro da Aula Prática 1

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). Não meça (agora) para valores maiores que 1000.

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!!!) com base nos resultados para n=1000. 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 (medida)	2000 (adivinhada)	2000 (medida)
Funcao0	0.0185990	0.0185990	0.0188260
Funcao que executa n somas	0.0029030	0.005806	0.0058210
Funcao que executa n^2 somas	2.3322160	9.328864	10.2657000
Funcao que executa 2n^2 somas	2.4471720	9.788688	10.2298100
Funcao que executa 4n^2 + n somas	2.3534460	9.413784	8.8703050

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

Uma vez que são utilizados números pequenos nas respectivas funções, o valor resultante do tempo não é claro, obtendo um tempo semelhante em todos os testes das funções, impedindo de concluir uma solução.

d) O que podemos concluir sobre as diferenças de tempo para valores maiores de N? (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)

Quanto maior os valores nos testes, mais claro é o tempo obtido nos resultados, uma vez que a variável de maior potência, aumenta o tempo de execução em números maiores, como N >= 1000.

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

A operação ct é a mais importante, uma vez que , ela é a operação básica, logo a mais executada do algoritmo, sendo assim ela pesa muito mais (será maior ou igual) em relação ao tempo de execução do algoritmo, do que “i++”.

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?

Função 1	ct += 1 (linha 12)
Função 2	ct += i + j (linha 21)
Função 3	ct += i + j (linha 21)
Função 4	ct += 1 (linha 12)

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

Função 1	N vezes
Função 2	(N * N) vezes
Função 3	(N * N * N) vezes
Função 4	(N * (N - 1) / 2) vezes

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	0.0001410	0.0001190	0.0008860	0.0018030
funcao2	0.0006920	0.0018760	0.0055750	0.0277820	0.0975140
funcao3	0.0008490	0.0062700	0.0890710	0.7322850	6.0949270

funcao4	0.0080250	0.0170400	0.0431130	0.1108060	0.2658580

d) O que podemos concluir sobre os tempos ?

Podemos concluir que os tempos de execução do algoritmo não são tão claros com valores pequenos de N, porém com valores maiores de N os tempos de execução de algoritmo são mais claros, outro ponto que podemos observar é que as operações básicas que contém maior expoente de N são as mais demoradas, como no caso da funcao 3, onde (N * N * N) vezes.

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?

```
if(numeros[j]==i)
```

Quantas vezes essa operação é executada para uma entrada de tamanho N?

N * N vezes

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?

(N * (N - 1) / 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 em relação ao programa original é consideravelmente mais eficiente, uma vez que o programa para de checar os outros valores posteriores do vetor após encontrar o de interesse.

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 (para qualquer tamanho de entrada, principalmente as maiores).

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), principalmente os maiores. Como esse tempo se compara com o obtido na versão original?

O tempo do programa se torna extremamente mais eficiente em relação ao programa original, uma vez que a operação básica deixa de ser executada N * N vezes, e passa a ser executada em N vezes, o que torna um software extremamente mais rápido em seu tempo de execução.

Submissao da aula pratica:

A solucao deve ser submetida ate as 18 horas da proxima Terça-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).