Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital

Estruturas de Dados Básicas I • IMD0029
– Análise Empírica de Algoritmos de Ordenação –
2 de outubro de 2017

Sumário

1 Introdução

Neste trabalho você deverá atuar como um consultor técnico contratado para ajudar na escolha do algoritmo ideal para realizar ordenação em um arranjo. Para cumprir seu objetivo você deverá utilizar análise empírica de algumas opções de algoritmos sugeridos tentando simular determinados cenários. Os resultados da análise deverão ser apresentados na forma de um relatório técnico com suas recomendações para os cenários avaliados em seus experimentos.

A seguir serão apresentados os cenários a serem simulado e os algoritmos para os quais você deverá realizar a análise empírica. Também serão listadas orientações para auxiliar na elaboração do relatório técnico a ser redigido.

Ao final do exercício você deve submeter, através do Sigaa, tanto o relatório técnico quanto o conjunto de programas desenvolvidos para realizar a análise empírica.

Nas próximas seções serão fornecidos mais detalhes sobre o trabalho, algumas instruções para a elaboração do relatório e a política de colaboração na execução do trabalho em equipe.

2 O Problema Computacional

O problema da **ordenação de um arranjo sequencial** tem como entrada:

> Uma *sequência*, $\langle a_1, \ldots, a_n \rangle$, de n elementos, com $n \in \mathbb{Z}$ e $n > 0$, tal que os elementos da sequência pertencem a um conjunto totalmente ordenável por '$\leq$'.

e como saída:

> Uma *permutação*, $\langle a_{\pi 1}, \ldots, a_{\pi n} \rangle$, da sequência de entrada tal que $a_{\pi 1} \leq a_{\pi 2} \leq \cdots \leq a_{\pi n}$.

Para solucionar este problema você deverá analisar oito algoritmos, com complexidade temporal variadas. Em suas simulações você deve considerar alguns cenários, descritos na próxima seção.

3 Cenários da Simulação

Você deve considerar que a simulação será realizada sobre um arranjo de inteiros (`unsigned int`) cujo comprimento, ou **tamanho de amostra** n, deverá ser "*muito grande*". Entenda o termo "*muito grande*" como sendo o limite da sua máquina, ou seja, o maior arranjo de inteiros longos que sua máquina consegue alocar de maneira dinâmica no segmento *heap* de memória. Portanto, recomenda-se que você realize experimentos preliminares com o objetivo de determinar qual é o limite L de tamanho para o arranjo que sua máquina suporta.

Como estamos interessados em avaliar o **comportamento assintótico** dos algoritmos em relação ao seu *tempo de execução*, os cenários deverão simular a execução dos algoritmos para diversos tamanhos de amostras com valores de n crescentes, até atingir o limite L encontrado. Por esta razão serão necessário pelo menos 25 tamanhos de amostras diferentes.

Com relação a *organização das amostras* é necessário simular três situações: arranjos com elementos em ordem *não decrescente*, arranjos com elemento em ordem *não crescente* e arranjos com elementos *aletórios*. Note, portanto, que é possível a ocorrência de elementos repetidos no arranjo.

4 Algoritmos

Você deve implementar e analisar oito algoritmos em seus experimentos. São eles: insertion sort, selection sort, bubble sort, shell sort, quick sort, merge sort e radix sort (LSD).

5 Recomendações e Dicas de Implementação

Para facilitar a construção do programa de testes para medição dos tempos de execução, recomenda-se que os oito algoritmos sejam armazenados em um vetor de ponteiros para função. Portanto, todos os oito algoritmos devem apresentar a **mesma assinatura**.

A segunda recomendação é fazer com que o programa seja flexível a ponto de receber, via argumentos de linha de comando, o número limite de amostras que se deseja testar. Outra

maneira de tornar o programa mais versátil é suportar uma forma eficiente de se definir quais algoritmos desejamos testar ao executar o programa. Esta estratégia permite a execução dos testes para algoritmos individuais ou grupos de algoritmos, conforme a necessidade.

É importante escolher qual escala de crescimento será utilizada para os diversos tamanhos n de amostras. Pode-se optar por uma escala linear de crescimento, ou mesmo uma *escala logarítmica*. Exemplos de escala logarítmica são 2^i, e^i e 10^i. Uma possibilidade de implementação seria $n = 2^i$, com $i \in [5; 30]$. Resta determinar, contudo, se a máquina onde os experimentos serão realizados consegue alocar um arranjo com $2^{30} = 1073741824$ elementos.

Se você for adotar o `gnuplot` como ferramenta de geração de gráficos, lembre-se de gravar os resultados na forma de colunas ao invés de linhas. Veja abaixo um exemplo de arquivo de dados com tempos de execução (em milissegundos) que podem ser lidos pelo `gnuplot`.

```
# N          ILS          IBS          RBS          CBS
32         0.0006781    0.0001379    0.000142     0.0003896
64         0.0001948    0.000127     0.0001426    7.52e-05
128        0.0003446    0.0001447    0.0001631    7.66e-05
256        0.0006513    0.0001596    0.0001681    7.92e-05
512        0.0012158    0.0001657    0.0001874    8.2e-05
1024       0.0028704    0.0001814    0.000207     8.63e-05
2048       0.0053431    0.0001986    0.0002213    9.17e-05
4096       0.0107326    0.000213     0.0002458    9.53e-05
8192       0.0214113    0.0002275    0.0002627    0.0001014
16384      0.0430576    0.0002437    0.0002788    0.0001078
32768      0.0889439    0.000259     0.0002954    0.000122
65536      0.177514     0.000276     0.0003172    0.000125
131072     0.439085     0.0005136    0.0005841    0.0002343
262144     1.07981      0.0003404    0.0003984    0.0001925
524288     1.57872      0.0004949    0.000418     0.0002355
1048576    3.36122      0.0006097    0.0004471    0.0002588
2097152    6.94192      0.0007354    0.0006473    0.0004338
4194304    13.232       0.0006152    0.000541     0.0003232
8388608    26.6012      0.0014357    0.000486     0.0002969
16777216   56.5531      0.0008658    0.0006288    0.0003637
33554432   115.472      0.0006561    0.0005413    0.0003267
67108864   210.225      0.0006529    0.0005563    0.0003993
134217728  455.382      0.0006664    0.0005392    0.0003586
268435456  901.417      0.000766     0.001398     0.0003791
536870912  1988.72      0.0016476    0.0018264    0.0015694
1073741824 14014.1      0.0080101    0.0005991    0.00701
```

Para cada algoritmo armazene os tempos de, por exemplo, 50 execuções de cada instância individual com tamanho n. Depois calcule a média aritmética dos tempos das 50 execuções: este será o tempo médio da execução do algoritmo para uma instância do problema com tamanho n.

Para evitar erros de arredondamento no cálculo da média aritmética das 50 tomadas de tempos, recomenda-se a utilização da média progressiva, ou seja, a média vai sendo atualizada a cada novo tempo computado. Para $k = 1, 2 \ldots, m$ execuções, temos

$$\begin{aligned} M_0 &= 0, && \text{valor inicial da média} \\ M_k &= M_{k-1} + \tfrac{x_k - M_{k-1}}{k}, && \text{atualização progressiva da média} \end{aligned}$$

onde x_k é tempo mensurado para o k-ésima execução e $M_{k=m}$ corresponde a média aritmética final da sequência de m tempos medidos. Esta fórmula evita que seja necessário somar todos os tempos primeiro (o que pode provocar erro de arrendondamento) para depois dividir a soma total por m, ou seja, não é recomendado a fórmula trivial $M = \frac{\sum_{k=1}^{m} x_k}{m}$.

Com relação a alocação do arranjo, ao invés de alocar/desalocar vários arranjos com tamanhos crescentes de n, recomenda-se que um único arranjo com tamanho máximo de n (digamos $n = 2^{30}$ elementos) seja alocado e preenchido no início do programa. Na hora de executar os algoritmos, você deve passar apenas a parte do arranjo correspondente ao tamanho da amostra da vez. Por exemplo, se você precisa testar uma amostra $n = 1024$ elementos, invoque a busca passando $l = 0$ e $r = 1023$, embora o arranjo possua $n = 2^{30}$ elementos no total. Neste caso a busca sera feita em uma subconjunto do arranjo original.

Esta simples estratégia evita que seu programa perca muito tempo alocando e desalocando arranjos de tamanhos menores. Se você for atencioso, vai perceber que perde-se mais tempo alocando o arranjo com tamanho máximo do que realizando a busca! Vale ressaltar que o tempo necessário para alocar o vetor não deve entrar no cálculo do tempo de execução dos algoritmos.

Após coletar e calcular a média dos tempos de execução para todos os tamanhos de instância n, gere um gráfico mostrando uma curva de crescimento (n no eixo X e tempo de execução no eixo Y) para cada algoritmo. Faça o mesmo tipo de comparação considerando o *número de passos executados da operação dominante*.

6 Relatório Técnico

Projete e implemente, os algoritmos listados na Seção 4. A seguir, realize testes empíricos comparativos entre seus desempenhos para pelo menos 25 entradas de tamanhos diferentes, cobrindo os cenários descritos na Seção 3.

Em seguida, elabore um relatório técnico com:

1. Uma **introdução**, explicando o propósito do relatório;
2. Uma seção descrevendo o **método** seguido, ou seja quais foram os materiais e a metodologia utilizados. São exemplos de materiais a caracterização técnica do computador utilizado (i.e. o processador, memória, placa mãe, etc.), a linguagem de programação adotada, tipo e versão do sistema operacional, tipo e versão do compilador empregado, a lista de algoritmos implementado (com código), os cenários considerados, dentre outros. São exemplos de metodologia uma descrição do método ou procedimento empregado para gerar os dados do experimento, quais e como as medições foram tomadas para comparar os algoritmo (tempos, passos, memória), etc.
3. Os **resultados** alcançados (gráficos e tabelas). Gere um gráfico para cada algoritmo testado. Gerar gráficos comparativos de algoritmos na mesma categoria de complexidade. Gerar gráficos comparando algoritmos nos cenários solicitados.
4. A **discussão** dos resultados, no qual você deve procurar responder questões como:
   (a) O que você descobriu de maneira geral?
   (b) Quais algoritmos são recomendados para quais cenários?

(c) Como o algoritmo de decomposição de chave (radix) se compara com os algoritmos de comparação de chaves?

(d) É verdade que o quick sort, na prática, é mesmo mais rápido que o merge sort?

(e) Aconteceu algo inesperado nas medições? (por exemplo, picos ou vales nos gráficos) Se sim, por que?

(f) Que função matemática melhor se aproxima para descrever o gráfico gerado? É possível estimar o tempo necessário para cada algoritmo executar com 100 milhões de elementos?

(g) A análise empírica é compatível com a análise matemática? Outras observações que você julgar interessante e pertinente ao trabalho, além destas, também podem ser acrescentadas à discussão. Uma boa discussão dependente da sua criatividade e/ou habilidade em contar uma "história" sobre o que aconteceu nos experimentos.

Um relatório técnico de algum valor acadêmico deve ser escrito de tal maneira que possibilite que uma outra pessoa que tenha lido o relatório consiga reproduzir o mesmo experimento. Este é o princípio científico da **reprodutibilidade**.

Note que para realizar uma comparação correta entre algoritmos, os *mesmos* valores gerados para cada tamanho n devem ser fornecidos como entrada para os algoritmos que estão sendo comparados.

Para a medição de tempo, recomenda-se a utilização da biblioteca `std::chrono` do C++11, cuja descrição pode ser encontrada aqui.

7 Autoria e Política de Colaboração

O trabalho pode ser realizado **individualmente** ou em **duplas**, sendo que no último caso é importante, dentro do possível, dividir as tarefas igualmente entre os componentes.

Qualquer equipe pode ser convocada para uma entrevista. O objetivo da entrevista é duplo: confirmar a autoria do trabalho e determinar a contribuição real de cada componente em relação ao trabalho. Durante a entrevista os membros da equipe devem ser capazes de explicar, com desenvoltura, qualquer trecho do trabalho, mesmo que o código tenha sido desenvolvido pelo outro membro da equipe. Portanto, é possível que, após a entrevista, ocorra redução da nota geral do trabalho ou ajustes nas notas individuais, de maneira a refletir a verdadeira contribuição de cada membro, conforme determinado na entrevista.

O trabalho em cooperação entre alunos da turma é estimulado. É aceitável a discussão de ideias e estratégias. Note, contudo, que esta interação **não** deve ser entendida como permissão para utilização de código ou parte de código de outras equipes, o que pode caracterizar a situação de plágio. Em resumo, tenha o cuidado de escrever seus próprios programas.

Trabalhos plagiados receberão nota **zero** automaticamente, independente de quem seja o verdadeiro autor dos trabalhos infratores. Fazer uso de qualquer assistência sem reconhecer os créditos apropriados é considerado **plagiarismo**. Quando submeter seu trabalho, forneça a

citação e reconhecimentos necessários. Isso pode ser feito pontualmente nos comentários no início do código, ou, de maneira mais abrangente, no arquivo texto `README`. Além disso, no caso de receber assistência, certifique-se de que ela lhe é dada de maneira genérica, ou seja, de forma que não envolva alguém tendo que escrever código por você.

8 Entrega

Você deve submeter um único arquivo com a compactação da pasta do seu projeto, incluindo o relatório técnico no formato PDF. O arquivo compactado deve ser enviado **apenas** através da opção Tarefas da turma Virtual do Sigaa, em data divulgada no sistema.

$\sim$ FIM $\sim$