Trabalo 2 – Ordenação

Técnicas de Programação Avançada — Bacharelado em Sistemas de Informação

Prof. Dr. Jefferson O. Andrade

Campus Serra — Instituto Federal do Espírito Santo (Ifes)

2019/2

Sumário

1 Introdução

A ordenação é um dos problemas algorítmicos mais fundamentais da ciência da computação. Já foi afirmado que até 25% de todos os ciclos de CPU são usados para ordenação, o que fornece um grande incentivo para estudos adicionais e otimização sobre ordenação. Além disso, há muitas outras tarefas (pesquisar, calcular a mediana, encontrar a moda, remover duplicatas etc.) que podem ser implementadas com muito mais eficiência depois que os dados são ordenados. O estudo da ordenação é particularmente interessante, visto que há muitas abordagens diferentes (dividir para conquistar, particionamento, *pigeonhole*, etc.) que conduzem a algoritmos de ordenação úteis, e essas técnicas podem muitas vezes ser reutilizadas para outras tarefas algorítmicos. A grande variedade de algoritmos nos dá muita riqueza para explorar, especialmente quando consideramos os *tradeoffs* oferecidos em termos de eficiência, mix de operações, complexidade de código, melhor/pior caso de entrada, e assim por diante. Este trabalho foca na implementação e análise de alguns dos algoritmos de ordenação.

Tabela 1: Algorimos de ordenação selecionados para o trabalho.

Algoritmo	**Identificador**
Selection sort	`selectsort`
Insertion sort	`insertsort`
Merge Sort	`mergesort`
Quicksort	`quicksort`
Heapsort	`heapsort`
Introsort	`introsort`
Timsort	`timsort`
Smoothsort	`smoothsort`
Patience sorting	`patience`

2 Enunciado

Este trabalho consiste de duas etapas:

1. Implementação de um conjunto de algoritmos de ordenação.
2. Análise do desempenho dos algoritmos implementados.

A descrição destas duas etapas será vista com mais detalhes abaixo.

2.1 Implementação

Esta etapa consiste na implementação de um subconjunto dos algoritmos indicados na Tabela 1. Os primeiros cinco algoritmos – i.e., *selection sort*, *insertion sort*, *merge sorg*, *quicksort* e *heapsort* – são sobrigatório. Dos quatro restantes, devem ser escolhidos dois.

Estes algoritmos deverão ser implementados em uma linguagem de programação à escolha do grupo. Todo o trabalho deverá ser implementado em uma mesma linguagem.

Os algoritmos de ordenação devem ser implementados de tal modo que eles funcionem para ordenação de **registros** em um vetor de acordo com uma **chave de ordenação**. Uma forma relativamente simples de se conseguir esse efeito é pasar uma função/objeto de comparação como argumento para o procedimento de ordenação, e utilizar esta função/objeto para realizar as comparações, ao invés de usar os operadores relacionais diretamente. Esta abordagem é seguida, por exemplo, pela função `qsort`, da linguagem C[1], e pelo método `sort`, da classe `Collections` em Java[2].

O tempo de execução absoluto não será importante, portanto, em princípio, mesmo uma linguagem "lenta" com Python pode ser utilizada sem prejuízo do trabalho. A única imposição com relação à linguagem utilizada é que o ambiente de desenvolvimento da linguagem (compilação, execução, etc.) deve ser livre, gratuito e disponível para instalação no sistema Operacional GNU Linux nas distribuição Ubuntu através do comando `apt`.

Os dados dos registros serão lidos de um arquivo texto em formato CSV (*Comma-separated Values*)[3]. Depois de lidos do arquivo de entrada e ordenados, os dados ordenados

[1] `qsort(void *ptr, size_t count, size_t size, int (*comp)(void *, void *))`
[2] `sort(List<T> list, Comparator<? super T> c)`
[3] `https://en.wikipedia.org/wiki/Comma-separated_values`

devem ser gravados em um arquivo de saída, também no formato CSV. A leitura e a gravação dos arquivos CSV não é o objeto de avaliação deste trabalho, portanto, o aluno pode utilizar alguma biblioteca já pronta para estas atividades, ou utilizar código já desenvolvido por terceiros, como o código disponível no sítio web Rosetta Code[4], por exemplo.

Conforme citado na Subseção 2.1 os algoritmos devem receber uma função ou objeto que faça a comparação. Para o teste do trabalho e a coleta dos tempos de execução a ordenação deve ser feita sempre pelo campo *Identificador de Usuário* (uid).

Algorithm 1 Template para programa de ordenação.

```
procedure MAIN(args)
    inputName ← PARSEINPUTFILENAME(args)
    outputName ← PARSEOUTPUTFILENAME(args)
    A ← READCSV(inputName)
    initTime ← GETCPUTIME
    XYZSORT(A)
    finishTime ← GETCPUTIME
    WRITECSV(A, outputName)
    REPORTTIME(A, initTime, finishTime)
end procedure
```

O Algoritmo 1 apresenta um modelo geral do que se espera do programa gerado para execução dos algorimtos de ordenação. Note que o tempo é medido apenas em relação à execução do algoritmo de ordenação. O último passo, i.e., relatar o tempo de execução do algoritmo, deve produzir uma única linha de saída (para cada execução do algoritmos de ordenação) contendo as seguintes informações, em ordem:

1. O identificador do algoritmo (veja Tabela 1).
2. A quantidade de registros ordenados.
3. O tempo de execução do algoritmo de ordenação em milisegundos.

Um exemplo de uso do programa é mostrado abaixo:

```
$$ ./isort -i data100.csv -o sorted100.csv
insertsort      100             0
```

No exemplo acima, a linha que começa com os caracteres sifrão ($$) contém o comando executado no terminal. Note que o nome `isort` é meramente uma sugestão. O nome do arquivo de entrada foi indicado pela opção `-i` e o nome do arquivo de saída foi indicado pela opção `-o`. Após a execução do programa, foi gerada a saída contendo o identificador do algoritmo (`insertsort`), o número de registros ordenados (100), e o tempo gasto para ordenação em milisegundos (0).

O modelo apresentado no Algoritmo 1 supõe que será criado um programa para executar cada algoritmo de ordenação. Também é possível criar apenas um programa principal que permita selecionar o algoritmo que será usado para ordenação.

[4] `https://www.rosettacode.org/wiki/CSV_data_manipulation`

Algorithm 2 Template para programa de ordenação com seleção do algoritmo.

```
procedure MAIN(args)
    sort ← SELECTALGORITHM(args)
    inputName ← PARSEINPUTFILENAME(args)
    outputName ← PARSEOUTPUTFILENAME(args)
    A ← READCSV(inputName)
    initTime ← GETCPUTIME
    SORT(A)
    finishTime ← GETCPUTIME
    WRITECSV(A, outputName)
    REPORTTIME(A, initTime, finishTime)
end procedure
```

O Algoritmo 2 apresenta uma versão modificada do modelo de programa principal. Nesta versão, a função que fará a ordenação é selecionada de acordo com o que for indicado nos argumentos de linha de comando. A função selecionada é depois aplicada ao vetor de entrada. O restante do modelo é idêntico à versão anterior. O exemplo abaixo mostra um exemplo/sugestão de uso do programa principal implementado de acordo com o modelo do Algoritmo 2.

```
$$ ./mysort --algorithm=sel -i data500.csv -o sorted500.csv
selectsort      500             13
```

2.2 Análise

O processo de análise dos algoritmos implementados será feito por experimentação. Serão fornecidos alguns arquivos de dados para que os algoritmos sejam testados e sejam coletados os dados dos tempos de execução. Esses arquivos de dados possuem números crescentes de registros: 10, 25, 50, 75, 100, 250, 500, 750, 1000, 2500, 5000, 7500, 10 000, 25 000, 50 000, 75 000, 100 000, 250 000, 500 000, 750 000, 1 000 000, 2 500 000, 5 000 000 e 7 500 000.

O processo de análise consitirá em executar cada um dos algorimtos implementados para cada um dos arquivos de entrada e registrar os tempos de execução. Deve-se executar cada algoritmo múltiplas vezes e registar os diversos tempos – todos os algoritmos devem ser executados a mesma quantidade de vezes. Ao fazer as análises, use a média dos tempos registrados.

Em seguida, para cada algoritmo, deve-se desenhar o gráfico correspondente aos dados coletados e analisar se os dados coletados estão de acordo com a **complexidade esperada** do algoritmo. Por fim, deve-se fazer uma análise conjunta de todos os algoritmos de ordenação e determinar para que faixa de tamanho da entrada cada um dos algoritmos é mais eficiente.

É provável que os algoritmos de ordenação menos eficientes levem um tempo muito grande para realizar a ordenação. Não é necessário esperar que os algoritmos menos eficientes terminem para todos os arquivos de dados. Você pode estipular um tempo máximo de espera e se este tempo for ultrapassado o processo pode ser abortado. Mas tome cuidado para não definir um tempo muito curto, de modo a inviabilizar a análise. Uma sugestão é executar os algoritmos mais eficientes primeiro e registrar o tempo para

o maior arquivo de dados e usar este tempo como parâmetro. Deste modo, se o tempo para ordenar o maior arquivo foi t_1 para algum dos algoritmos mais eficientes, você pode definir o *timeout* como $10t_1$ ou $20t_1$, por exemplo. A informação de como foi calculado o tempo de *timeout* e de qual foi exatamente este tempo deve constar do relatório.

Todos os valores coletados devem ser apresentados tanto na forma de tabelas quanto na forma de gráficos para facilitar a análise dos dados. Uma recomendação que pode economizar tempo na confecção dos relatórios é utilizar o comando `\csvreader` do pacote `csvsimple` do LaTeX. Os gráficos podem ser gerados utilizando-se qualquer pacote de software. Uma sugestão é utilizar a ferramenta `gnuplot`.

2.3 Arquivos de Dados

Os arquivos de dados estarão no formato CSV e serão compostos pelos seguintes campos:

1. Email (`email`); tipo: texto.
2. Sexo (`gender`); tipo: caractere; valores válidos: `M`, `F`, `O`.
3. Identificador de Usuário (`uid`); tipo: alfa-numérico; único.
4. Data de nascimento (`birthdate`); tipo: data; formato: ISO-8601.
5. Altura, em centímetros (`height`); tipo: inteiro.
6. Peso, em kilogramas (`weight`); tipo: inteiro.

Os arquivos estão disponíveis para *download* no diretório compartilhado `https://drive.google.com/drive/folders/1eeuM04049iAlYTOAp_lFCMlMQN9CUr8w?usp=sharing`, no Google Drive.

3 Entrega

O trabalho deve ser executado por grupos de até três (3) alunos (veja a Seção 4 para informações sobre colaboração com colegas), e deve ser entregue até a data definida no AVA.

Deve ser entregue um arquivo ZIP[5] contendo, ao menos, o seguinte:

1. Um arquivo chamado `Readme.md` descrevendo o que está sendo entregue, e como compilar (se for o caso) e executar o(s) programas no sistema GNU/Linux.
2. O código fonte desenvolvido para o projeto – preferencialmente organizado em um diretório próprio.
3. O código fonte em LaTeX do relatório de análise.
4. O relatório de análise em formato PDF.

[5]Outros formatos, como RAR e LHA, serão desconsiderados.

4 Política de Colaboração

Resolver um problema algorítmico é um processo criativo. Quando apresentado a um novo problema, é sua tarefa "desmontá-lo" e alcançar seu próprio entendimento. Este é um processo meticuloso e demorado. Há muito a ser aprendido com o processo de pensar em soluções para os problemas propostos. Obter ajuda de outro lugar destrói esse processo. No entanto, discutir com os outros depois de ter passado algum tempo pensando em como solucionar um problema pode ajudar o processo e trazer à luz outros aspectos do problema.

Você pode discutir os problemas com o professor ou com outros alunos da sua turma, depois de ter pensado o suficiente. Mas quando chegar a hora de escrever sua solução, ela deve ser seu próprio trabalho e deve estar em suas próprias palavras. Se você recebeu ajuda de qualquer outra fonte, é necessário citar sua fonte no local apropriado da lição de casa, ou seja, anote a URL ou o nome da pessoa ou o autor e título do texto do qual sua solução foi adquirida.

Depois de obter ajuda de alguma fonte, tente escrever a solução com suas próprias palavras. Se você discutiu com um colega de classe ou amigo e chegou a uma solução juntos, então vocês dois devem indicar isso em seus trabalhos. Se você estiver ajudando alguém ou fornecendo sua solução para alguém, certifique-se de que ela seja anotada como a fonte da solução. Você também pode indicar que você ajudou essa pessoa com o problema especificado. Se você não escrever onde obteve ajuda, isso seria considerado uma fraude ("cola").

Qualquer evidência de fraude (sem citar a fonte) resultará em severa penalização de todas as partes envolvidas.