Projeto da Disciplina - Algoritmos de Busca e Ordenação

Objetivo

O objetivo deste trabalho é implementar algoritmos de busca sequencial e binária, bem como os algoritmos de ordenação utilizando os métodos simples e os métodos eficientes (sofisticados). Feita a implementação, os algoritmos deverão ser avaliados por meio de uma série de experimentos. Sendo que, os resultados desses experimentos deverão ser relatados.

Experimentos

Os algoritmos deverão ser avaliados utilizando diferentes tamanhos e características de conjuntos de valores. Tais conjuntos devem conter 10, 100, 1.000, 10.000, 100.000, 1.000.000, 10.000.000 e 100.000.000 de elementos. Cada conjunto deve ser gerado de 5 formas diferentes:

1. **Aleatoriamente**: utilizando o método ***random***
2. **Aleatoriamente quase ordenado**: utilizando o método ***nearSort***
3. **Aleatoriamente com muitos valores repetidos**: utilizando o método ***fewUnique***
4. **Ordenado**: utilizando o método ***sort***
5. **Invertido**: utilizando o método ***reversed***

Os métodos acima descritos estão implementados no arquivo *GeraVetor* disponibilizado no *GitHub* do projeto da disciplina[1]. Esses métodos retornam um conjunto de elementos de tamanho especificado. Para evitar problemas com as massas de dados gerados, procure gerá-los inicialmente e salvá-los em arquivos. Assim, o experimento poderá ser replicado e executado inúmeras vezes para obter valores para realizar, por exemplo, uma média de tempo de execução.

Os algoritmos a serem implementados são:

- Métodos Simples
  - *Insertion Sort*
  - *Selection Sort*
  - *Bubble Sort*
  - *Comb Sort*
- Métodos Eficientes (Sofisticados)
  - *Quick Sort*
  - *Merge Sort*
  - *Shell Sort*
  - *Heap Sort*
  - *Radix Sort*
  - *Gnome Sort*
  - *Count(ing) Sort*
  - *Bucket Sort*
  - *Cocktail Sort*
  - *Timsort*

[1] https://github.com/ceplan/jSorting

Os algoritmos acima descritos serão distribuídos em equipes. Tais informações serão detalhadas na Seção **Equipes e Algoritmos**.

Além desses algoritmos, cada equipe também deverá avaliar o algoritmo *Bogosort*. O algoritmo *Bogosort* (também conhecido como *CaseSort* ou *Estou com Sort*), é um algoritmo de ordenação extremamente ineficiente. É baseado na reordenação aleatória dos elementos. Não é utilizado na prática, mas será usado para comparação com algoritmos mais eficientes. Esse algoritmo já está implementado no projeto **jSorting**, o qual é detalhado na seção de mesmo nome.

Além dos algoritmos de ordenação, dois algoritmos de busca serão implementados: sequencial (linear) e binária. Essas buscas serão avaliadas de duas formas: pré-ordenação e pós-ordenação.

De modo a exemplificar a rotina dos experimentos, considere um conjunto em teste *X = {1,4,33,67,5,7,8,30,15,100}* gerado por uma das 5 formas. Para cada forma de geração de conjuntos e para cada tamanho de conjunto, os passos dos experimentos serão:

1. Seleciona-se um elemento aleatoriamente do conjunto em teste *X*, por exemplo, 67.
2. Aplica-se as buscas sequencial e binária no conjunto *X*, ainda sem ordenação, procurando o elemento selecionado.
3. Avalia-se o desempenho das buscas.
4. Guarda-se o estado do conjunto *X* para próximos experimentos, ou seja, realiza-se uma cópia.
5. Para cada algoritmo de ordenação:
    a. Aplica-se um algoritmo de ordenação no conjunto *X.*
    b. Avalia-se o desempenho da ordenação.
    c. Aplica-se as buscas sequencial e binária no conjunto *X*, agora ordenado, procurando o elemento selecionado.
    d. Avalia-se o desempenho das buscas.
    e. Reinicia-se o conjunto *X* para o estado original, ou seja, antes de iniciar o **Passo 5**. Dessa forma, o conjunto *X* agora é o valor de sua cópia realizada no **Passo 4**.
    f. Volta-se ao **Passo a** para aplicar o próximo algoritmo de ordenação, se houver.

Análise

A análise deverá ser realizada comparando os algoritmos em relação a:

- Complexidade (Análise de Algoritmos)
- Desempenho
    - Tempo de execução
    - Movimentações de trocas
    - Comparações realizadas
- Outras métricas que a equipe achar pertinente

Tais análises deverão responder as seguintes questões de pesquisa:

- **QP1**: Qual algoritmo de busca e ordenação são os mais eficientes? Para responder essa questão deverá ser analisado qual o algoritmo é melhor para cada conjunto de dados e qual for o melhor em média.
    - **Sugestão**: Uma forma de se obter essa informação é por meio da análise (inferência) dos dados usando estatística. Não há necessidade de avaliar o tipo de amostra para determinar o melhor método. Métodos comuns disso são:

    - **Kruskal-Wallis**: pode ser utilizado para comparar vários algoritmos em relação a um conjunto de dados. Isso é adequando para verificar qual algoritmo determinar uma faixa de conjunto de dados: ex: *Bubble sort* com dados de 10 a 1.000, *Selection Sort* com dados de 10.000, 100.000 e 1.000.000, e *Merge Sort* com 10.000.000 e 100.000.000 elementos.
    - **Friedman**: pode ser utilizado para comparar os algoritmos em relação a todos os conjuntos.
- **QP2**: Como a forma em que os vetores estão ordenados influencia os algoritmos de busca e ordenação?
- **QP3**: A complexidade dos algoritmos de busca e ordenação influencia nos resultados obtidos?
- **QP4**: Existe alguma tendência de aproximação ou distanciação dos tempos conforme o conjunto de dados aumenta?

Dicas:

- Utilizar matrizes
    - Destaque os melhores valores em negrito
- Utilizar gráficos
    - Pode-se utilizar gráfico de linha, porém recomenda-se o gráfico de caixa (*boxplot*) devido a prover uma melhor análise.
    - O gráfico de linha é interessante utilizar comparando a evolução dos algoritmos, exemplo: Tempo X Tamanho do Conjunto.

Forma de Entrega

O trabalho pode ser feito em grupos de até três alunos. A equipe deverá criar um relatório em formado de artigo, utilizando o modelo SBC. Nesse relatório deverá contar com um título, nome dos alunos, resumo e com as seguintes seções:

- Introdução
- Revisão Bibliográfica
    - Nesta seção vocês apresentarão os algoritmos utilizados:
        - Estratégia utilizada
        - Funcionamento
        - Complexidade
        - Pior e melhor caso
        - Pseudocódigo
        - etc.
- Descrição do experimento
    - Nesta seção descreve-se sobre os conjuntos utilizados e as formas em que os elementos estão dispostos. Ver Seção **Experimentos**. Além disso, nessa seção que são apresentadas as questões de pesquisa, como os algoritmos serão avaliados, aspectos de implementação (Java). Além disso, há as ameaças à validade o qual deverá ser informado o que compromete os resultados do trabalho, por exemplo, os experimentos foram realizados em diferentes máquinas, os conjuntos de dados são insuficiente para determinar uma melhor acurácia dos resultados, etc.
- Resultados e análises
    - Nesta seção os resultados são apresentados.
- Conclusões e Trabalhos Futuros.

Além do relatório a equipe deverá confeccionar uma apresentação para mostrar os resultados obtidos e apresentar os algoritmos diferenciados atribuídos à equipe para a turma: pseudocódigo, como funciona, etc. A apresentação deverá utilizar o modelo de apresentações padrão da UDESC. O relatório e a apresentação deverão estar disponíveis em formato PDF.

Os arquivos deverão estar em um repositório no *GitHub* ou no *GitLab* e o link de acesso do repositório público deverá ser enviado ao professor via *Moodle*. Para isso, uma atividade será criada. O repositório no *GitHub* ou no *GitLab* não será o mesmo do **jSorting**. Para isso, crie um novo repositório no *GitHub* ou no *GitLab.*

Nesse repositório deverá contar com um arquivo README.md que explicará o repositório, como executar os algoritmos e realizar os experimentos (ou seja, prover informações de como replicar os testes, quais programas deve ter instalado no computador: Java 8, Java 9), qual arquivo é o relatório e qual é a apresentação. Essas informações irão auxiliar o professor para avaliar o trabalho. Desse modo, forneça informações detalhadas.

De modo a auxiliar os alunos, no *Moodle* foram disponibilizados exemplos de relatório e apresentação. Devido ao curso focar em linguagem de programação Java, o trabalho deverá ser desenvolvido em Java.

Avaliação

Serão avaliados os seguintes pontos:

- Entrega do código-fonte com os algoritmos de busca e ordenação.
- O trabalho deve ser feito de forma que possa ser compilado e executado nos computadores.
- Entrega da apresentação e do relatório com os resultados da análise.
- Apresentação do trabalho para a turma (funcionamento do algoritmo, dificuldades encontradas na implementação, vantagens/desvantagens do algoritmo, análise dos resultados).
- O trabalho deve ser entregue no formato que foi especificado na Seção **Forma de Entrega**.
- Para cada defeito encontrado será descontado ponto.
- Corretude, o algoritmo realiza a busca e a ordenação corretamente.
- Clareza do código e comentários.

Equipes e Algoritmos

Cada equipe deverá implementar todos os algoritmos de busca e ordenação de métodos simples, bem com os algoritmos *Quick Sort* e *BogoSort.* Além desses algoritmos, cada equipe deverá implementar um conjunto de algoritmos de métodos eficientes definidos pelo professor.

Equipes:

1. **Leandro, Ricardo, Marcelo, Aline**
2. **Patrick, Deyvison, Leonardo**

As equipes, identificadas numericamente, deverão implementar os seguintes algoritmos de ordenação:

Equipe	Algoritmo 1	Algoritmo 2	Algoritmo 3	Algoritmo 4
1	*Heap Sort*	*Counting Sort*	*Radix Sort*	*Bucket Sort*
2	*Merge Sort*	*Gnome Sort*	*Timsort*	*Cocktail Sort*

Cronograma

Data da entrega do link do repositório público contendo o relatório, a apresentação e os arquivos fontes via *Moodle*: 17/06/2018 (até às 23:00)

Data da apresentação: 21/06/2018.

Apresentações

As apresentações contarão com 45 minutos para cada equipe, sendo dispostas da seguinte forma:

Horário	Equipe
18:15 – 19:15	1
19:15 – 20:15	2

Observações:

- Todos os membros da equipe devem apresentar, salvo atestado médico.
    - Dica:
        1. Como há três algoritmos de ordenação e cada equipe conta com três alunos, cada aluno poderá apresentar um algoritmo de ordenação
        2. Separem a apresentação para que cada aluno apresente o mesmo tempo
- Se não haver todos os membros presentes, salvo atestado médico, a nota atribuída será **zero**.
- Todos os membros presentes devem estar preparados para responderem questões sobre o trabalho, por exemplo: explicar se determinado ajuste no código como pode afetar o algoritmo.
- Se uma equipe não apresentar, a próxima equipe será chamada. Ou seja, se a próxima equipe não estiver pronta, por exemplo, membros da equipe não estão presentes, então, a nota atribuída será **zero**.
- Aprecia-se que as equipes mostrem os códigos funcionando no momento da apresentação.
    - Dicas:
        1. Não necessita mostrar todos os passos apenas: como era o vetor antes, depois, tempo de busca.
        2. Para exemplificar, utilizar apenas um conjunto de dados devido ao tempo de apresentação.
        3. Enquanto um aluno apresenta outro pode estar executando o código.

jSorting

De modo a promover uma experiência de desenvolvimento de software aos discentes, um projeto da disciplina foi criado e denominado jSorting. Esse projeto contém um padrão de arquitetura que utiliza interfaces e classes abstratas, bem como a aplicação do gerenciador de dependências Maven. Esse projeto está disponível no GitHub por meio do link: https://github.com/ceplan/jSorting

A partir desse projeto, cada equipe **poderá** realizar um "*Fork*" do projeto e criar **apenas** os algoritmos que lhe foram designados. Após a criação dos algoritmos e seus respectivos casos de testes (utilizando jUnit), a equipe **poderá** realizar um "*Pull Request*" ao projeto principal. Desse modo, o projeto conterá todos os algoritmos fornecidos aos discentes e assim provendo, além da experiência de desenvolvimento de software, o compartilhamento do conhecimento para outras pessoas que desejarem conhecer esses algoritmos.

Dicas:

1. Usando Maven, instale o projeto jSorting na sua máquina, já com os algoritmos desenvolvidos. Para isso, execute o comando na pasta do projeto: "mvn install"
2. Posteriormente, crie um novo projeto Maven para os experimentos e utilize o jSorting como dependência. Para isso, vá ao pom.xml e adicione a dependência.

Observação:

1. O uso do projeto **jSorting** não é obrigatório.