Universidade Federal do Rio Grande do Norte - Instituto Metrópole Digital
IMD0029 - Estrutura de Dados Básicas I
Prof. João Guilherme

Atividade Prática 01: Algoritmos de busca + Recursão

1. **Introdução**

Este arquivo descreve como deve ser executada a primeira atividade prática. Esta tarefa é individual e tem como objetivo:

- Exercitar a implementação dos algoritmos de vistos em sala de aula:
    - Busca sequencial
    - Busca binária
    - Recursão

Nas seções a seguir, são apresentados: a descrição da tarefa, os arquivos que devem ser entregues para correção e os critérios de correção que serão seguidos.

2. **Descrição da atividade**

Baixe o arquivo .zip disponibilizado. Ele contém a seguinte estrutura de diretórios e arquivos:

```
Atividade_1/
├── include_c/
│   ├── recursao.h
│   ├── busca_seq_ordenada.h
│   └── busca_binaria.h
├── include_cpp/
│   ├── recursao.hpp
│   ├── busca_seq_ordenada.hpp
│   └── busca_binaria.hpp
├── src_c/
│   └── (implemente aqui os arquivos .c correspondentes aos headers em include_c)
├── src_cpp/
│   └── (implemente aqui os arquivos .cpp correspondentes aos headers em include_cpp)
├── test_c/
│   └── test_algorithms.c
├── test_cpp/
│   └── test_algorithms.cpp
└── Makefile
```

O aluno deverá implementar os algoritmos nas linguagens C ou C++, conforme a sua escolha, colocando as implementações nos diretórios src_c/ ou src_cpp/ correspondentes.

No arquivo test_algorithms.c (para C) ou test_algorithms.cpp (para C++), o aluno precisa:

- Popular os vetores de teste de forma adequada.
- Ajustar as chamadas de função de acordo com a assinatura correta.
- Inserir as entradas e o valor esperado para os testes utilizando a macro RUN_TEST(...).
- Garantir que os nomes das funções utilizadas nos testes estejam em conformidade com os nomes definidos nos arquivos de cabeçalho e de implementação.

Exemplo de chamada de teste em C:

```
int arr[] = {1, 2, 3, 2, 1, 4};
RUN_TEST("Busca ordenada", conta_especialidades_distintas(arr, 6), 4);
```

Após implementar as funções e configurar corretamente os testes, o aluno pode compilar e executar os testes com os seguintes comandos, no terminal:
Para C:

```
make test_c
```

Para C++:

```
make test_cpp
```

Se tudo estiver correto, o executável será gerado e os testes serão executados automaticamente, exibindo se cada teste passou ou falhou.

3. **Entregáveis**

O trabalho deverá ser entregue via SIGAA até **23h59 do dia 23 de setembro de 2025**.

O entregável deverá ser um arquivo .zip contendo os seguintes itens:

1. Relatório das questões teóricas, em formato compatível com leitores de PDF.
2. Captura de tela (print) dos resultados obtidos após a execução dos testes com o comando make anexados ao relatório mencionado no item 1.
3. O link do repositório onde foi feita a implementação.
4. No arquivo (README) do repositório, toda a especificação das funções implementadas, descrevendo claramente o funcionamento de cada uma, os parâmetros esperados e as instruções de uso dos scripts fornecidos.
5. Todos os arquivos-fonte das questões práticas, devidamente organizados nos diretórios apropriados.

Além disso, os arquivos do relatório e do .zip devem obedecer rigorosamente o seguinte padrão de nomenclatura:

<CODIGO_DISCIPLINA> - <PRIMEIRO_NOME>_<ULTIMO_NOME> - <MATRICULA> - <NOME_DOCUMENTO>

Desta forma, os arquivos descritos anteriormente deverão ser nomeados da seguinte maneira:
IMD0029 – JOAO_GUILHERME – 20250123 – Relatorio.PDF
Da mesma forma, o arquivo .zip contendo os arquivos acima também deverá seguir nomenclatura similar, como apresentado a seguir:
IMD0029 – JOAO_GUILHERME – 20250123 – Entregavel.zip

Os arquivos deverão ser entregues via SIGAA.

Percebam que não há acentos nos nomes dos arquivos.

4. **Critérios de correção**

A avaliação será feita com base em uma análise criteriosa dos documentos e códigos entregues. Os seguintes aspectos serão considerados:

Critérios principais

- Clareza e objetividade da escrita no relatório;
- Organização e estrutura do relatório;
- Qualidade, legibilidade e organização do código-fonte;
- Coerência entre os resultados apresentados e as discussões realizadas.

Penalizações

O aluno poderá ser penalizado nos seguintes casos:

- Alguma das funções de busca não foi corretamente implementada;
- Alguma das funções de busca não foi devidamente documentada;
- Falta de comentários explicativos nas funções implementadas;
- Desrespeito ao padrão de nomenclatura dos arquivos solicitados.

Plágio e uso indevido de ferramentas IA

Trabalhos que apresentarem plágio ou uso indevido de ferramentas de inteligência artificial serão desconsiderados e receberão nota zero.

Se houver suspeita de plágio, o aluno poderá ser convocado a apresentar oralmente seu trabalho em sala de aula. O não comparecimento para essa apresentação, dentro do prazo de uma semana após a convocação, também implicará nota zero.

5. Questões

5.1 Primeira versão defeituosa

Você é um gerente de produto liderando uma equipe no desenvolvimento de uma nova aplicação. Infelizmente, a versão mais recente do seu produto falhou na verificação de qualidade. Como cada versão é desenvolvida com base na anterior, todas as versões subsequentes a uma versão defeituosa também serão defeituosas.

Suponha que você tenha n versões numeradas de 1 a n e deseja identificar a primeira versão defeituosa, que causa todas as versões seguintes também serem defeituosas.

Você possui uma API bool isBadVersion(int version) que retorna true se a versão for defeituosa e false caso contrário. Implemente uma função para encontrar a primeira versão defeituosa, minimizando o número de chamadas à API.

Especificações da Função:

- **Nome:** busca_binaria
- **Parâmetros:**
    - int n: o número total de versões.
- **Retorno:** o número da primeira versão defeituosa.

Requisitos:

- Utilize o algoritmo de busca binária para minimizar o número de chamadas à API isBadVersion.

- A função isBadVersion(int version) será fornecida. Para fins de teste, você pode simular essa função conforme necessário.

```
// Suponha que a versão 4 seja a primeira defeituosa
int n = 5;
int bad = 4;

// Simulação da API isBadVersion
bool isBadVersion(int version) {
   return version >= bad;
}

int busca_binaria(int n) {
…
}
```

Nota:

- A função busca_binaria deve ser implementada no arquivo src_c/busca_binaria.c para C ou src_cpp/busca_binaria.cpp para C++.
- O cabeçalho correspondente deve ser colocado em include_c/busca_binaria.h ou include_cpp/busca_binaria.hpp, respectivamente.
- Certifique-se de que a função isBadVersion esteja acessível no contexto da função busca_binaria.

5.2 Equipes iguais e diversas

Uma turma possui alunos com diferentes especialidades. Cada especialidade é representada por um número inteiro. A professora deseja saber **quantas especialidades diferentes estão presentes na turma**.

Você deverá escrever uma função busca_seq_ordenada que percorre um vetor ordenado e diz se uma especialidade já foi encontrada. Com ela, você deverá implementar uma função conta_especialidades_distintas, que usa a busca sequencial ordenada para contar quantas especialidades únicas existem.

Especificações:

Você deve implementar:

- Uma função de ordenação (por exemplo, bubble sort).
- A função busca_seq_ordenada(int arr[], int n, int alvo), que retorna o índice do elemento ou -1 se não encontrar.
- A função conta_especialidades_distintas, que usa a busca_seq_ordenada para não contar repetidos.

Função principal esperada no teste:

```
int conta_especialidades_distintas(int arr[], int n);
```

Exemplo de uso no teste

```
RUN_TEST("Especialidades distintas (ex1)",
      conta_especialidades_distintas(arr1, 6),
      /* esperado */ 3);

// Exemplo de entrada:
int arr1[] = {4, 2, 1, 4, 2, 1};
// Após ordenação: [1, 1, 2, 2, 4, 4]
// Esperado: 3 (valores únicos: 1, 2, 4)
```

5.3 Contagem Recursiva

Você deve implementar a função recursao que recebe uma string e um caractere e retorna, de forma recursiva, quantas vezes o caractere aparece na string.

A função deve obrigatoriamente ser recursiva, ou seja, não pode utilizar laços (for, while, do-while).

A função a ser implementada deve ter o nome:

```
int recursao(const char *str, char alvo);   // C
```

```
int recursao(const std::string &str, char alvo);  // C++
```

Exemplos de uso

```
RUN_TEST("Recursiva", recursao("banana", 'a'), /* esperado */ 3);
```

6. **Questões teóricas - Responda todas as questões utilizando as suas palavras**

Questão 6.1 — Complexidade de Algoritmos de Busca

Tema: Busca sequencial vs. busca binária

Enunciado:
Compare os algoritmos de busca sequencial e busca binária em termos de complexidade, aplicabilidade e limitações**. Discuta:**

- Em quais contextos cada um é mais apropriado?
- Quais são os pré-requisitos para aplicar a busca binária corretamente?
- Como a ordenação influencia a escolha do algoritmo de busca?
  Ilustre seu argumento com um exemplo concreto de uso para cada tipo de busca.

Questão 6.2 — Impacto da Ordenação nos Algoritmos de Busca

Tema: Pré-processamento e desempenho de algoritmos

Enunciado:
Considere uma situação onde é necessário fazer várias buscas sobre um mesmo conjunto de dados.

- Vale a pena ordenar os dados previamente?
- Qual o impacto dessa escolha no desempenho do sistema a longo prazo?

- Como isso se relaciona com o tempo de execução dos algoritmos envolvidos?

Questão 6.3 — Recursão x Iteração

Tema: Estratégias de resolução de problemas

Enunciado:
Explique as principais diferenças entre recursão e iteração na resolução de problemas algorítmicos. Discuta:

- Quais são os prós e contras de cada abordagem?
- Em quais tipos de problemas a recursão pode ser mais vantajosa?
- Existe alguma desvantagem de desempenho ou consumo de memória?

Inclua um exemplo simples em pseudocódigo ou linguagem C/C++ para ilustrar sua resposta.

Questão 6.4 — Análise de um Cenário Real

Tema: Escolha de algoritmos na prática

Enunciado:
Imagine que você trabalha no setor de tecnologia de uma empresa de logística. O sistema precisa localizar rapidamente pacotes com base em códigos de rastreio que chegam em tempo real. Os dados chegam desordenados.

- Qual algoritmo de busca você recomendaria para este sistema?
- Justifique tecnicamente sua escolha considerando tempo de resposta, necessidade de ordenação e custo computacional.
- Caso os dados pudessem ser pré-processados, sua escolha mudaria?