Universidade Federal do Piauí - Campus Senador Helvídio Nunes de Barros – CSHNB

Disciplina: Estrutura de dados I

Relatório de análise da execução com algoritmos recursivos, ordenação e busca.

Dupla: João Carlos de Sousa fé e Vitor José Ferreira dos Santos Santana

1. Resumo do projeto

O projeto envolve a resolução de 3 exercícios, onde cada um deles é necessário aplicar técnicas de algoritmos utilizados em estruturas de dados, a fim de manter a organização e otimização do código.

2. Introdução

As técnicas utilizadas no projeto definem estruturas de dados aplicadas de maneira clara e objetiva, a fim de manter um código otimizado e modular. As técnicas e algoritmos utilizados foram:

TADS: Tipos abstratos de dados;
Recursão: recursão sem pendência e com pendencia;
Ponteiros: passagem de parâmetros por referência e retornos;
Busca: busca binária;
Ordenação: Algoritmos Quick Sort e Insertion Sort.

3. Seções Específicas

Detalhes arquiteturais: O sistema operacional utilizado para a realização dos exercícios foi o Windows 10, com o compilador *C MinGw da Microsoft* e com o ambiente de desenvolvimento *Falcon C + +* e sistema de versionamento de código *Git e Github*.

Exercícios:

1ª questão: a questão pede a resolução do determinante da matriz até ordem 4. Foram utilizadas técnicas de modularização de funções, operações recursivas, passagem por referência, chamadas de funções e utilização de técnicas matemáticas conhecidas como *cofatores* e transformação de triangulação da matriz. bibliotecas padrões utilizadas: *stdio.h, stdlib.h.*

2ª questão: exige a ordenação de matriz de *strings, contagem de dígitos* de uma determinada string e quantidade de consoantes de uma coluna inteira. Foram utilizadas técnicas de modularização, ordenação (algoritmo *Quicksort),* busca binária, alocação de matriz dinamicamente, *typedef*, transformações de *strings e* recursivo com pendência e sem pendência. Bibliotecas utilizadas: *stdio.h, stdlib.h, string.h, ctype.h.*

3° questão: Exige operações de inserção de dados, busca, exclusão e ordenação. Foram utilizadas TADS, diretivas do pré processador (*#define*), constantes, busca binária, ordenação (Quicksort), manipulação de strings, passagem de parâmetros por referência (ponteiros), funções com e sem retorno de dados. Bibliotecas utilizadas: *stdio.h, stdlib.h, string.h, locale.h.*

4. Resultados da execução

Após as escritas de código de cada exercício, foram realizados vários tipos de testes: testes de validação, entradas de dados variadas e teste das funções individualmente para avaliar desempenho e comportamento com diferentes dados. Depois foram avaliadas as chamadas de todas as funções como um todo, a fim de se saber como se dá a comunicação entre elas e se os retornos e passagem de dados é tido como o esperado.

Exercício 1: dados de exemplo e ordem de chamada das funções

<table>
<tr><th>Entrada de dados</th><th>Funções chamadas (sequencialmente)</th></tr>
<tr><td>

Entrada 1:

2

Entrada 2:

5	6
8	10

</td><td>

1	“preencherMatriz(ordem, 0, 0, matriz)”
2	“mostrarMatriz(ordem, 0, 0, matriz)”
3	“transformarEmTriangulo(ordem, matriz, 0, 0, 0, &count)”
4	“espalhar(i, j, k, ordem, matriz)” ou “factorCalc(k, i, j, ordem, matriz)”
5	troca(i, j, k, ordem, matriz);
6	multiplicarMatriz(0, &calc, ordem, matriz);

</td></tr>
</table>

A função ““preencherMatriz(ordem, 0, 0, matriz)”” é utilizada para o usuário fornecer todos os dados da matriz e em seguida é mostrada para o mesmo as informações preenchidas através da função “mostrarMatriz(ordem, 0, 0, matriz)”. Assim que dados são preenchidos e mostrados, a função “transformarEmTriangulo(ordem, matriz, 0, 0, 0, &count)” é chamada para cálculo do determinante. Ela aplica uma fórmula conhecida como “transformação da matriz em uma matriz triangular”, onde multiplicações e somas sucessivas são realizadas. Após aplicado a “triangulação”, a função “ multiplicarMatriz(0, &calc, ordem, matriz)” é chamada, para realizar a multiplicação da diagonal principal com os dados preenchidos pela função de “triangulação”.

Exercício 2: dados de exemplo e ordem de chamada das funções

<table>
<tr><th>Opções de entrada da aplicação</th><th>Funções chamadas (sequencialmente)</th></tr>
<tr><td>1</td><td>
<table>
<tr><td>1</td><td>“alocarMatrizStrings(linhas, colunas”</td></tr>
<tr><td>2</td><td>“alocarVetorString(linhas * colunas)”</td></tr>
<tr><td>3</td><td>lerMatriz(linhas, colunas, matriz, 0, 0);</td></tr>
</table>
</td></tr>
<tr><td>2</td><td>
<table>
<tr><td>1</td><td>"mostrarMatrizOriginal(linhas, colunas, matriz, 0, 0)”</td></tr>
</table>
</td></tr>
<tr><td>3</td><td>
<table>
<tr><td>1</td><td>“ordenarFinal(linhas, colunas, matriz, vetor_strings)”</td></tr>
<tr><td>2</td><td>“quicksort(vetor_string, l)”</td></tr>
<tr><td>3</td><td>“tolowerAll(arr[i], temp_string1)”
“tolowerAll(arr[length - 1], temp_string2);”</td></tr>
<tr><td>4</td><td>“strcmpi(temp_string1, temp_string2)”</td></tr>
<tr><td>5</td><td>“swap(arr + i, arr + piv++)”</td></tr>
</table>
</td></tr>
<tr><td>4</td><td>
<table>
<tr><td>1</td><td>“mostrarMatriOrdenada(linhas, colunas, matriz, 0, 0)”</td></tr>
</table>
</td></tr>
<tr><td>5</td><td>
<table>
<tr><td>1</td><td>“contarDigitosLetrasMaiuculas(matriz[linfo][cinfo], 0, strlen(matriz[linfo][cinfo]), 0, 0)”</td></tr>
<tr><td>2</td><td>“isdigit(string[i])”</td></tr>
</table>
</td></tr>
<tr><td>6</td><td>
<table>
<tr><td>1</td><td>“contarColunasStringConsoante(linhas, 0, matriz, cinfo, 0)”</td></tr>
<tr><td>2</td><td>“isalpha(matriz[i][coluna][0])”</td></tr>
<tr><td>3</td><td>toupper(matriz[i][coluna][0])</td></tr>
</table>
</td></tr>
</table>

Após o usuário informar as linhas e colunas da matriz, a mesma é alocada dinamicamente através da função “alocarMatrizStrings(linhas, colunas)”. Essa função primeiramente aloca a

quantidade de linhas necessária . Em seguida, a partir das linhas já alocadas, cada índice é “é preenchido” com uma nova alocação, agora a das colunas. depois das linha e colunas alocadas, ocorre uma nova alocação para as mesmas, a alocação da quantidade de caracteres para cada linha e coluna. Após ter a matriz preenchida com os dados, o usuário terá a opção de ordená-las utilizando de funções conhecidas como quicksort. No quicksort, é aplicado uma técnica de colocação para duas string (temporárias), transformando-as em letras minúsculas para verificação dos caracteres (se um é menor que o outro). Caso seja, às 2 strings originais sem transformação, é repassada para a função de “swap” (troca).

Exercício 3: dados de exemplo e ordem de chamada das funções

<table>
<tr><th>Opções de entrada da aplicação</th><th>Funções chamadas (sequencialmente)</th></tr>
<tr><td>a</td><td><table>
<tr><td>1</td><td>“cadastrarCurso(cursos, &count_cursos)”</td></tr>
<tr><td>2</td><td>“PesquisarCurso(cursos, cursos[*cc].codigo_curso*cc)”</td></tr>
<tr><td>3</td><td>“quicksortCursos(cursos, 0, *cc)”</td></tr>
</table></td></tr>
<tr><td>b</td><td><table>
<tr><td>1</td><td>“cadastrarDisciplina(cursos, &count_cursos)”</td></tr>
<tr><td>2</td><td>“PesquisarCurso(cursos, temp_codigo, *cc)”</td></tr>
<tr><td>3</td><td>“quicksortDisciplinas(cursos[i].disciplinas, 0, cursos[i].qtdD)”</td></tr>
</table></td></tr>
<tr><td>c</td><td><table>
<tr><td>1</td><td>“mostrarCurso(cursos, &count_cursos)”</td></tr>
<tr><td>2</td><td>PesquisarCurso(cursos, temp_codigo, *cc);</td></tr>
</table></td></tr>
<tr><td>d</td><td><table>
<tr><td>1</td><td>“mostrarDiciplina(cursos, &count_cursos)”</td></tr>
<tr><td>2</td><td>“PesquisarDisciplina(cursos[i].disciplinas, temp_codigo, cursos[i].qtdD”</td></tr>
</table></td></tr>
<tr><td>e</td><td><table>
<tr><td>1</td><td>“mostrarCursosTurno(cursos, &count_cursos)”</td></tr>
<tr><td>2</td><td>“strcmp(turno, cursos[i].turno)”</td></tr>
</table></td></tr>
</table>

<table>
<tr><td>f</td><td><table>
<tr><td>1</td><td>“mostrarDisciplinasCurso(cursos, &count_cursos)”</td></tr>
<tr><td>2</td><td>“PesquisarCurso(cursos, temp_codigo, *cc)”</td></tr>
<tr><td>3</td><td></td></tr>
</table></td></tr>
<tr><td>g</td><td><table>
<tr><td>1</td><td>“mostrarDisciplinasPeriodo(cursos, &count_cursos)”</td></tr>
<tr><td>2</td><td>“PesquisarCurso(cursos, temp_codigo, *cc)”</td></tr>
</table></td></tr>
<tr><td>h</td><td><table>
<tr><td>1</td><td>“removerDisciplina(cursos, &count_cursos)”</td></tr>
<tr><td>2</td><td>“indexCurso = PesquisarCurso(cursos, temp_codigo, *cc)”</td></tr>
<tr><td>3</td><td>“PesquisarDisciplina(cursos[indexCurso].disciplinas, temp_codigo, cursos[indexCurso].qtdD)”</td></tr>
</table></td></tr>
<tr><td>i</td><td><table>
<tr><td>1</td><td>“removerCurso(cursos, &count_cursos)”</td></tr>
<tr><td>2</td><td>“PesquisarCurso(cursos, temp_codigo, *cc)”</td></tr>
<tr><td>3</td><td></td></tr>
</table></td></tr>
</table>

Nesse exercício, assim que o usuário cadastra a informação (curso e disciplina), a mesma já é ordenada (Quicksort) e já incrementa um novo espaço para adição futura de novos dados (máximo de 10). Ao buscar por uma informação, a busca binária é chamada (já que os com certeza já foram ordenados), trazendo o índice, caso encontre, para ser utilizada para a exibição das informações pretendidas.

5. Conclusão.

A resolução dos exercícios propostos tiveram várias aplicações de técnicas de programação, aprendizagem e criação de novas técnicas e aperfeiçoamentos das técnicas já aprendidas. Utilizando-se de modularização das funções, resultaram numa maior clareza e solução dos problemas encontrados durante o desenvolvimento, assim como as novas técnicas trouxeram alguns problemas, ao final elas ajudaram no encontro dos erros, permitindo a realização dos testes de maneira fácil e prática.

Como esse trabalho foi realizado em dupla, o sistema de versionamento de código Git e Github foram utilizados para uma melhor coordenação de trabalho. O link do repositório está disponível em: joaocarlos-losfe/ED1_Avalia1 (github.com) No repositório está contido todo o histórico de tráfego das alterações e correções de erros encontrados nos exercícios e como foi a resolução dos mesmos.