UNIVERSIDADE FEDERAL DO PIAUÍ - UFPI
CAMPUS SENADOR HELVÍDIO NUNES DE BARROS - CSHNB

SISTEMAS DE INFORMAÇÃO – SI

HUMBERTO JOSÉ DA SILVA JÚNIOR

RELATÓRIO DE ANÁLISE DE ALGORITMOS COM O PARADIGMA FUNCIONAL DO TRABALHO REFERENTE A SEGUNDA AVALIAÇÃO

1. Resumo

O presente relatório descreve as resoluções dos exercícios propostos no trabalho referente à primeira avaliação da disciplina de Estrutura de Dados I. Os algoritmos de solução foram desenvolvidos no paradigma imperativo, utilizando a linguagem de programação C. Os mesmos foram desenvolvidos com os conhecimentos apresentados na disciplina.

2. Introdução

Este relatório tem como objetivo descrever o trabalho complementar à primeira nota da disciplina Estrutura de Dados I, do curso superior de Sistemas de Informação ofertado pela Universidade Federal do Piauí, Campus Helvídio Nunes de Barros, em Picos - PI.

O presente trabalho tem como propósito proporcionar ao discente [do curso] conciliar os conhecimentos teóricos à prática por meio das resoluções de enigmas matemáticos, regras de negócios não triviais, em diferentes níveis de complexidade.

As atividades propostas no supramencionado trabalho, foram solucionadas utilizando o paradigma imperativo, especificamente na linguagem de programação C, stack adotada para fins de estudo da disciplina.

3. Sessões Específicas

a. Informações técnicas

Foi utilizado um notebook de marca Acer, modelo aspire 3, com um processador Intel Core i5 de sétima geração, com oito gigabytes de memória ram, quinhentos e doze gigabytes SSD (Solid State Drive), um terabyte de arazendanto HDD (Hard Disk Drive) com o sistema operacional Linux Mint.

Para o desenvolvimento dos algoritmos, foi utilizado o Visual Studio Code - como editor de códigos, e o GCC na versão 9.3 - como compilador.

b. Funções reutilizáveis

A função quicksort recebe como parâmetro a matriz que será ordenada, o início - índice inicial (0), e o fim - índice final (N-1). A função inicialmente irá verificar se o início é menor que o fim, condição de parada, caso a condição seja aceite, será realizado calculado o posição do pivô que será calculada por meio da função partition, posteriormente, é chamado a função quicksort recursivamente passando a matriz, início, e sublista a esquerda, ou seja, a posição do pivô menos um (p-1), por fim, a função será

novamente chamada, com a sublista da esquerda, ou seja, passando a matriz, o pivô mas um (p+1), e o fim.

A função auxiliar, partition, que trata de fazer a movimentação em relação ao pivô, colocando os elementos menores a esquerda e os maiores a direita. Inicialmente é calculado o pivô, que será o último elemento da matriz (m[e]), em seguida é calculado a barra roxa, atribuído a variável “i” o início, em seguida, a barra roxa, que será a variável “j”, que sempre estará avançando dentro do laço de repetição for, ela vai percorrer do ínicio até o fim menos um (e-1). Dentro do laço, é verificado se o elemento da posição “j” é menor ou igual ao pivô, caso seja aceita, será realizada a troca de posição, entre o elemento “j” - o que sempre avança, com o “i” - parte dos menores. Para realizar troca de posições, será passado o para a função swap, a matriz e os índices “i” e “j”, também é o caso onde é incrementado a variável “i”, referente aos menores elementos.

Após ser comparado o elemento antecessor do pivô, vai ser realizada a troca do pivô, ou seja, colocado no meio entre as regiões que têm os menores elementos e maiores que ele, trocando a posição “i” com a fim “e”. Por fim, será retornada a posição do pivô, que fica armazenado na variável “i”.

A função swap, trata de receber como parâmetro uma matriz e duas posições nas quais será realizada a troca de posições entre elas.

Uma função menu, foi desenvolvida para além de mostrar as opções, fazer a bridge entre o main do programa e as demais funções.

c. Exercício 02

Na segunda questão, é solicitado que seja implementado um programa que leia uma matriz de strings, e então, solucione quatro problemas, sendo eles: Ordenar cada uma das colunas da matriz; Mostrar cada coluna da matriz antes e depois de ordenar; Dado a coluna e linha, mostra a quantidade de dígitos e letras maiúsculas; e que dado uma coluna informe a quantidade de strings da mesma que iniciam com consoante.

Para ordenar as colunas, é solicitado para que seja utilizado o algoritmo quicksort. A função quicksort foi implementada, ela recebe como parâmetro uma matriz tridimensional do tipo char, um inteiro como segundo que representa o início, um segundo inteiro como o fim, e por último, outro inteiro que representa a coluna que vai ser ordenada.

Para mostrar a matriz antes e depois de ordenada, foi implementada uma função, “showMatriz”, que trata de exibir a todas as colunas e suas respectivas linhas. Para exibir a matriz antes de ordenar, é passada como parâmetro para a função “showMatriz”, uma cópia da matriz lida, ou seja, a matriz original.

Para exibir a quantidade de dígitos e letras maiúsculas de uma string da matriz dada a coluna e sua linha, foi desenvolvida a função “qtdDigitsAndCapitalLeters”, que trata de devolver por referência os resultados. A função recebe como parâmetro a string na qual será verificada, e dois endereços de memória no qual serão armazenados os resultados. Na mesma, é implementado um laço de repetição que percorre toda a string, em cada elemento da mesma é verificado se é um dígito ou uma letra maiúscula, caso algum dos condicionais seja verdadeiro, é incrementado no conteúdo do seu respectivo ponteiro.

Por último, no exercício de número dois, é solicitado que calcule a quantidade de strings de uma determinada coluna que iniciam com consoante.

Para este problema, foi implementada a função “qtdStartingWithCosoant”, a mesma recebe como parâmetro uma matriz e uma coluna na qual será verificada. A função percorre todos os valores da coluna informada verificando a negação da função “isVowals” é verdadeira, caso seja, a variável “r”, que é iniciada com zero, é incrementada e por fim é retornada.

A função “isVowals” trata de verificar se um caractere é vogal, caso seja, é retornado um (1), caso contrário zero (0).

d. Exercício 03

No exercício três, é solicitado que seja desenvolvido um programa para cadastrar, buscar e imprimir cursos e suas disciplinas.

Para este exercício, foi desenvolvido dois TAD (Tipo Abstrato de Dados), um para representar o curso - “Course” e outro para a disciplina - “Discipline”.

O tipo “Course” é composto pelo inteiro id - para representar o código, char name - para o nome, inteiro qtdPeriods - para a quantidade de períodos, inteiro workLoad - para carga horária e um char shift - para o turno.

O tipo “Disicpline” é composto pelo inteiro id - para representar o código, inteiro fkCourse - para o código do curso, char name - para o nome, inteiro período - para o período da disciplina, e o inteiro workLoad - para a carga horária da disciplina.

Para ler o curso, foi implementada a função setCourse, a mesma trata de ler os dados do usuário. A mesma recebe como parâmetro um array do tipo “Course”, um ponteiro da variável contador lógico de cursos, e o tamanho total de cursos.

Inicialmente é verificado se o contator de cursos cadastrados é menor que o tamanho total de cursos predefinidos, caso seja a condição seja favorável, será solicitado o código para o curso, em seguida o mesmo será verificado por meio da função “getCourseById”, se o mesmo já não foi

cadastrado, caso o código ainda não exista, será solicitado os demais dados para o novo curso, caso contrário, será solicitado um novo código.

Para cadastrar disciplina o processo é semelhante, a função “setDiscipline” foi implementada, a mesma recebe como parâmetro um array do tipo Discipline, um array do tipo Course, ponteiro para a variável contador lógico de cursos, outro ponteiro para a contador lógico disciplina e um inteiro que corresponde ao tamanho total predefinido de disciplinas.

Inicialmente é solicitado o código para a disciplina e em seguida verificado se o mesmo já não existe, ou seja, se já foi cadastrado, por meio da função “getDisciplineById”, caso a condição seja favorável, será solicitado do usuário o código da disciplina e verificado se a mesma existe, por meio da função “getCourseById”, caso o curso exista, será solicitado o nome da disciplina e em seguida o período da mesma, o valor segue sendo colicitado enquanto o mesmo for maior que a quantidade de períodos do curso, e por último, é solicitado a carga horária da disciplina e verificado se a mesma é múltipla de quinze pela função “checkMulti15”, caso não seja, é solicitado novamente, até que a condição seja atendida.

A função chekMulti15, recebe como parâmetro um número e retorna um (1) caso o resto da divisão do mesmo por quinze seja igual a zero, caso contrário é retornado zero (0).

Para mostrar um curso ou disciplina é solicitado um código do mesmo que seja mostrar e é passado para a função “getCourseById” ou “getDisciplineById”, caso existe um código o mesmo é mostrado por meio da função “getCourse” ou “getDisciline”.

Para retornar os cursos por turno, é solicitado via input o turno que deseja buscar e passada como parâmetro para a função “getCourseByShift”. A função supramencionada recebe um array do tipo course, o início, o fim, um inteiro para representar se foi encontrado ou não cursos, e a string do cursos informado. A mesma percorrer todos os cursos e verifica se o turno é igual ao informado, caso seja atendida é impresso através da função “getCourse”, a função continua recursivamente e incrementando o início em mais um (s+1), até que a condição de parada seja atendida, e caso nenhum curso seja encontrado é impresso uma mensagem informativa.

As opções de busca disciplina por curso ou por período são semelhantes a busca de cursos por turno. Para busca por curso, é comparada o valor informado pelo input é igual ao de cada um dos cursos, e por período é verificada se o período informado é igual a algum das disciplinas, Se algum dos condicionais for atendido, é mostrada pela função “getDiscipline”, caso contrário é mostrada uma mensagem informativa.

Para a opção remover disciplina, é solicitado via input o código referente a disciplina na qual será removido, com base no código é atribuído a variável “n” a posição que ela encontra no array e passada para a função

"removeDisciplineById" e por último, é decrementado o valor do contador lógico das disciplinas.

A função "removeDisciplineById", recebe o array do tipo disciplina, um inteiro referente ao início da troca, a última posição e por fim o ponteiro referente ao contador lógico das disciplinas. A mesma chama a função de troca, "swapDisciplina", e faz a troca da posição do código informado para ele mais um, e chama a função remove recursivamente incrementado em mais um, até que a disciplina informada ocupe a última posição.

Para remover um curso, o processo quase semelhante, é adicionado apenas mais um condicional, o mesmo trata de verificar a quantidade de disciplina que o curso informato tem, caso seja zero (0) é removido, caso contrário é informado que o mesmo possui disciplinas cadastradas.

A função recursiva "getQtdDisciplineByCourse", recebe dois arrays, o primeiro do tipo course e o segundo disciplina, início, fim e um inteiro referente ao curso informado. A mesma percorre todas as disciplinas e verifica se o código do curso é igual ao informado, se aceito é incrementado em mais um (r+1) o contador, e retornado no final.

4. Resultados das Execução dos Programas

Para comprovar o funcionamento dos algoritmos solucionados em cada questão, foram realizados inúmeros testes a fim de tratar eventuais erros. Uma tabela de duas colunas - Entrada e Saída, demonstra logo abaixo.

A segunda questão, solicita que seja lido uma matriz de strings para então realizar as seguintes opções: (1) ler matriz, (2) - ordenar, (3) - mostrar matriz ordenada e original, (4) número de dígitos e de letras maiúsculas de uma string, e (5) - quantidade de string que iniciam com consoante de uma determinada coluna.

ENTRADA	SAÍDA
1 - string,int, char bool, vscode,suBlime1, androidstu, xcode.	
2	String ordenada
3	#ORDENADA bool androidstu char suBlime1 int vscode string xcode #ORIGINAL string vscode int suBlime1 char androidstu bool xcode

4 - Coluna: 1, linha: 1	Dígitos: 1 Maiúsculas: 1
5 - Coluna: 1	Consoantes: 3

No exercício de número três (03), é solicitado que faça um menu que leia, cadastre, busque e imprima cursos e suas disciplinas. Menu: (1) - Cadastrar curso, (2) - Cadastrar disciplina, (3) - Mostrar um curso, (4) - Mostrar uma disciplina, (5) - Cursos por turno, (6) - Disciplinas de um Curso, (7) - Disciplina por período, (8) - Remover uma disciplina, e (9) - Remover curso.

ENTRADA	SAÍDA

1 - 3,Biologia,10,2600,Integral;
1,Nutrição,8, 2400, Manhã; 2 ,
Sistemas, 8, 2400, Noite.
	# CURSOS
	Código = 1, Nome = Nutrição, Qtd de períodos = 8, Carga horária = 2400, Turno = Manhã;
	Código = 2, Nome = Sistemas, Qtd de períodos = 8, Carga horária = 2400, Turno = Noite;
	Código = 3, Nome = Biologia, Qtd de períodos = 10, Carga horária = 2600, Turno = Integral

2 - 2, 2, ED I, 3, 60; 1, 2, Estatística, 3, 60; 3, 3, Bioestatística, 7, 45.	# DISCIPLINAS
	Código = 1, Curso = 2, Nome = Estatística, Período = 3, Carga horária = 60;
	Código = 2, Curso = 2, Nome = ED I, Período = 3, Carga horária = 60;
	Código = 3, Curso = 3, Nome = Bioestatística, Período = 7, Carga horária = 45.

3 - 4	Curso não encontrado! :(
3 - 2	Código = 3, Nome = Biologia, Qtd de períodos = 10, Carga horária = 2600,

	Turno = Integral
4 - 0	Disciplina não encontrada! :(
5 - Noite	Código = 2, Nome = Sistemas, Qtd de períodos = 8, Carga horária = 2400, Turno = Noite
6 - 2	Código = 1, Curso = 2, Nome = Estatística, Período = 3, Carga horária = 60; Código = 2, Curso = 2, Nome = ED I, Período = 3, Carga horária = 60.
7 - 3	Código = 1, Curso = 2, Nome = Estatística, Período = 3, Carga horária = 60; Código = 2, Curso = 2, Nome = ED I, Período = 3, Carga horária = 60.
8 - 2	Sucess!
4 - 2	Disciplina não encontrada! :(
9 - 1	Success!

5. Conclusão

É perceptível a importância do trabalho, como já supracitado, em aliar a teoria das aulas com a prática das resoluções dos problemas propostos, que apesar de serem um pouco complexos, foi bem divertido de resolvê-los.

Por fim, após a resolução dos problemas, fica uma sensação de dever cumprido e é notório que o trabalho já supracitado, exigem tempo e acima de tudo muita dedicação.