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 REFERENTE AO TRABALHO DA SEGUNDA AVALIAÇÃO DA DISCIPLINA DE ESTRUTURAS DE DADOS I

1. Resumo

O presente relatório descreve as resoluções dos exercícios propostos no trabalho complementar da segunda avaliação da disciplina de Estruturas 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 segunda 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 a distribuição Zorin OS, do sistema operacional linux.

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 comuns

Algumas funções são comuns nos códigos, ou seja, são chamadas mais de uma vez em um ou mais scripts,sendo elas: A função nomeada de “init” tanto em fila ou pilha - seja dinâmica ou estática, trata de iniciar estrutura passada; A função “isEmpty” retorna “1” caso a estrutura esteja vazia e “0” caso contrário;

A função enqueue (enfileirar), tem como finalidade enfileirar um “tdado”, na fila, a mesma recebe como parâmetro o ponteiro de uma fila e o dado que será adicionado na mesma.

Na fila estática, inicialmente, a função verifica se a mesma está cheia, caso contrário a mesma incrementa o índice atual e adiciona o elemento no mesmo, a função retorna “1” se conseguiu adicionar com sucesso e “0” se houve algum erro no processo.

Já na fila dinâmica, inicialmente é alocado um espaço na memória e em seguida verificado se o processo foi realizado com sucesso e adicionado o dado no novo endereço, e o próximo para “NULL”, posteriormente é verificado se o fila está vazia e adicionado no início, caso contrário, é adicionado no final.

A função dequeue (desenfileirar) - como o próprio nome já diz, tem o objetivo oposto da função enqueue, ou seja, remover o primeiro elemento da fila.

Na fila estática, o processo é bem simples, a mesma atribui o valor inicial da fila a um variável auxiliar, incrementa o inicial em mais um (s+1) e devolve o valor contido na variável auxiliar.

Na dinâmica, o processo é um pouco diferente, o endereço do início da fila é atribuído a uma ponteiro auxiliar e em seguida verificado se o mesmo é diferente de “NULL”, caso seja atendida, o início da fila pronta para o próximo e o novo dado removido é atribuir a uma variável auxiliar para ser retornado, e o endereço do dado na lista é liberado.

c. Exercício 01

Na primeira questão é solicitado que seja desenvolvido um programa que simule o estacionamento de uma única alameda, e toda vez que um carro saia do mesmo, que seja exibido o número de vezes em que o carro foi manobrado para fora do estacionamento para permitir que outros carros saíssem, além de mostrar quantos carros foram manobrados para que ele saísse.

Para este problema, um TAD (Tipo Abstrato de Dado) nomeado de “Car”, com os campos placa - do tipo char, e dois contadores - um para contar a quantidade vezes que o mesmo saiu e outro para monitorar a quantidade de carros que precisaram ser manobrado para que o mesmo saísse.

Inicialmente o usuário entra com a placa do veículo, e posteriormente se o mesmo vai entrar ou sair do estacionamento, caso seja para entrar, a placa é verificada se o veículo já esteja contido no mesmo, por meio da “searchQueue”, caso a condição seja atendida é exibido uma mensagem informando que o mesmo já está no estacionamento, caso contrário, o mesmo é enfeitado na fila.

Para remover o veículo, a placa também é passada como parâmetro para a função “searchQueue”, porem desta vez se o carro é encontrado o mesmo é passado como parâmetro para a função “maneuver” que retornar o carro removido e posteriormente é exibido os dados do mesmo.

A função “maneuver” recebe o veículo que vai sair da fila e vai removendo os elementos da mesma e verificando se o elemento é igual ao que vai ser removido, caso a condição seja atendida, a quantidade de carros vai receber o valor do contador da quantidade de carros que saíram, e atribuído “1” a variável - flag, “resut” e é retornado por referência os dados do carro, caso contrário, é verificado por meio do flag “resut”, se o carro ainda não foi encontrado, caso satisfeita a condição, é incrementado em mais um o número de vezes que o mesmo saiu e o carro que saiu é enfileirado na fila reserva, por fim, todos os carros da fila reserva voltam para a original.

d. Exercício 02

O exercício dois segue a mesma lógica do primeiro, mudando apenas o tipo de fila, de estática para dinâmica.

e. Exercício 03

No exercício três, é solicitado que seja desenvolvido um programa - em c, que após dado uma expressão matemática no modo in-fixa, então seja verificado se a mesma é válida e logo em seguida, com o auxílio de pilha estática, seja convertida para o modo pós-fixa.

A função nomeada de “checkExp” foi desenvolvida com a finalidade de validar a mesma. XXXXX

Já a função “infixToPostfix” faz a conversão da expressão para pós-fixa. A mesma recebe um endereço para uma pilha e uma string que representa a expressão in-fixa.

A função quebra toda a string em sub-strings onde houver espaços enquanto ela for diferente de null, e em seguida é executado o algoritmo de conversão. Inicialmente, é verificado se a primeira sub-string é um operando e é adicionado à pilha resultante, caso não seja, é verificado se a mesma é uma abertura de parênteses “(“, se sim, a mesma é adicionada na pilha auxiliar, caso contrário, é verificado se é um fecha parêntese “)”, caso seja, os elementos da pilha auxiliar serão removidos enquanto a fila for diferente de vazia ou o elemento do top for diferente da abertura de uma parêntese, e adicionado na fila resultante, caso não entre em nenhuma das condições anteriores, é quando um operador é encontrado e é desempenhado da auxiliar e empilhado na resultante enquanto a fila auxiliar não esteja vazia e a precedência da substring seja menor o igual a do topo da pilha resultante, caso ainda sobre substrings, a mesa é empilhada na pilha auxiliar. E por fim, todos os operadores da fila auxiliar são desligados da fila auxiliar e empilhados na pilha resultante.

f. Exercício 04

O exercício quatro segue a mesma lógica do terceiro, mudando apenas o tipo de pilha, de estática para dinâmica.

g. Exercício 05

O exercício cinco é solicitado que seja desenvolvido um programa em c que simule a partir de três filas de processos um algoritmo escalonador de processos.

Inicialmente é solicitado os dados do processo, bem como seu número, seu tempo de execução - em segundos, e sua prioridade, em seguida os dados são juntos com as três filas de prioridade existentes para a função “enqueuePrioty”, que vai tratar a sua prioridade em enfileirar na sua respectiva fila [de prioridade].

A opção dois do menu chama o escalonador, que antes verificar se as três filas estão vazias, caso estejam é apresentado a mensagem informativa, já caso alguma esteja preenchida com pelo menos um processo, é executado o algoritmo.

A função “scheduler” trata de receber três filas e inicialmente verifica se a fila 1 não está vazia, caso a condição seja atendida, a fila 1 e 2 é passada para a função “scheduler2”, caso contrário, é verificado a mesma condição com a fila 2, e passada fila 2 e 3 para função “scheduler2”, e caso contrário é verificado com a fila 3, caso seja atendida, é passada a fila 3 e para a função “scheduler2”, e já caso nenhuma das condições seja atendida é exibida uma mensagem informativa que as três filas estão vazias.

A função nomeada de “scheduler2” recebe quatro parâmetros dois endereços de filas, um endereço para um tdado, e uma inteiro nomeado de “tf” que representa qual regra vai ser aplicada, no início é desenfileirar um elemento da fila 1, e caso o “tf” seja “0” a regra vai se aplicar para a fila 1 e 2, e chama a função “useProcessor” e em seguida é verificado se o tempo de processo é diferente de zero e caso seja atendida é verificada a se a quantidade de vezes que o processo foi executado é diferente de 4, caso atendida a quantidade de vezes que o processo passou é zerada e o processo enfileirado na fila 2, caso contrário enfileirado novamente na fila 1, já caso o “tf” seja igual a “1”, ou seja , é valído para fila 3, o processo é passado para a função “useProcessor” e em seguida verificado se o processo é diferente de zero, caso seja atendida é enfileirado novamente na fila.

Para pegar o próximo elemento a usar o processador, a função “getNextUseProcessor” foi desenvolvida, a mesma recebe as três filas e retorna por referência um processo, caso tenha algum. Inicialmente a função verifica se a primeira fila não está vazia , caso atendida, ele retorna o primeiro elemento da fila, caso contrário a mesma segue verificando as

demais e se não entrar em nenhuma, é exibida uma mensagem informando que as mesmas estão vazias.

Para retornar a quantidade de elementos da fila, é verificado apenas se a mesma não estiver vazia e em seguida retornado o tamanho que encontra-se na estrutura da fila.

Já para calcular o tempo de uma determinada fila, a função "missingHowLong" foi codificada, a mesma percorre todos os elementos somando enquanto todo o tempo de processamento for diferente de zero e a quantidade vezes for diferente de 2, isso para a fila primeira e segunda, para a terceira fila, é a fila é percorrida enquanto o tempo de processamento for diferente de zero e incrementado em mais o resultado.

Para buscar o tempo de processamento para chegar até um determinado processo, a função "processingTimeForProcess" foi implementada, se a fila for 1 ou 2, a função percorre a fila verificando se o número é igual ao informado pelo usuário e a variável - flag, "resul1" recebe 1, e em seguida o tempo do processo é somada a variável na variável "result2" enquanto o processo não foi encontrado. Caso a função seja a terceira, ela também percorre toda a fila verificando se o processo é maior que quatro, é incrementado em mais quatro, caso contrário é incrementado o tempo do processo.

Já para calcular o tempo de todos os processo, a função "missingHowLong2", a mesma percorre todas as filas incrementando os tempos dos processos.

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 primeira questão, solicita que seja lido uma matriz de strings para então realizar as seguintes opções: (0) sair; (1) - estacionamento; e (2) - mostrar carros.

ENTRADA	**SAÍDA**
1 (E) - Placas: abc123; qwe123; xyz123; hjr123; test123; pia123; zeb123; e ovo123.	# SUCCESS!
1 (S) - "ovo123	CAR | Placa: ovo123 - Mov carro: 0 - Qtd carros mov: 7
1 (E) - "qwe123"	[!] - Já existe!

1 (S) - “abc123”	CAR | Placa: abc123 - Mov carro: 1 - Qtd carros mov: 0
1 (S) - “abc123”	[!] - Carro não encontrado!
1 (E) - “abc123”	# SUCCESS!
1 (S) - “hjr123”	CAR | Placa: hjr123 - Mov carro: 1 - Qtd carros mov: 2
1 (S) - “xyz123”	CAR | Placa: xyz123 - Mov carro: 2 - Qtd carros mov: 1

O exercício número dois é similar ao primeiro, mudando apenas o tipo de fila, de estática para dinâmica.

ENTRADA	**SAÍDA**
1 (E) - Placas: abc123; qwe123; xyz123; hjr123; test123; pia123; zeb123; e ovo123.	# SUCCESS!
1 (E) - “qwe123”	[!] - Já existe!
1 (S) - “qwe123”	Placa: qwe123 - Mov carro: 0 - Qtd carros mov: 1
1 (S) - “abc123”	CAR | Placa: abc123 - Mov carro: 1 - Qtd carros mov: 0
1 (S) - “ovo123”	CAR | Placa: ovo123 - Mov carro: 0 - Qtd carros mov: 5
1 (S) - “zeb123”	CAR | Placa: zeb123 - Mov carro: 1 - Qtd carros mov: 4
1 (E) - “zeb123”	# SUCCESS
1 (S) - “zeb123”	CAR | Placa: zeb123 - Mov carro: 0 - Qtd carros mov: 4

No exercício de número três (03), é solicitado que desenvolva um algoritmo em C que dado uma notação no modo in-fixa retorne uma no modo pós-fixa.

ENTRADA	**SAÍDA**

1 + 1	1 1 +
1 + (2 * 100)	[!] Expressão inválida!
2 + 100 * 2	2 100 2 * +
2 + (100 * 2)	2 100 2 * +
1 * 180 / 3.14	1 180 3.14 / *

O exercício número quatro é similar ao terceiro, mudando apenas o tipo de pilha, de estática para dinâmica.

ENTRADA	**SAÍDA**
1 * 3.14 / 180	1 3.14 180 / *
15 / (3 * 14)	15 3 14 * /
14 / 100 * (2 - 3.14)	14 100 2 3.14 - * /

O último exercício proposto, o quinto, solicita que seja codificado um programa para ler processos e: (1) inserir em uma fila; (2) Escalonador (3) todos; (4) próximo; (5) quantidade de processos; (6) tempo falta de uma fila; (7) tempo para um processo; e (8) tempo total dos processos.

ENTRADA	**SAÍDA**
1 - ID = 1, TEMPO = 3, PRIORIDADE 1; ID = 2, TEMPO = 2, PRIORIDADE 1; ID = 3, TEMPO = 1, PRIORIDADE 1; ID = 6, TEMPO = 10, PRIORIDADE 1; ID = 4, TEMPO = 4, PRIORIDADE 2; e ID = 5, TEMPO = 5, PRIORIDADE 3.	
2	ID = 1 | TEMPO = 2 | PRIORIDADE = 1 | COUNT = 1
3	ID = 2, TEMPO = 2, PRIORIDADE 1; ID = 3, TEMPO = 1, PRIORIDADE 1; ID = 6, TEMPO = 10, PRIORIDADE 1; ID = 4, TEMPO = 4, PRIORIDADE 2; ID = 5, TEMPO = 5, PRIORIDADE 3; ID = 1, TEMPO = 3, PRIORIDADE 1;
4	ID = 2 | TEMPO = 2 | PRIORIDADE = 1

5 - 1	# Quantidade de processos da f1 = 4
5 - 3	# Quantidade de processos da f1 = 1
2	ID = 2 | TEMPO = 1 | PRIORIDADE = 1 | COUNT = 1
6 - 2	# Tempo falta da f2 = 2
6 - 3	# Tempo falta da f3 = 5
7 - 7	[!] Não encontrado!
7 - 1	# TEMPO QUE FALTA: 2
7 - 5	# TEMPO QUE FALTA: 10
8	# TEMPO = 23

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, exige tempo e acima de tudo muita dedicação.