Documentação para o Trabalho Prático de AEDS2

Nome: Martinelle Araujo dos Santos

Turma: TW

Matrícula: 2017079442

Prof. Adriano Alonso Veloso

**1. Introdução**

Um dos problemas cotidianos do ICEX são as grandes filas na cantina. Como a quantidade de bandeja e talheres é limitada a falta desses recursos pode ser um agravante para o atraso no atendimento.

O objetivo deste trabalho é fornecer o tempo médio que os usuários levam para completar seu atendimento (entrada na fila até terminar de servir os alimentos) e propor uma configuração que reduza esse tempo. Espera-se com isso praticar os conceitos de Tipos Abstratos de Dados (filas e pilhas) usando-os para simular o processo de atendimento que ocorre no mundo real.

**2. Implementação**

**Estrutura de Dados**

Para a implementação do trabalho foram criados os Tipos Abstratos de Dados Fila e Pilha com as seguintes estruturas:

*TPilha*

A estrutura consiste basicamente de um inteiro Topo (definido como Apontador) que armazena a posição do último elemento inserido (consequentemente o tamanho da pilha). Como os elementos são bandejas não há necessidade de se armazenar mais informações, além da altura da pilha. Analogamente ao conceito de pilhas que usamos no dia a dia os dados sao inseridos e retirados no topo. Como o tamanho da pilha é predefinido não há necessidade de uma implementação dinâmica

*TFItem*

Um Item de uma Fila que contém uma Chave (int), usada para verificação do numero de pessoas atendidas.

*TCelulaF*

Célula de uma Fila que consiste um TFItem e um Apontador para a próxima posição da fila.

*TFila*

Estrutura que será usada para simular as filas reais da cantina. Como não há limite para o tamanho é implementada dinamicamente utilizando apontadores. Consiste em um 2 Apontadores para Células (Frente e Tras). Onde Frente aponta para célula cabeça da fila (uma célula ‘’vazia’’ em que o apontador aponta para primeira posição da fila).

**Funções e Procedimentos**

O TAD Pilha criado possui as seguintes funções:

void FPVazia(TPilha \*Pilha): Recebe um ponteiro para TPilha e monta uma Pilha com 30 elementos.

int Vazia(TPilha Pilha): Recebe uma TPilha e retorna se ela está vazia (1) ou não (0).

void Empilha10(TPilha \*Pilha); Recebe um ponteiro para TPilha e empilha 10 itens na Pilha, se com isso a quantidade de itens ultrapassar o máximo que a Pilha pode ter, ela passa a ter a quantidade máxima.

void Desempilha(TPilha \*Pilha): Recebe uma TPilha e desempilha 1 item (do Topo).

int Tamanho(TPilha Pilha): Recebe uma TPilha e retorna o tamanho dela.

O TAD Fila criado possui as seguintes funções:

void FFVazia(TFila \*Fila): Recebe um ponteiro para TFila e monta uma Fila vazia.

int VaziaF(TFila Fila): Recebe uma TFila e retorna se ela está vazia (1) ou não (0).

void Enfileira(TFItem x, TFila \*Fila): Recebe um TFItem e um ponteiro para TFila e coloca esse item na ultima posição da fila.

void Enfileira2(TFila \*ficha, int inicio): Recebe um TFItem e um ponteiro para TFila e coloca esse item na ultima posição da fila.

void Desenfileira(TFila \*Fila): Desenfileira o item da primeira posição da fila.

Além das funções e procedimentos dos TADs foram implementadas as seguintes funções e procedimentos:

void DesenfileiraEnfileira(TFila \*FilaS, TFila \*FilaE): Recebe um ponteiro para Fila em que sairá um elemento e outro para Fila em que esse elemento que saiu da Fila anterior irá entrar e realiza esse procedimento, ou seja o elemento sai de uma fila e entra em outra.

int PegarBandeja(TPilha \*pilha\_bandejas, TFila \*bet): Recebe um ponteiro para TPilha e um para TFila, verifica se há elementos na pilha e na fila, se houver nos dois, desenfileira um elemento da fila e um da pilha.

**Programa Principal**

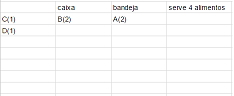
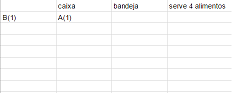
O programa principal declara e cria 1 fila vazia fichas, 1 fila para bandejas e talheres e 1 pilha de bandejas, além das variáveis (int)inicio (marcação do tempo que vai de 0 até 4hrs, ou seja, 240 min, (int)atraso que armazenará o atraso gerado pela falta de bandejas na pilha de bandejas, (int)tempo\_gasto que armazenará a soma do tempo gasto de todas as pessoas que serão atendidas, (int)atendidos que armazenará o total de pessoas atendidas e (int)cte que auxiliará no cálculo do tempo gasto por cada pessoa.

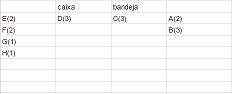
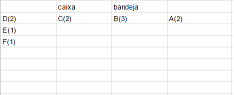
Logo após a inicialização das variáveis o programa transforma o tempo de horas para minutos e retira 4 unidades de tempo antes de começar as iterações, já que o código omite a parte do processo depois que os alunos pegam a bandeja e os talheres (servir os 4 tipos de alimento), já que no problema proposto não há possibilidade de atraso nessa etapa do atendimento, logo, ao fim das n iterações na verdade as 4 últimas pessoas que seriam consideradas ‘’atendidas’’ pelo código na verdade ainda estariam se servindo.

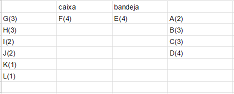
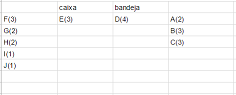
A próxima etapa do código é o loop onde cada iteração descreve 1 intervalo de tempo equivalente a 1 minuto. Como no enunciado do problema foi informado que a cada 12 min 10 bandejas são repostas na pilha de bandejas, a primeira etapa consiste em verificar se a variável início (que corresponde ao tempo naquele instante) tem resto 0 na divisão por 12, se for verdadeiro, chama-se a função Empilha10() que adiciona mais 10 bandejas na pilha de bandejas. No enunciado também foi informado que a cada intervalo de tempo 2 pessoas chegam na fila de fichas, logo chama-se a função Enfileira2(), que enfileira 2 pessoas na fila de fichas em cada iteração. chama-se a função PegarBandeja() que verifica se a pilha de bandejas não está vazia e se há alguma pessoa na fila para pegar bandejas, se sim, realiza a ação pegar bandeja e soma ao tempo gasto total por todas as pessoas (tempo\_gasto) o tempo atual (inicio) + cte + o atraso e logo após, adiciona +1 as pessoas atendidas. Se não houver bandejas na pilha e existir alguém na fila para pegar bandejas e talheres incrementa na variável atraso, já que o atraso se propagará para todo o resto da fila daquela posição em diante. Em cada intervalo de tempo a função DesenfileiraEnfileira faz com que uma pessoa compre a ficha e vá para fila de bandejas e talheres. Logo após, a cada minuto par o número cte é decrementado em 1 unidade para manter o cálculo do tempo gasto por cada pessoa correto. Ao fim, os resultados obtidos sao exibidos na tela.

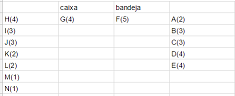
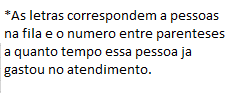
**Implementação e Detalhes Técnicos**

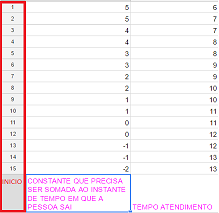
O código está dividido em seis arquivos: tp\_fila.c, tp\_fila.h, tp\_pilha.c e tp\_pilha.h que implementam os Tipos Abstratos de Dados fila e pilha, o arquivo functions.c que implementa outras funções que não sao TADS, enquanto o arquivo main.c implementa o programa principal. O valor especificado para a constante MAX foi 30, de forma a atender as especificações do programa principal. Para simplificar a execução, encontrei um padrão entre o instante em que a pessoa entra na fila e o tempo que ela demoraria para realizar o processo se não houvesse nenhuma interrupção que será sempre mantido, pois em todos os intervalos de tempo 2 novas pessoas entram na fila e necessita ser corrigido apenas se há atraso causado pela falta de bandejas na pilha de bandejas, o padrão é mostrado nas figuras abaixo. Como dito anteriormente, também para simplificação do código foram omitidas as etapas em que a pessoa iria se servir de cada alimento, já que no cenário descrito não há possibilidade de atraso nessas etapas. O compilador utilizado foi o MinGW(GCC), por meio do plugin linter na IDE Atom, rodando no sistema operacional Windows 10.









**3. Análise de Complexidade**

void FPVazia(TPilha \*Pilha)

O(1). As instruções são executadas um número fixo de vezes.

int Vazia(TPilha Pilha)

O(1). As instruções são executadas um número fixo de vezes.

void Empilha10(TPilha \*Pilha)

O(1). As instruções são executadas um número fixo de vezes.

void Desempilha(TPilha \*Pilha)

O(1). As instruções são executadas um número fixo de vezes.

int Tamanho(TPilha Pilha)

O(1). As instruções são executadas um número fixo de vezes.

void FFVazia(TFila \*Fila)

O(1). As instruções são executadas um número fixo de vezes.

int VaziaF(TFila Fila)

O(1). As instruções são executadas um número fixo de vezes.

void Enfileira(TFItem x, TFila \*Fila)

O(1). As instruções são executadas um número fixo de vezes.

void Enfileira2(TFila \*ficha, int inicio)

O(1). As instruções são executadas um número fixo de vezes.

void Desenfileira(TFila \*Fila)

O(1). As instruções são executadas um número fixo de vezes.

void DesenfileiraEnfileira(TFila \*FilaS, TFila \*FilaE)

O(1). As instruções são executadas um número fixo de vezes.

int PegarBandeja(TPilha \*pilha\_bandejas, TFila \*bet)

O(1). As instruções são executadas um número fixo de vezes.

Programa Principal

O(n). Loop tem n iterações em que são chamadas apenas funções O(1) e cálculos, verificações simples e impressão na tela que tem ordem de complexidade constante.

**4. Testes**

Além dos testes com a configuração padrão, foram realizados os seguintes testes com suas respectivas configurações (também no intervalo de tempo de 4 horas):

//Padrão

//1 fila de fichas

//1 fila de bandejas e talheres

//1 pilhas de bandejas

Resultado: ATENDIDOS 218 pessoas. MÉDIA TEMPO GASTO POR PESSOA 65 min

//Alteração 1

//2 caixas

//1 fila de fichas

//1 fila de bandejas e talheres

//1 pilha de bandeja com 40

Resultado: ATENDIDOS 218 pessoas. MÉDIA TEMPO GASTO POR PESSOA 65 min

//Alteração 2

//1 filas de fichas

//1 fila de bandejas e talheres

//2 pilhas de bandejas

Resultado: ATENDIDOS 235. MÉDIA TEMPO GASTO POR PESSOA 65 min.

//Alteração 3

//2 caixas

//1 filas de fichas

//1 fila de bandejas e talheres

//2 pilhas de bandejas

Resultado: ATENDIDOS 235. MÉDIA TEMPO GASTO POR PESSOA 65 min.

//Alteração 4

//2 filas de fichas

//1 fila de bandejas e talheres

//2 pilhas de bandejas

Resultado: ATENDIDOS 235. MÉDIA TEMPO GASTO POR PESSOA 65 min.

//Alteração 5

//1 fila de ficha

//2 filas de bandejas

//1 pilha de bandeja

Resultado: ATENDIDOS 235 pessoas. MÉDIA TEMPO GASTO POR PESSOA 68 min.

**5. Conclusão**

O teste da configuração solicitada teve por resultado 218 pessoas atendidas com uma média de duração de 65 min do começo ao fim do atendimento.

Por meio das comparações dos resultados obtidos nas diferentes configurações de atendimento conclui-se que das configurações testadas a melhor média de tempo obtida atendendo ao maior número de pessoas possível seria de 235 pessoas atendidas com tempo médio de atendimento de 65 min. Resultado que pode ser obtido com as alterações 2,3 e 4.

A implementação do trabalho transcorreu sem maiores problemas e os resultados

ficaram dentro do esperado. A principal dificuldade encontrada foi a falta de dados para comparação e averiguação dos resultados encontrados.

**Referências**

[1] Ziviani, N., Projeto de Algoritmos com Implementações em Pascal e C, 2ª Edição,

Editora Thomson, 2004.

Anexos

Listagem dos programas:

* tp\_fila.c
* tp\_fila.h
* tp\_pilha.c
* tp\_pilha.h
* function.c
* function.h
* main.c