TP1 - Cantina

Leandro Balmant de Paula Souza

15 de maio de 2017

1 Introdução

Este trabalho tem como o objetivo de simular o funcionamento de uma cantina em um intervalo de 4 horas, utilizando estruturas de dados para a criação de filas de caixa, filas de bandeja e pilhas de prato. A problemática principal é calcular a média de tempo do atendimento durante 240 minutos. A cada minuto, chega 2 clientes que são encaminhados para a fila do caixa. Sendo assim, quando o tempo atinge as 4 horas, a fila do caixa é fechada, porém todos que estão nela ainda serão atendidos e seus tempos contabilizados na média geral de atendimento.

Por meio dessa simulação, deve ser possível alterar a quantidade das filas e pilhas, com o objetivo de alcançar qual a quantidade necessária para otimizar o tempo de atendimento dos clientes da cantina nesse intervalo de tempo.

2 Implementação

2.1 Estrutura de Dados

Abaixo serão explicadas as estruturas utilizadas. É importante saber que ambas foram utilizadas como arranjos, por exemplo, foi criado um vetor de filas que são alocadas dinamicamente. O objetivo dessa implementação é a possibilidade de alterar a quantidade de filas e pilhas de acordo com o usuário.

2.1.1 TAD Fila

Esta estrutura é do Tipo Fila, criada para ser utilizada representado a fila da Bandeja e fila do caixa. Basicamente, sua funcionalidade é a primeira posição ser a primeira a ser retirada e a última posição ser a última. Além disso, esse TAD também armazena o tempo que cada cliente entrou na fila.

2.1.2 TAD Pilha

Esta estrutura do tipo Pilha representa a pilha de pratos e tem a função de retirar somente quem estiver no topo. Ou seja, o primeiro que entrar será o último a sair e o último que entrar será o primeiro a sair. A pilha é limitada a 30 posições, que no caso são os pratos. Além de ser reposta com 10 pratos a cada 12 minutos.

2.2 Funções e Procedimentos

Serão citadas nesse tópico as implementações principais para o programa.

2.2.1 TAD Fila

A função int EscolheFila(TFila **Fila, int nFila) recebe como parâmetro o arranjo de Filas e o numero de total de filas.

O objetivo dessa função é retornar o índice da fila que tem menos clientes.

Algoritmo 1 Fila.c

```
function INT ESCOLHEFILAMAIOR(TFila **Fila, int nFila)
    int i,a
    int maior ← −1;
    if nFila = 1 then
        return 0
    for i ← 0; i < nFila; i++ do
        if (*Fila)[i].Tamanho > maior then
            maior ← (∗Fila)[i].Tamanho
            a ← i
    return a
```

A função retorna um inteiro, que é o índice do vetor de filas que será utilizado. Primeiro, verifica se o número de filas é igual a 1, sendo assim o índice 0. Caso contrário, será necessário acessar todas as filas do vetor e verificar o tamanho de todas (que já é incrementado quando insere uma posição na fila), retornando o índice do maior vetor.

O algoritmo 2 funciona da mesma forma da EscolheFilaMaior, porém, ela retorna o índice do menor vetor.

O algoritmo 3 faz o arranjo de filas, recebendo um *TFila e o número total de filas, juntamente à função FFVazia, que tem como objetivo a criação de uma fila vazia, ou seja, sem nenhum elemento além dos fundamentais para o funcionamento da estrutura.

Algoritmo 2 Fila.c

```
function int EscolheFila(TFila **Fila, int nFila)
    int i,a;
    int menor ← 100000;
    if nFila = 1 then
        return 0
    for i ← 0; i < nFila; i++ do
        if (*Fila)[i].Tamanho < menor then
            menor ← (*Fila)[i].Tamanho
            a ← i;
    return a
```

Algoritmo 3 Fila.c

```
function void FazArranjoFila(TFila *Fila, int nFila)
    int i
    for i ← 0; i < nFila; i++ do
        FFVazia(&Fila[i])
```

2.2.2 TAD Pilha

Em relação ao TAD Pilha, foi criado também uma função para escolher o índice, porém, como os pratos são empilhados impreterivelmente a cada 12 minutos, não é necessário saber em qual pilha de pratos está menor para empilhar, pois são empilhados em todas as pilhas. Agora para a retirada dos pratos, foi feito a função que retorna o índice da pilha que tem mais pratos.

Algoritmo 4 Pilha.c

```
function int EscolhePilha(PilhaPratos **Pilha, int nPilha)
    int i,a;
    int maior ← −1;
    if nPilha = 1 then
        return 0
    for i ← 0; i < nPilha; i++ do
        if (*Pilha)[i].Tamanho > maior then
            maior ← (*Pilha)[i].Tamanho
            a ← i;
    return a
```

Existe também a função FazArranjoPilha e seu funcionamento é idêntico ao FazArranjoFila.

2.2.3 Programa principal

O main.c é basicamente divido em duas partes: a primeira parte é a leitura de dados, criações de todas as estruturas necessárias e a segunda parte é o andamento da cantina. Inicialmente, é recebido por linha de comando a quantidade de filas de caixa, filas de bandeja e pilhas de pratos que serão utilizados na cantina. Após isso, é alocado dinamicamente dois ponteiros para TFila e um para PilhaPratos, que serão os arranjos dessas estruturas. Um ponto importante nessa parte é a criação da variável int tempo, que será incrementada no loop principal do jogo. Porém, nessa primeira etapa, é considerada que o tempo já está em 1 e a pessoa que estava na fila do caixa, já esta na fila da bandeja e o que estava atrás dele, já está no caixa. Tudo isso, porque no tempo 0 não tem ninguém em nenhuma fila, por isso, o atendimento é imediato.

Em relação ao loop principal do programa, que é a simulação da cantina, pode ser divido em 5 condicionais que serão explicados a seguir.

Antes disso, é válido ressaltar que a quantidade de filas de caixa, filas de bandeja e pilhas de prato são recebidas do usuário utilizando argc e argv. Por ex.: 1 2 3, para 1 fila de caixa, 2 filas de bandeja e 3 pilhas de prato.

- Fila do caixa e fila da baneja

Algoritmo 5 Main.c

```
if !Vazia(filaCaixa[EscolheFilaMaior(&filaCaixa, nFilaC)] then
    for i ← 0; i <nFilaC; i++ do
        if !Vazia(filaCaixa[i]) then
            Desenfileira(&filaCaixa[EscolheFilaMaior(&filaCaixa, nFilaC)],
&x)
            Enfileira(x,&filaBandeja[EscolheFila(&filaBandeja, nFilaB)])
```

Na condição acima, verifica se a maior fila do caixa está vazia, caso não esteja significa que ainda tem gente para ser atendida, então, um loop irá acessar todas as filas, desenfileirar uma pessoa e enfileira ela na menor fila da bandeja. É necessário acessar todas as filas do caixa, pois caso existam mais de um, as pessoas que estão na mesma posição porém em caixas diferentes, elas serão atendidas no mesmo tempo.

Um exemplo interessante é o caso de duas pessoas chegaremno tempo x e existirem 3 filas do caixa com 1 pessoa no caixa 1 e uma pessoa no caixa 2. A inserção é feita na menor fila, ou seja, no caixa 3. Agora ambas as filas possuem a mesma quantidade de pessoas, sendo assim, o cliente será redirecionado ao caixa 1.

- Incremento na fila do caixa

Algoritmo 6 Main.c

```
if tempo<tempoTotal then
    for i ← 0;i<Pessoas; i++ do
        x.Tempo = tempo
        Enfileira(x,&filaCaixa[EscolheFila(&filaCaixa, nFilaC)])
        countPessoas++
```

A condição do algoritmo a seguir, representa a incrementação das pessoas na(s) fila(s) do caixa, definida a quantidade de 2 pessoas por minuto. Além disso, é armazenado no TAD Fila da posição do cliente, o tempo inicial que de seu atendimento. Ele é enfileirado na fila de menor tamanho e é o contador de pessoas é incrementado.

É importante ressaltar que esse processo só acontece até os 240 minutos da simulação, ou seja, até esse tempo, a cada minuto entrará 2 pessoas na fila do caixa.

- Pilha de Pratos

Algoritmo 7 Main.c

```
if !PilhaVazia(pilha[EscolhePilha(&pilha, nPilha)] then
    for i ← 0; i <nFilaB; i++ do
        if !Vazia(filaBandeja[i] && !PilhaVazia(pilha[EscolhePilha(&pilha,
nPilha) then
            Desenfileira(&filaBandeja[EscolheFilaMaior(&filaBandeja, nFi-
laB)], &x)
            mediaInd = (tempo - x.Tempo) + 5
            media += mediaInd
            DesempilhaPratos(&pilha[EscolhePilha(&pilha, nPilha)], &y)
            countFinal++
```

No algoritmo acima, a primeira condição verifica se existem pratos disponíveis para os clientes, pois caso não tenha, será necessário aguardar a próxima reposição. Se existirem pratos, é preciso acessar todas as filas da bandeja, desinfireilar, resgatar o tempo inicial e diminuir com o tempo atual do fim do atendimento e enviar esse resultado para a variável media, que armazena todos esses tempos. O 5 que é somado na mediaInd são os tempos constantes, ou seja, 1 minuto pra pegar a bandeja e 4 para servir as guarnições. Por fim, desempilha um prato

na pilha e contabiliza mais um que terminou o atendimento na variável countFinal.

- Reposição dos pratos

Algoritmo 8 Main.c

```
if tempo%12==0 then
    for j ← 0; j < nPilha; j++ do
        for i ← 0; i < 10; i++ do
            if pilha[j].Tamanho < 29 then
                EmpilhaPratos(y, &pilha[j])
```

Se o tempo for divido por 12 e com resto zero significa que está no momento de repor os pratos da bandeja. Para isso, é preciso acessar todas as pilhas de pratos caso exista mais do que uma e adicionar 10 pratos se possível, pois existe o limite de 30 pratos na pilha.

Algoritmo 9 Main.c

```
if Vazia(filaBandeja[EscolheFilaMaior(&filaBandeja, nFilaB)]) && Vazia(filaCaixa[EscolheFilaMaior(&filaCaixa, nFilaC)] then
    fim ← 0
```

Por fim, essa é a condição fundamental do programa. A variável fim recebendo 0, o loop é finalizado. Porém, antes é verificado se a maior fila da bandeja está vazia e se a maior fila do caixa está vazia, caso sendo ambas confirmadas, todos os clientes que chegaram até 240 minutos já foram atendidos.

2.2.4 Compilação

gcc -c fila.c -o fila.o
gcc -c pilha.c -o pilha.o
gcc main.c fila.c pilha.c
a 1 1 1 (exemplo de chamada)

3 Análise de Complexidade

- **int EscolheFila(TFila **Fila, int nFila), int EscolheFilaMaior(TFila **Fila, int nFila) e int EscolhePilha(PilhaPratos **Pilha, int nPilha):** Essas funções são muito idênticas e possuem complexidade

O(n) pois além de fazerem algumas atribuições O(1), há um loop que rodará n-1 vezes.

- **void FazArranjoPilha(PilhaPratos *Pilha, int nPilha) e void FazArranjoFila(TFila *Fila, int nFila):** São funçoes bem simples e também com o mesmo objetivo porém para estruturas diferentes. Nelas existem atribuiçoes O(1) mas há um loop que será inicializado n-1 vezes, por isso a complexidade é O(n).

- **programa principal:** Essa parte como dito anteriormente foi divida em duas. A primeira possui complexidade O(n) pelo fato da criação dos vetores de fila e pilha. Em relação a segunda parte, que é o loop principal, aumenta a quantidade de variáveis e de loops. Sendo o melhor caso a situação que os pratos não irão acabar, o while principal depende do n, que é o numero de pessoas que entrão na cantina, por isso, é sua complexidade é O(n). Dentro desse loop, existem outros no caso dependem da quantidade de fila do caixa,que é representada por O(c), filas da bandeja O(b) e pilhas de prato O(p). Em resumo, a complexidade é desse caso é dada por O(n)(O(c)+O(b)+O(p)).

  As demais funçoes são de caráter constante, ou seja, O(1), por fazerem somente atribuições e alocações.

4 Testes

Tabela 1: Tempo de atendimento

Fila Caixa	Fila Bandeja	Pilha Pratos	Media de Atendimento
1	1	1	152 minutos
2	1	1	152 minutos
1	2	1	150 minutos
1	1	2	126 minutos
2	2	1	147 minutos
2	2	2	11 minutos
3	3	2	10 minutos
3	3	3	6 minutos
2	2	3	6 minutos

5 Conclusão

O desenvolvimento do trabalho, em geral, não houve complicações. Foi possível alcançar as metas iniciais que eram simular o funcionamento de uma cantina, podendo alterar a quantidade de filas de caixa, filas de bandeja e pilhas de prato.

Além disso, a partir da comparação dos resultados no tópico de Testes, é possível concluir que a pilha de pratos é o fator fundamental para um funcionamento fluído das filas. Sendo assim, um resultado otimizado na simulação é a utilização de 2 filas de caixa, 2 filas de bandeja e 3 pilhas de pratos, atingindo assim o tempo médio de atendimento de 6 minutos, que é o melhor possível no contexto da cantina.