Universidade Federal de Minas Gerais - UFMG
Departamento de Ciência da Computação
DCC004 - Algoritmos e Estrutura de Dados II

TP01 - Filas, Pilhas e Complexidade

Turma TW
Aluno: Matheus Henrique Antunes Lima
Professor: Adriano Alonso Veloso

Maio
2018

Universidade Federal de Minas Gerais - UFMG
Departamento de Ciência da Computação
DCC004 - Algoritmos e Estrutura de Dados II

TP01 - Filas, Pilhas e Complexidade

Documentação do Trabalho Prático 01 da disciplina
DCC004 - Algoritmos e Estrutura de Dados II

Turma TW

Aluno: Matheus Henrique Antunes Lima

Professor: Adriano Alonso Veloso

Maio
2018

Conteúdo

1 Introdução

Todos os dias, muitas pessoas almoçam no restaurante da cantina do ICEx. Um dos problemas enfrentados pelos usuários, no entanto, são as grandes filas enfrentadas para pagar pela comida e se servir. Além disso, a quantidade de bandejas e pratos disponíveis é limitada, podendo ocorrer falta desses recursos dependendo da situação. A pessoa que gerencia a cantina quer contratar um sistema que melhore a distribuição de tarefas dentro da cantina.

Neste trabalho de consultoria foi feito então, a simulação do serviço da cantina no período de 4 horas afim de, com auxilio das estruturas de dados fila e pilha, analisar as melhores configurações de serviço para a otimização do tempo e quantidade de atendimentos efetuados.

2 Desenvolvimento

Como requisitado no enunciado do trabalho prático, foram implementadas, na linguagem C, as estruturas de dados fila e pilhas para simular a rotina de serviço da cantina. A estrutura de atendimento do serviço é descrito com uma fila para o caixa, uma fila para que se pegue uma bandeja de uma pilha e uma fila para se servir cada um das quatro opções de alimentos.

Fica claro pelo nome das estruturas que as filas do caixa, das bandejas e a das comidas foram implementadas utilizando a estrutura de dados fila. Nestas, o primeiro cliente a chegar é o primeiro a sair e, portanto, foi a mais indicada para estes casos. Também obviamente, a pilha de bandejas foi implementada utilizando a estrutura de dados pilha. Neste caso as bandejas são empilhadas e a ultima da pilha é a primeira a ser utilizada.

Cada uma das estruturas, fila e pilha, possuem uma TAD com operações e atributos específicos. Na TAD Fila, uma fila *Queue*, possui células individuais, *qCell*, que representa cada cliente e são compostas por:

- um *ticket*, que representa a ficha de atendimento ou a senha;
- um *queueTime*, que armazena o tempo em que a cliente está numa determinada fila (caixa, bandeja ou comida);
- um *serviceTime*, que armazena o tempo total do serviço, desde quando o cliente chega na fila do caixa até quando ele serve o ultimo alimento e finaliza o atendimento;
- e um ponteiro *next*, que aponta para o próximo cliente na fila.

As *Queue* possuem dois ponteiros, *first* e *last* que apontam respectivamente para o primeiro e para o último cliente na fila e um inteiro *size* que armazena o tamanho atual da fila. As operações da TAD Fila são:

- *clearQueue*, que inicializa uma fila vazia;
- *emptyQueue*, que verifica se a fila está ou não vazia e retorna o resultado booleano;
- *enQueue*, que insere ao fim da fila um cliente;
- *deQueue*, que remove o primeiro cliente da fila e retorna seu *ticket*;
- *increaseQueueTime*, que percorre toda a fila incrementando o tempo de fila, *queueTime*, de todas as células cliente;
- e *increaseServiceTime*, que percorre toda a fila incrementando o tempo de serviço total, *serviceTime*, de todas as células cliente.

Na TAD Pilha, a pilha *Stack*, também possui células *sCell* que representam cada uma, uma bandeja na pilha. Cada estrutura *sCell* é composta por:

- um *trayNumber*, que representa o número da bandeja. Funciona como uma identificação de cada bandeja;
- e um ponteiro *next*, que aponta para a próxima bandeja na pilha.

A *Stack* possuem também dois ponteiros, *top* e *bottom*, que apontam respectivamente para o topo e para o fundo da pilha além de um inteiro size que armazena o tamanho atual da pilha. Na parte das operações estão:

- *clearStack*, que inicializa uma pilha vazia e define o máximo de bandejas permitidas;
- *emptyStack*, que verifica se a pilha está ou não vazia e retorna o resultado booleano;
- *fullStack*, que verifica se a pilha está ou não cheia e retorna o resultado booleano;
- *push*, que insere ao topo uma bandeja;
- *pop*, que remove a bandeja do topo retornando seu *trayNumber*;
- e *fillStack*, que enche a pilha com n bandejas.

A simulação do serviço consiste então na instanciação das *Queue* referentes as três filas, a do caixa, a para pegar bandejas e a da comida, respectivamente *ticketQueue*,*trayQueue* e *foodQueue*. Também é instanciado uma *Stack*, *trayStack*, que representa a pilha de bandejas. As ações de cada cliente nas filas são repetidas então em um loop que dura 240 unidades de tempo, que são as 4 horas de serviço a serem analisadas.

Para cada situação de análise são definidas, por parâmetro, o número de pilhas e o máximo de bandejas em cada uma delas, o número de caixas de atendimento e o período de tempo em que ocorre a reposição de bandejas na pilha. Os valores padrões foram definidos para o caso inicial proposto. São eles:

- quantidade máxima de bandejas em uma pilha, $max\_stack = 30$;
- quantidade de caixas de atendimento, $cashiers\_number = 1$;
- quantidade de pilhas de bandeja, $stacks\_number = 1$;
- e o período de tempo para a reposição de bandejas, $refill\_time = 1$.

3 Implementação

Cada TAD, Fila e Pilha, foi implementada em módulos individuais. O programa principal, que contém a função *main*, também foi implementado em um arquivo .c individual.

No TAD Fila, os métodos *enQueue* e *deQueue* seguem o padrão para enfileirar e desenfileirar células. A única diferença é que, no método *enQueue* também é passado por parâmetro o tempo de serviço, *serviceTime*, do cliente. Assim é possível carregar este tempo com o cliente quando ele muda de uma fila para outra afim de, no final do atendimento, termos o tempo total gasto.
Foram implementados dois métodos para o controle de tempo dos clientes nas filas. Um *increaseQueueTime*, que percorre toda a fila e incrementa o *queueTime* de cada cliente. Com esse método é possível saber quanto tempo cada cliente está em uma determinada fila e assim determinar quando ele deve sair. Por exemplo, quando o cliente está na fila de alimentos sabe-se que ele tem que se servir das 4 opções disponíveis, logo, ele fica nesta fila pelo menos 4 unidades de tempo. Outro uso é na fila do caixa, pois o cliente gasta pelo menos uma unidade de tempo para entrar na fila e outra unidade para ser atendido. Portanto, neste caso, é verificado se o *queueTime* dele é pelo menos 1.
O outro método, *increaseSeviceTime* também percorre toda a fila e incrementa o *serviceTime* de cada cliente. Com esse atributo é feito o cálculo do tempo médio de atendimento ao final do ciclo de serviço.

No TAD Pilha, os métodos *push* e *pop* também seguem o padrão para empilhar e desempilhar células. Foi implementado o método *fillStack* onde neste a pilha é preenchida com um número n de bandejas, que é passado por parâmetro. Este método é utilizado antes do início do serviço, preenchendo totalmente a pilha de bandejas e também nos períodos de tempo definido para que haja a reposição das bandejas.

No programa principal são instanciados três *Queue*, *ticketQueue*, *trayQueue* e *foodQueue* que representam respectivamente as filas de caixa, bandejas e comidas. Também é instanciado uma *Stack*, *trayStack*, representando a pilha de bandejas. Também são instanciadas variáveis auxiliares para o controle do tempo do ciclo e para a contagem total do tempo de serviço e quantidade de clientes atendidos afim de calcular o tempo de atendimento médio ao final da execução.
Para a variação dos casos de análise, ficou definido que o número de filas não é relevante para o tempo de atendimento médio, por isso são instanciadas apenas três. O que realmente varia este tempo médio são o número de caixas

para atendimento e o de pilhas de bandeja, para que assim duas pessoas consigam ser atendidas simultaneamente. Outro fator para a análise é o tempo para a reposição das bandejas e o número máximo de bandejas na pilha. Essas variáveis citadas são instanciadas por padrão com os valores definidos para o caso inicial e podem ser alteradas com a passagem de parâmetros na chamada da execução.

O ciclo de serviço funciona com um loop de ações que acontecem em uma unidade de tempo. Este é repetido 240 vezes, de acordo com o que foi requisitado pelo problema, simulando assim as 4 horas de serviço. Para evitar que, por exemplo, um cliente saia da fila do caixa e, na mesma unidade de tempo, já pegue uma bandeja, as ações são computadas de trás para frente no ciclo. Começando pela fila de alimentos, para a fila de bandejas e por fim, a fila do caixa.
Para a fila de alimentos, é incrementado inicialmente o *queueTime* para todas as células. Este incremento acontece primeiro nesta fila pois ficou definido que não há gasto de tempo entre pegar uma bandeja e começar a servir o primeiro alimento. Após isso, é verificado se o primeiro da fila já tem um queueTime de pelo menos 4. Caso positivo, isto indica que ele terminou de sevir a 4ª opção de alimento e portanto finalizou seu atendimento. O tempo total do serviço é incrementado para o cálculo final da média de atendimento e ele é removido da fila.
Seguindo para a fila de bandejas, é verificado se há bandejas na pilha. Caso haja verifica-se o *queueTime* do primeiro da fila. Este tempo deve ser pelo menos 1, que indica o tempo gasto entre entrar na fila de bandejas e pegar uma bandeja da pilha. Caso positivo o cliente é sai da fila de bandejas e entra na fila de alimentos. Esta ação é repetida de acordo com o número de pilhas existentes pois a implementação destas pilhas extras foi feita de forma que elas são todas unidas em uma única pilha. Aos que restaram na fila é incrementado o *queueTime*.
Por fim, na fila do caixa, é verificado também *queueTime* do primeiro da fila para que este seja ao menos 1. Isso indica o tempo gasto entre entrar na fila e ser atendido pelo caixa. Caso positivo este é removido da fila do caixa e inserido na fila das bandejas. Esta ação também é repetida de acordo com o número de caixas de atendimento.
Após os movimentos nas filas, dois novos clientes entram na fila do caixa e, caso seja o tempo definido para reposição das bandejas, 10 novas bandejas são empilhadas. O tempo de serviço de todos os clientes é incrementado ao final de todas as ações.

4 Análise

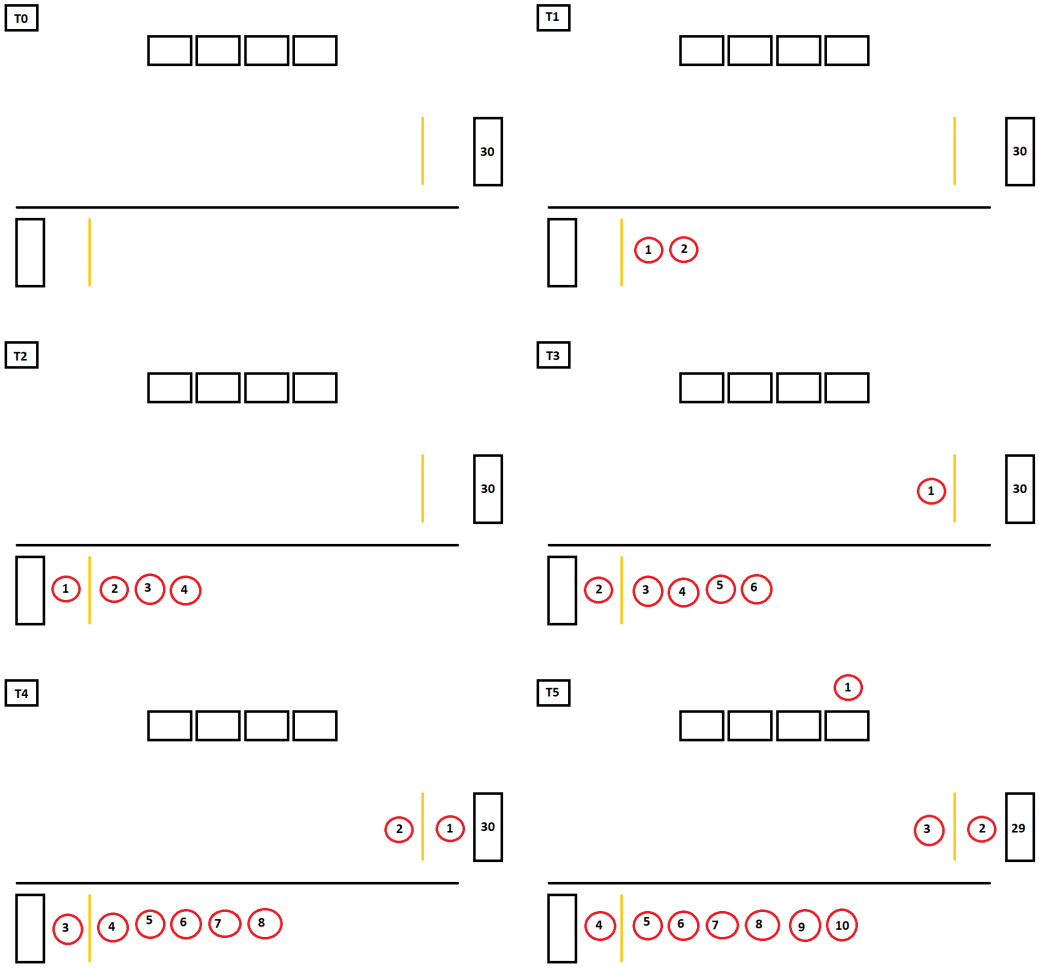
A análise de complexidade do algoritmo ficou bem simples e o mesmo se manteve totalmente linear.
Na instanciação das filas e pilhas, as funções *clearQueue* e *clearStack* tem complexidade $O(1)$ e a função *fillStack* tem complexidade também $O(1$, apenas multiplicado pelo tamanho e quantidade de pilhas definidas mas que são valores constantes.
Durante a execução do ciclo de atendimentos, as funções *deQueue*, *enQueue*, *pop* tem todas complexidade $O(1)$, também multiplicadas de acordo com os parâmetros do caso em teste mas que continuam sendo constantes. As únicas funções com complexidade diferente destas são as para incremento dos queueTime e *serviceTime* das células nas filas. Estas possuem complexidade $O(n)$ e são chamadas durante cada ciclo um total de 6 vezes, o que não altera a complexidade final.
Por fim, um ciclo completo em uma unidada de tempo tem, portanto, complexidade $O(n)$ e, como este é repetido 240 vezes, $240 * O(n) = O(n)$ sendo esta a complexidade final de todo o algoritmo.

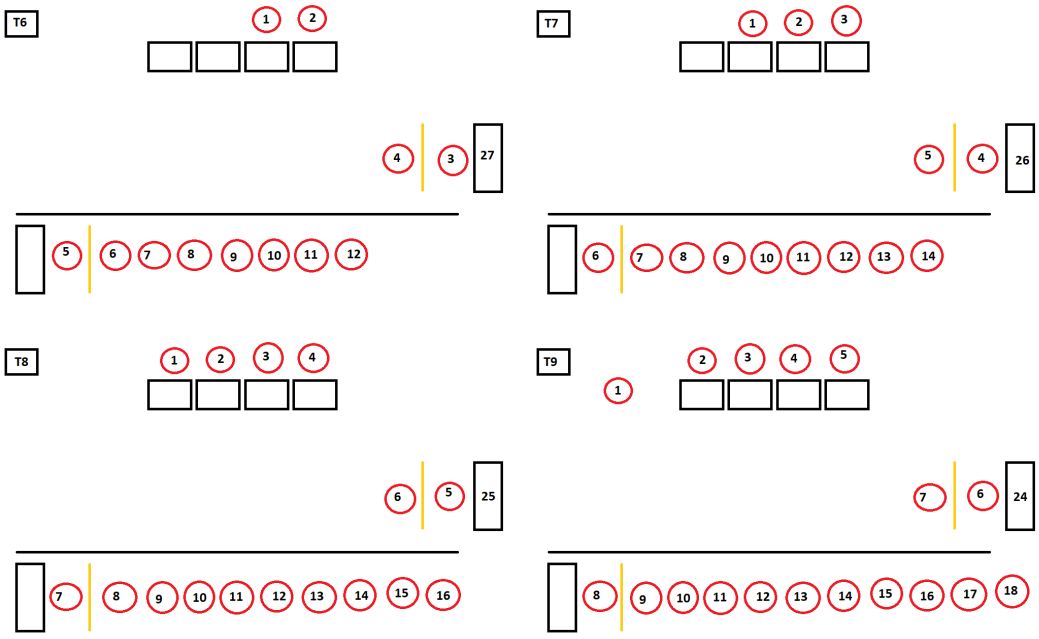
5 Resultados

Para a análise de quais seriam as melhores configurações foram testados alguns casos. Para as configurações definidas no caso inicial os resultados foram:

- Um total de 216 atendimentos finalizados;
- Um tempo médio de atendimento de 65.185 unidades de tempo;

Estas imagens ilustram a movimentação do primeiro cliente a chegar para atendimento até o tempo em que ele finaliza o serviço nestas configurações:





Após a realização de alguns casos de testes foram obtidos os seguintes resultados em relação às variáveis definidas por parâmetro e a quantidade final de atendimentos e o tempo médio:

max_stack	cashiers_number	stacks_number	refill_time	A	T
30	1	1	12	216	65.185
40	1	1	12	226	64.978
50	1	1	12	232	66
200	1	1	12	232	66
30	5	1	12	216	65.185
30	10	1	12	216	65.185
30	1	2	12	232	66
100	5	5	12	232	66
30	1	1	11	232	66

*A: Quantidade de atendimentos finalizados
*T: Tempo médio de cada atendimento

A partir desses dados fica verificado que, apesar de todas as variáveis quando aumentadas levarem, ao que parece ser um limite, de 232 atendimentos e um tempo médio de 66 unidades de tempo, algumas demandam muito mais custo para a gerencia da cantina. Por exemplo, para sairmos de uma pilha de 30 para duas seriam necessárias mais 30 bandejas disponíveis. Ou para sairmos de uma pilha com 30 bandejas para uma com 50 são necessárias mais 20 bandejas disponíveis. Ficou evidente também que, sozinha, o número de

caixas para atendimento não acelera o serviço pois, de nada adianta mais pessoas irem para a fila de bandejas se chegando lá elas não conseguem pegar uma da pilha. Por fim, ficou evidente também que a melhor solução se encontra no tempo de reposição, pois, com apenas uma unidade de tempo de diferença já foi possível alcançar o suposto limite de atendimento e tempo de atendimento.

6 Conclusão

Com este trabalho prático foi possível ver, do panorama da programação, o funcionamento das estruturas de dados pilha e fila e também da importância da modularização das TAD. Um ponto desafiador foi conseguir, com apenas um programa principal simular os diversos cenários de análise. Desafio este superado com a ajuda dos parâmetros que são passados durante a execução.

De um panorama financeiro foi possível ver como pode ser feita a análise de configurações de serviço para que seja otimizado o lucro de uma empresa por exemplo.