Universidade Federal de Minas Gerais
Departamento de Ciência da Computação

Documentação TP1
AEDs II

Breno Tanure Prata

Belo Horizonte, 15/05/2017

Introdução

Este primeiro trabalho de AEDs II tem por objetivo exercitar os conceitos de estruturas de dados e implementá-las, de forma a rever certos conceitos de AEDs I também. Por meio disso, deveria ser implementada uma solução de um cenário no qual deveria ser simulada a cantina do ICEx na hora do almoço, de forma a ajudar o gerente a melhorar a distribuição de tarefas dentro dela para obter maior eficiência no atendimento de seus clientes.

Para resolver esse problema proposto, foram implementados tipos abstratos de dados (TADs), de forma a simular a fila do caixa, a fila de bandejas, a pilha de bandejas e a área na qual são servidos cada tipo de alimento.

Desenvolvimento

Foram feitos TADs para simular a pilha de bandejas, a fila do caixa, a fila das bandejas e a área na qual cada tipo de alimento é servido. O TAD de filas foi utilizado para

ambas as filas do caixa e das bandejas, e o TAD pilha, para a pilha de bandejas

```c
#include <stdio.h>
#define MAX 1000

typedef struct{
    int chave;
}TBandejas;

//typedef struct t_celula celula_bandejas;
typedef struct celula_bandejas{
    TBandejas* bandeja;
    struct celula_bandejas *prox;
}t_celula_bandejas;

typedef struct{
    t_celula_bandejas *topo;
    t_celula_bandejas *fundo;
    int tamanho;
}pilha_bandejas;

pilha_bandejas FPvazia(void);
TBandejas* desempilha(pilha_bandejas*);
void libera_pilha(pilha_bandejas*);
void empilha(TBandejas*, pilha_bandejas*);
bool pilha_ta_vazia(pilha_bandejas);
int Tamanho(pilha_bandejas pilha);
```

TAD de Pilha

Nele há as partes básicas para uma pilha: uma celula com um item e um ponteiro para a próxima célula, um tipo de item (no caso, bandejas) e a estrutura da pilha em si, com um topo e um fundo (inicio e fim da pilha, para referência na inserção, remoção e contagem de elementos da pilha). Pilhas são do tipo LIFO (*last in first out),* o que indica que suas operações são realizadas em um lado apenas da pilha: o topo. Há também as declarações das funções a serem utilizadas (e cujas complexidades serão determinadas posteriormente).

```c
#include <stdio.h>
#include <stdbool.h>

typedef struct{
    int chave;
    int chegada;
    int saida;
    int tempo_gasto;
}TPessoa;


struct Tcelula;
typedef struct t_celula_pessoa celula_pessoa;

typedef struct t_fila_pessoas {
    celula_pessoa *inicio;
    celula_pessoa *final;
} fila_pessoas;


fila_pessoas cria_fila_pessoas(void);
void libera_fila_pessoas(fila_pessoas*);
void insere_fila_pessoas(fila_pessoas*, TPessoa*);
TPessoa* remove_fila_pessoas(fila_pessoas*);
bool fpessoas_ta_vazia(fila_pessoas);

```

TAD de Fila

Nele há as partes básicas para a implementação de uma fila: um tipo para um item (no caso, pessoas, ou como está no código, TPessoa), uma célula e a fila em si, com início e final (importantes para a referência de inserção, remoção e contagem dos elementos da fila). Filas são do tipo FIFO (*first in first out)*, o que indica que suas operações ocorrem no final (inserção) e no início (remoção). Há também as declarações das funções que serão utilizadas, cujas complexidades serão determinadas posteriormente.

O plano era inserir duas pessoas a cada iteração na fila do caixa; retirar uma quando for atendida; colocá-la na fila das bandejas; retirar quando conseguisse pegar uma bandeja (o que apenas ocorria caso a pilha não estivesse vazia, visto que a reposição de bandejas acontece somente a cada 12 minutos, o que poderia significar espera significativa do cliente nessa fila) e, a cada intervalo de tempo (medido no programa como uma iteração, comparado nos termos deste problema a 1 minuto em tempo real), fazer com que o cliente seja servido um tipo de comida dentre os 4 disponíveis (assim,

completar seu prato levaria 4 intervalos), para cada cliente. Seria calculado o tempo que cada cliente gastou desde sua entrada na fila do caixa até sua saída (já com seu prato completo), uma média do tempo de cada cliente atendido dentro do espaço de tempo limite de 4 horas.

Implementação

Como visto nas imagens acima, foram implementados dois TADs distintos, um de pilhas e outro de filas. O primeiro foi utilizado para a pilha de bandejas e contém as funções de remoção de um elemento (que retornava o elemento removido), de inserção de um novo elemento na pilha, uma que retornava o tamanho da pilha em formato de inteiros, uma que criava a pilha em si (inicia-se como vazia), uma que libera a pilha (desaloca seus elementos e os remove dela) e uma que retorna de forma intuitiva se a pilha está vazia ou não.

O TAD de filas contém funções similares: uma para inserção de um novo elemento, uma para remoção de um elemento (que retorna o que removeu), uma que libera a fila (também desaloca e remove seus elementos), uma que cria a fila (inicialmente vazia) e uma que retorna de forma intuitiva se a fila está ou não vazia.

O código do Main foi feito por meio de operações sobre os TADs descritos acima, de forma que seguisse o plano inicial para a resolução do problema dado.

```
for(i=0; i<30; i++){
    empilha(bandejas, &pilha);
    printf("empilhou\n");
    printf("tamanho da pilha: %d\n", Tamanho(pilha));
}
```

Empilha 30 bandejas (número inicial máximo)

```c
for(i=0; i<tempo; i++){ //de 0 a 4h (60 min *4)
    cliente = malloc(sizeof(TPessoa));
    outro_cliente = malloc(sizeof(TPessoa));
    cliente->chegada = i;     //guarda a hora na qual o cliente chegou na fila
    outro_cliente->chegada = i;//

    insere_fila_pessoas(&fila_caixa, cliente);      ///
    insere_fila_pessoas(&fila_caixa, outro_cliente); ///chegam 2 clientes por vez
    printf("inseriu na fila do caixa\n");
    //printf("tempo chegada: %d\n", cliente->chegada);
    num_pessoas+=2;
```

Inserção de clientes na fila do caixa

```c
if(caixa != NULL){
    insere_fila_pessoas(&fila_bandejas, caixa); //vai à fila de bandejas
    caixa = remove_fila_pessoas(&fila_caixa);
}

else{
    caixa = remove_fila_pessoas(&fila_caixa);
}
```

Atendimento dos clientes

```c
if(i%12 == 0){
    for(x=0; x<10; x++){
        if(Tamanho(pilha)==30){
            break;
        }
        empilha(bandejas, &pilha);
        printf("trabalhou nas bandejas\n");
    }
}
```

Tratamento para a inserção de mais 10 bandejas (ou o restante menor que 10 que faltar até completar 30)) a cada 12 minutos

```c
if(pilha_ta_vazia(pilha) == false){ //o cliente só sai da fila quando ha bandejas para ele
        desempilha(&pilha);
        remove_fila_pessoas(&fila_bandejas); //pegou a bandeja
        printf("saiu da fila de bandejas\n");
        cliente->saida = i+4; //pega cada um dos 4 alimentos em um tempo cada

        cliente->tempo_gasto = cliente->saida - cliente->chegada;
        soma_tempo += cliente->tempo_gasto;
        atendidos++;
        free(cliente);
        free(outro_cliente);
}
```

Remoção do cliente da fila de bandejas (apenas realizada quando há bandeja para ele), acréscimo das comidas para o prato, cálculo do tempo gasto e seu acréscimo ao tempo total das pessoas atendidas. Final do programa: o cliente está completamente atendido e sai da cantina para comer seu prato

Complexidade

fila_pessoas cria_fila_pessoas(void): O(1)

void libera_fila_pessoas(fila_pessoas*): O(n)

void insere_fila_pessoas(fila_pessoas*, TPessoa*): O(1)

TPessoa* remove_fila_pessoas(fila_pessoas*): O(1)

bool fpessoas_ta_vazia(fila_pessoas): O(1)

pilha_bandejas FPvazia(void): O(1)

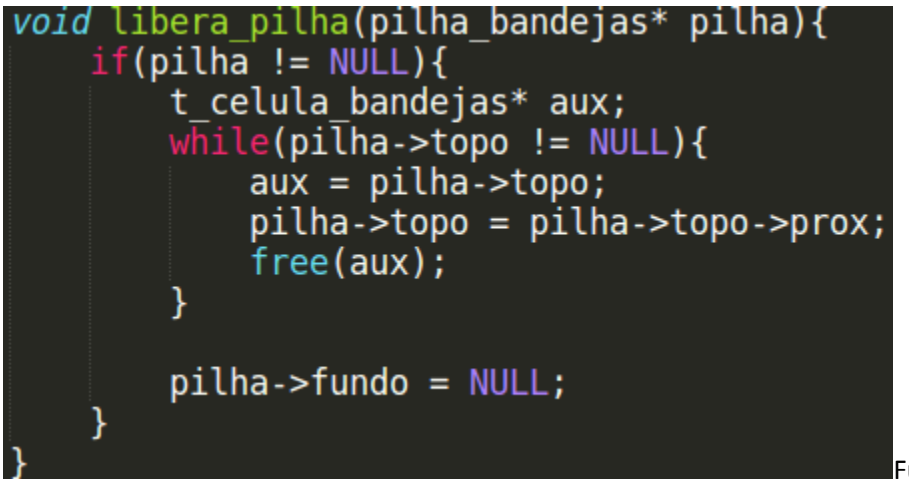
TBandejas* desempilha(pilha_bandejas*): O(1)

void libera_pilha(pilha_bandejas*): O(n)

void empilha(TBandejas*, pilha_bandejas*): O(1)

bool pilha_ta_vazia(pilha_bandejas): O(1)

int Tamanho(pilha_bandejas pilha): O(1)



Função libera_pilha: complexidade O(n)

```c
void libera_fila_pessoas(fila_pessoas* filac){
    if(filac != NULL){
        celula_pessoa* aux;
        while(filac->inicio != NULL){
            aux = filac->inicio;
            filac->inicio = filac->inicio->prox;
            free(aux);
        }

        filac->final = NULL;
    }
}
```

Função libera_fila_pessoas: complexidade O(n)

Resultados

Percebeu-se que o aumento na quantidade de bandejas a serem empilhadas e repostas contribui significativamente para o aumento de eficiência no atendimento aos clientes. Além disso, a inclusão de mais filas de caixa permite que sejam atendidos mais clientes em um mesmo tempo, além de que mais filas de bandejas (com números maiores de bandejas, para suprir ambas as filas eficientemente) também podem ser úteis na redução do tempo médio de atendimento. Uma fila de caixa e duas de bandejas não é muito eficiente, assim como a redução do número de bandejas a serem repostas (a não ser que se reduza o tempo neessário para que elas sejam recolocadas na pilha). Mais pilhas de bandejas contribuem positivamente para um atendimento mais rápido também.

Conclusão

A configuração mais eficiente foi aumentar o número de bandejas dispostas aos clientes, com menor tempo de reposição e mais de uma fila no caixa. Dessa forma, conclui-se que ela poderia ser adotada pela gerência da cantina para tentar aumentar a eficiência no atendimento a seus clientes e, consequentemente, o número de clientes atendidos, o que poderá gerar maior lucro da empresa e maior satisfação de seus clientes.