Universidade Federal da Bahia
Estrutura de Dados e Algoritmos I – MATA40
Departamento de Ciências da Computação
Professor: Danilo Santos

PROJETO 1

Sistema de Manipulação de Cheques para uma Rede de Supermercados

Alunos:
Antônio Jose Azevedo
Icaro Pereira
Pedro Gabriel Carrano
Tiago Severo

Salvador
Novembro de 2014

1. INTRODUCAO

Este projeto visa construir um simulador para um sistema de manipulação de cheques para uma determinada rede de supermercados em linguagem C. Este sistema deve, dada uma central de cheques devolutos, classifica-los, ordena-los e retorna-los, conforme pedido pelo usuário.

Para a realização deste projeto utilizamos uma pilha de processos, cada processo conteria um cheque e os seguintes dados do mesmo:

- ID (identificação do processo)
- Valor do cheque
- Data do cheque
- Nome do cliente
- Identidade do cliente
- Telefone do cliente
- Endereço do cliente
- Nome do Supermercado

A pilha funciona da seguinte forma; os cheques vão sendo empilhados na ordem em que chegam de acordo com o valor (quanto mais alto o valor do documento, maior a prioridade), e o processamento se da no cheque que se encontra no topo da pilha

Esta central deve servir para toda a rede, porem, conforme determinação do usuário pode priorizar um único supermercado, movendo os cheques deste para o topo da pilha.

2. ESTRUTURA DE DADOS UTILIZADA

A estrutura utilizada para formar a pilha foi esta:

```
18 typedef struct tno{
19
20      processo P[MAX];
21      int topo;
22
23 }pilha;
24
```

Um vetor de “processo” com um índice apontando para o processo do topo. Escolhemos um vetor pela facilidade que ele traz para manipular o conteúdo dele, além do tempo de acesso a memoria ser constante para esse caso.

A estrutura para escolhida para definir um processo foi esta:

```
typedef struct {

    char nome[15];
    char endereco[30];
    char nomeSupermercado[15];
    int rg;
    int valorCheque;
    int telefone;
    int id;

}processo;
```

Dessa forma conseguimos abranger todos as informações recorrentes ao cheque na estrutura processo.

Dividimos o programa em um header com as assinaturas das funções a serem utilizadas e uma implementação dessas funções, de forma que ao chamar as assinaturas no main, estas chamavam as funções implementadas.

3. ALGORITMO

Fizemos nosso código buscando preencher os requisitos do sistema. Aqui temos uma lista das nossas funções principais, em seguida iremos detalhar algumas delas.

```
bool empilhar(pilha *p, processo dado);

processo desempilhar(pilha *p, processo * dado);

processo* infProcesso(processo *p);

void mudarPrioridade(pilha *p, int id, int valor);

void listarTodos(pilha p);

void listarId(pilha p, int id);

void listarTopo(pilha p);

void listarBase(pilha p);

void removerProcesso(pilha *p, int id);

void supermercadoProcessos(pilha *p, char *nome);
```

Iniciamos com a criação de uma função empilhar diferenciada, de forma que os processos, ao serem empilhados, obedecessem ao ordenamento por valor.

```
24  bool empilhar(pilha *p, processo dado){
25
26      int i, j;
27      processo aux;
28
29      if(pilhaCheia(*p))
30          return false;
31
32      p->P[++(p->topo)]=dado;
33
34      if(p->topo > 0){
35          for(j = 0 ; j <= (p->topo) ; j++){
36              for(i = 1 ; i <= (p->topo) ; i++){
37                  if((p->P[i-1].valorCheque) > (p->P[i].valorCheque)){
38                      aux = p->P[i];
39                      p->P[i] = p->P[i-1];
40                      p->P[i-1] = aux;
41                  }
42              }
43          }
44      }
45      return true;
46  }
```

O novo processo seria inserido de forma ordenada na pilha utilizando o método BubbleSort.

Outra preocupação foi acerca da possibilidade do usuário retirar ou acessar um processo que não estivesse no topo da pilha. Fizemos para isso, duas funções diferentes: a função listarID, que retornava os valores do processo:

```
170 void listarId(pilha p, int id){
171
172     int i;
173
174     for(i = 0 ; i <= (p.topo) ; i++){
175         if(p.P[i].id == id)
176             break;
177     }
178     printf("NOME : %s\n", p.P[i].nome);
179     printf("ENDEREÃ‡O : %s\n", p.P[i].endereco);
180     printf("NOME DO SUPERMERCADO: %s\n", p.P[i].nomeSupermercado);
181     printf("RG: %d\n", p.P[i].rg);
182     printf("TELEFONE: %d\n", p.P[i].telefone);
183     printf("VALOR DO CHEQUE: %d\n", p.P[i].valorCheque);
184     printf("ID: %d\n", p.P[i].id);
185 }
186
```

E uma função removerProcesso, que além de buscar o processo na pilha, conseguia remove-lo, sem provocar alterações no ordenamento da estrutura.

```
186
187 void removerProcesso(pilha *p, int id){
188
189     int i, j;
190     processo aux;
191
192     if(pilhaVazia(*p))
193         exit(0);
194     for(i = 0 ; i <= (p->topo) ; i++){
195         if(p->P[i].id == id)
196             break;
197     }
198     for(j = i; j<(p->topo); j++){
199         aux = p->P[j];
200         p->P[j] = p->P[j+1];
201         p->P[j+1] = aux;
202     }
203     p->topo --;
204 }
```

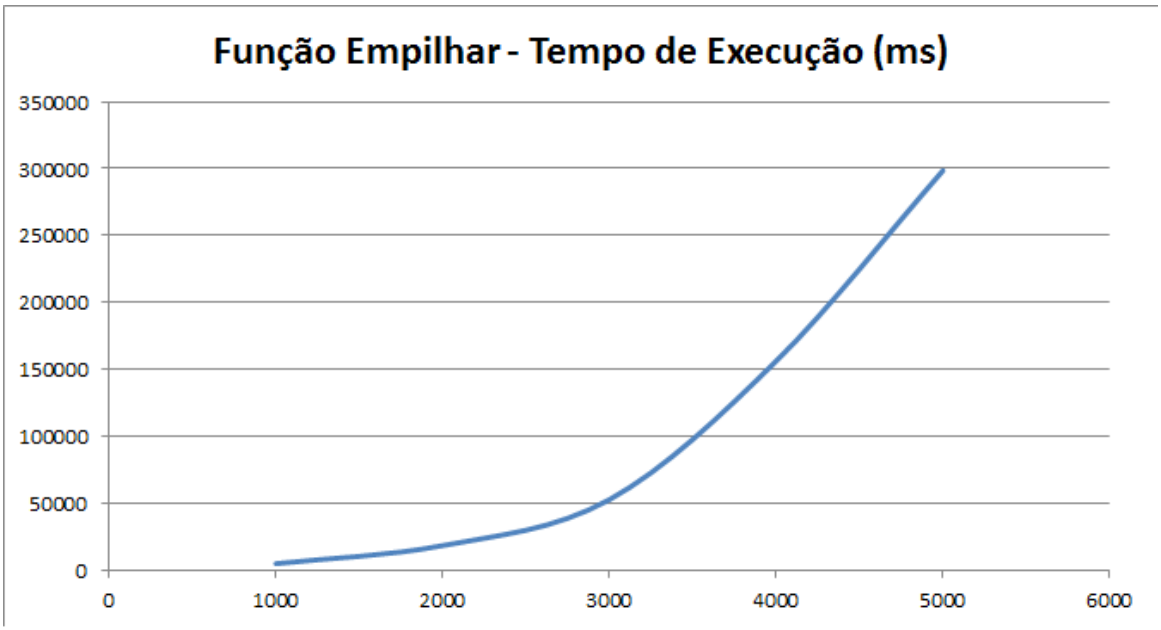
Outro requisito do usuário, era que o sistema pudesse priorizar um único supermercado, movendo assim todos os processos relacionados a ele para o topo da pilha, para isso implementamos a função supermercadoProcessos. Ela varre a pilha procurando por processos com o mesmo nome do supermercado dado pelo usuário, e move estes para o topo da pilha.

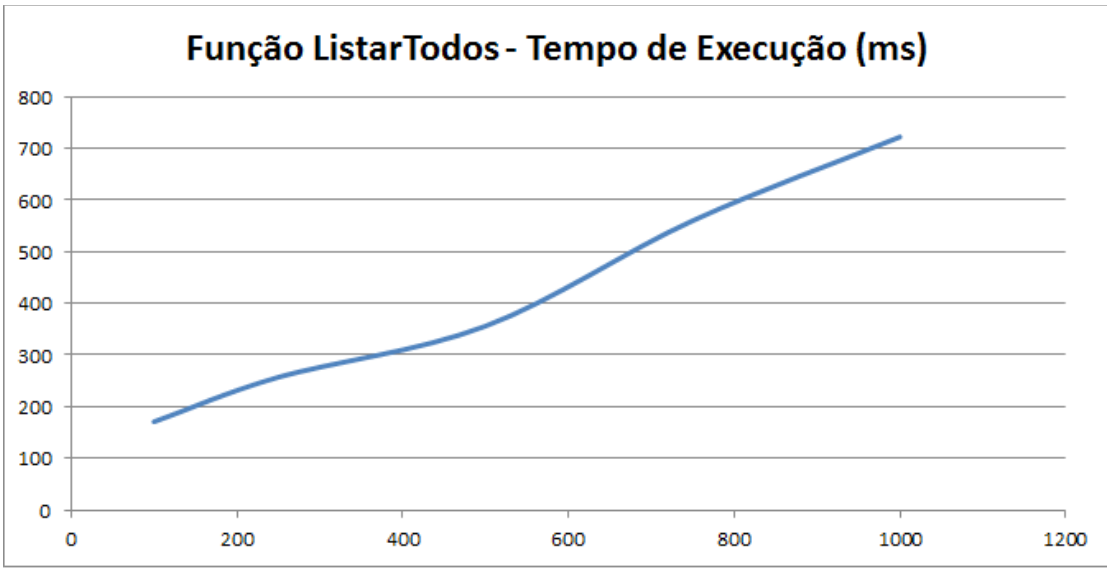
```
 98 void supermercadoProcessos(pilha *p, char *nome){
 99
100     int i, j,c = 0, flag = 0, cont =p->topo, contar=0 ;
101     processo aux;
102
103     for(j = 0 ; j< p->topo ; j++ ){
104         if (strcmp (p->P[j].nomeSupermercado, nome) == 0)
105                 contar++;
106     }
107     for(j = 0 ; j< p->topo ; j++ ){
108         if(flag == 1){
109             j = j -1;
110             flag = 0;
111         }
112         if (j == (cont - contar))
113             break;
114         if (strcmp (p->P[j].nomeSupermercado, nome) == 0){
115             for(i = j ; i <(cont) ; i++ ){
116                 aux = p->P[i];
117                 p->P[i] = p->P[i+1];
118                 p->P[i+1] = aux;
119             }
120         }
121         c = cont - contar;
122         if ((strcmp (p->P[j].nomeSupermercado, nome) == 0) && j < (cont - contar))
123                 flag = 1;
124         }
125 }
```

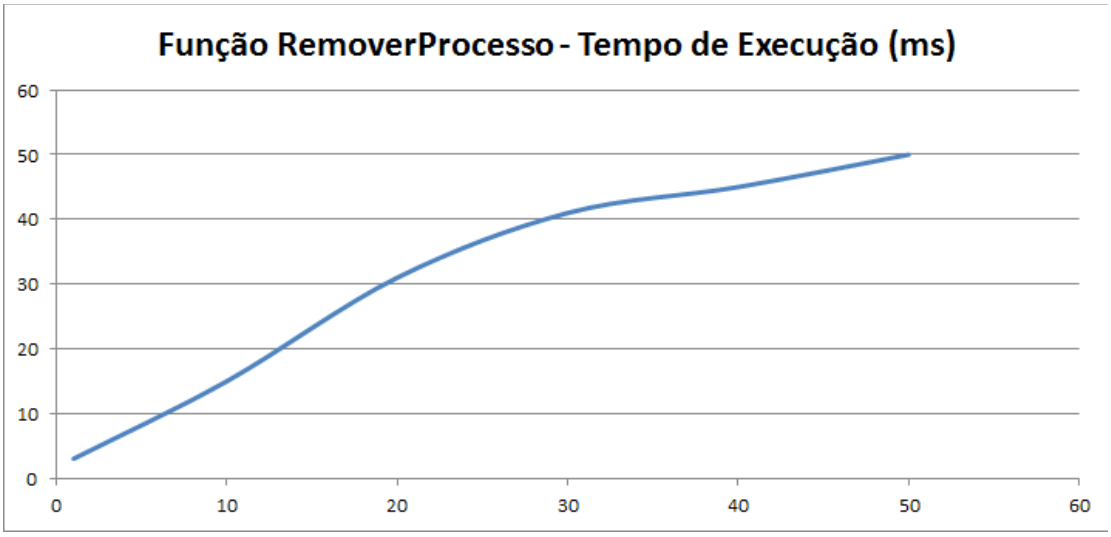
4. ANALISE EXPERIMENTAL

Para verificar a eficiência do código, fizemos os testes de algumas funções, tomando nota do tempo de execução para um certo numero de amostras.

As amostras utilizadas foram os processos.







Pode-se perceber, com base nos gráficos, que a medida que aumentamos o numero de processos, o tempo de execução cresce linearmente para todas as funções, exceto para a função empilhar. Esta cresceu exponencialmente, devido ao fato de ser implementada em conjunto com o algoritmo de ordenação (BubbleSort).

5. CONCLUSAO

Com este projeto, podemos concluir que e possível utilizar a ferramenta da programação estruturada, para modificar estruturas normalmente vistas como limitadas, e assim atender nossos objetivos.

Moldamos aqui a pilha, de modo a conseguir acessar os diferentes processos além do topo, e conseguimos também modificar a ordem, entretanto, sem alterar a regra básica da mesma, “o elemento a sair, e o elemento do topo”.