Universidade Federal Da Bahia
Curso: Sistemas de Informação
Disciplina: Estrutura De Dados
Docente: Danilo Santos
Discentes: Cássio Dourado; Carla Drieli; Edicarla Conceição; Igor Andrade; José Paulo; Priscila de Almeida; Rafael Sandes; Victor Nunes.

Projeto I: Programa de Cadastro e Organização de Cheques Sem Fundo.

Salvador, Julho de 2014

Introdução

Como trabalho final da disciplina Estrutura de Dados, foi proposto o desenvolvimento de um programa capaz de tratar os processos de cheque sem fundos usados para pagar compras nos supermercados de uma grande rede.

O sistema solicita a entrada dos dados do cheque, valor, data, estabelecimento, e os dados do cliente, nome, RG, endereço e telefone para inserir um novo processo. Esses processos são organizados em uma pilha e cada um possui um identificador único, os serviços que estão no topo desta pilha são executados primeiramente. A pilha tem prioridade relacionada ao valor do cheque.

Neste relatório descrevemos a estrutura de dados utilizada, o algoritmo, a análise experimental simplificada e as conclusões.

Toda implementação encontra-se no repositório da equipe:
https://github.com/ED20141UFBA/ProcessosSupermercado/

Descrição da Estrutura de Dados Utilizada

A estrutura de dados ‘proc’, renomeada para ‘Processo’ possui 9 tipos (incluindo ponteiro):

Tipo	Nome	Tamanho	Função
int	ID	-	Recebe o identificador que cada processo terá
char	NomeCliente	49 caracteres	Recebe o nome do cliente
char	EnderecoCliente	49 caracteres	Recebe o endereço do cliente
char	RGCliente	11 caracteres	Recebe o RG do cliente
char	TelefoneCliente	14 caracteres	Recebe o telefone do cliente
float	ValorCheque	-	Recebe o valor do cheque sem fundos.
char	DataCheque	10 caracteres	Recebe a data de compensação do cheque
char	NomeMercado	30 caracteres	Nome do Supermercado em que o cliente usou o cheque
struct proc (ponteiro)	prox	-	Ponteiro que aponta para outra estrutura do mesmo tipo

Descrição do Algoritmo

Variáveis globais

Na aplicação existem 12 variáveis globais, 2 inteiros: GeraID, que tem a função de gerar um ID único para cada processo; e QuantidadeProcessos, que contabiliza a quantidade atual de processos, usado quando o usuário seleciona a quantidade de processos na pilha (MENU); e os 9 números reais: Clock_Insere, Clock_Apaga, Clock_ApagaID, Clock_ExibeTudo, Clock_ExibeID, Clock_ExibeProximo, Clock_ExibeUltimo, Clock_ExibeTamanho, Clock_LimpaTudo e Clock_OrganizaPorNome, que recebe o tempo de execução de cada função. Nos tempos não são considerado o tempo de espera do usuário, neste momento a contagem é pausada e continua na linha seguinte.

Função OrganizaUltimoProcesso(Processo *Pilha)

Complexidade: O(n).

A Função recebe o endereço de uma pilha e verifica se o último nó (processo) está na ordem correta de acordo ao valor dos cheques, caso contrario ele será remanejado. Ou seja, ela só verifica e ordena o último nó de uma pilha. As funções ‘ExibirProcessos’ e ‘OrganizaProcessoPorNomeMercado’ utilizam esta função.

Os ponteiros *UltimoElemento e *PenultimoElemento guardarão a última e penúltima posição da pilha. Se o *UltimoElemento não estiver na posição correta, será remanejado deixando *PenultimoElemento como último(topo da pilha).

Os ponteiros *Ant e *Pos percorrem a Pilha até chegar ao final ou até que o valor que *UltimoElemento aponta seja menor que o valor de *Pos, nesse caso *UltimoElemento será realocado entre os ponteiros *Ant e *Pos.

Função OrganizaProcessoPorNomeMercado(Processo *Pilha)

Complexidade: $O(n^2)$

A Função recebe o endereço de uma pilha e realoca todos os processos de um mesmo estabelecimento para o topo da pilha (ordenamento principal), deixando ordenado por valor dos cheques (ordenamento secundário).

É criado um Nó *PilhaAux que receberá todos os processos de mesmo estabelecimento, retirando-os da pilha principal. Ao final da função, o ponteiro do Nó que está no topo da pilha principal irá apontar para o primeiro (base) Nó da PilhaAux.

Os ponteiros *Ant e *Pos tem a função de percorrer a pilha principal, quando *Pos apontar para um Nó que tenha nome igual ao do estabelecimento pesquisado, *Ant apontará para o segundo Nó a frente. O Nó excluído será enviado para o topo da *PilhaAux.

Os ponteiros *CorrePilhaAux e *NovoElementoAux são usados para tratar a *PilhaAux.

Função main()

Na função principal é criado um ponteiro *Pilha do tipo da Estrutura Processo e alocado um primeiro nó para qual *Pilha aponta. Em seguida é chamada a função Menu e quando finaliza pelo usuário, a Pilha é liberada da memória.

Função Menu(Processo *Pilha)

A Função Menu recebe o endereço de uma pilha e entra em um laço que imprime as opções que o usuário pode fazer e o tempo que cada função levou para ser executada na ultima vez que foi chamada e pede que o usuário digite de [0 – 9] para escolher uma opção, com o uso de um seletor (switch case) é chamada a função que o usuário selecionou.

Função ExibeNo(Processo *No)

Complexidade: O(1)

A função recebe um ponteiro de um nó da Pilha e imprime todas as variáveis deste nó.

Função teste_vazia(Processo *Pilha)

A Função Menu recebe o endereço de uma pilha e retornará o valor ‘1’ se a pilha estiver vazia, ‘0’ se tiver 1 elemento ou mais.

Função ExibirProcessos(Processo *Pilha)

Complexidade: O(n)

A Função ExibirProcessos usa um ponteiro *aux que percorre toda a pilha imprimindo cada nó.

Função ExibirID(Processo *Pilha)

Complexidade: O(n)

É pedido ao usuário que informe o número do ID do processo que ele deseja ver, ao receber esse valor, a função usa o ponteiro *Pos que percorre toda a pilha até achar o nó com o ID procurado, caso não tenha é informado ao usuário.

Função ExibirProximoProcesso (Processo *Pilha)

Complexidade: O(n)

A Função ExibirProximoProcesso verifica se possui algum elemento na p ilha, caso tenha, o ponteiro *Pos percorre toda a pilha até o ultimo elemento e imprime.

Função ExibirUltimoProcesso (Processo *Pilha)

Complexidade: O(1)

A Função ExibirUltimoProcesso verifica se possui algum elemento na p ilha, caso tenha, imprime o primeiro elemento.

Função Libera (Processo *Pilha)

Complexidade: O(n)

A Função Libera verifica se possui algum elemento na p ilha, caso tenha, percorre a pilha liberando cada nó.

Função EmpilharProcesso (Processo *Pilha)

Complexidade: O(n)

A Função EmpilharProcesso cria um novo nó e pede que o usuário preencha os dados. Então é verificado se possui algum elemento na pilha, caso tenha, percorre toda a pilha inserindo este novo nó no final da mesma, após isso, é chamada a função OrganizarUltimoProcesso que ordenará esse novo nó.

A variável global QuantidadeProcessos é incrementada.

Função ApagarID (Processo *Pilha)

Complexidade: O(n)

É pedido ao usuário que informe o número do ID do processo que ele deseja apagar, ao receber esse valor, a função usa o ponteiro *Pos que percorre toda a pilha até achar o nó com o ID procurado, então o nó é excluído da pilha e liberado da memória. Caso não tenha o ID procurado, é informado ao usuário.

A variável global QuantidadeProcessos é decrementada.

Função Apagar (Processo *Pilha)

Complexidade: O(n)

A função usa o ponteiro *ult que percorre toda a pilha até achar o ultimo nó, então o nó é excluído da pilha e liberado da memória.

A variável global QuantidadeProcessos é decrementada.

Análise de Tempo de Execução

Foram utilizadas as três funções de maior complexidade:

- Empilhar
- Desempilhar
- Organizar por Nome (Priorizar por nome de estabelecimento)

Computador utilizado:

- AMD Phenom 9550 Quad-Core de 2.20Ghz
- 4 Gb de Ram DDR-2
- SO: Windows 7 - 64x
-

Empilhar:

Desempilhar:

Organizar por Nome

:

Conclusão

Diante do software desenvolvido no trabalho, percebemos a importância de utilizar as estruturas de dados. Sem estas, algumas aplicações poderiam ficar limitadas na linguagem utilizada. Analisamos, também, a relevância de codificar com funções e modularizações, a fim de ter um código organizado e que possibilite um fácil reuso do que foi construído.