Trabalho Prático: GeekComerce

Programação e Desenvolvimento de Software II

Catarina Enya Diniz Pereira
Isabelle Vieira Barbosa
Larissa Gabrielli da Silva Pereira
Tatiana Alvares Guimarães de Oliveira
Victoria Rocha Saliba

1. Introdução

O objetivo deste trabalho é a implementação de um sistema de ecommerce para venda de artefatos geek e da cultura pop.

2. Modelagem do Problema

2.1 Atributos, Funções e Procedimentos - Produtos

Para modelagem de Produto, utilizou-se uma classe com os seguintes atributos:

float _preco : representa o preço do produto.
std::vector<int> _avaliacoes : recebe as avaliações dos usuários
float _mediaAvaliacoes : recebe a média das avaliações
std::string _nome : representa o nome do produto
std::string _categoria : representa a categoria do produto
std::string _cor : representa a cor do produto
std::string descricao : representa a descrição do produto
std::string _material : representa o material do produto
Int _codigoProduto : representa o código do produto

E as seguintes funções:

Void avaliarProduto : recebe o número de avaliações do produto dado pelo usuário
Void getComentarios : imprime os comentários dado pelo usuário
Void setComentario : recebe o comentário dado pelo usuário
Void getPreco : retorna o preço do produto
Void getMediaAvaliacoes : recebe a média das avaliações do produto
Void getNome : recebe o nome do produto
Void getCategoria : recebe a categoria do produto
Void getCor : recebe a cor do produto
Void getDescricao : recebe a descrição do produto
Void getMaterial : recebe o material do produto
Void getCodigoProduto : recebe o material do produto
Int retornaNumComentarios : retorna o número de comentários no produto
std::vector<std::string> _comentarios : representa os comentários sobre o produto

2.1.2 Atributos, Funções e Procedimentos - Produtos - Canecas

Para modelagem de Caneca, utilizou-se uma classe com os seguintes atributos, além dos herdados de Produto:

Float _diametro : representa o diametro da caneca

E as seguintes funções, além das herdadas de Produto:

Float getDiametro : retorna o diâmetro da caneca
Void imprimeProduto : imprime os atributos da caneca

2.1.3 Atributos, Funções e Procedimentos - Produtos - Acessórios

Para modelagem de Acessorio, utilizou-se uma classe com os seguintes atributos, além dos herdados de Produto:

Std::string _tipo : representa o tipo do acessório

E as seguintes funções, além das herdadas de Produto:

Std::string getTipo : retorna o tipo do acessório
Void imprimeProduto : imprime o os atributos do acessório

2.1.4 Atributos, Funções e Procedimentos - Produtos - BlusasEMoletom

Para modelagem de BlusasEMoletom, utilizou-se uma classe com os seguintes atributos, além dos herdados de Produto:

Char _tamanho : representa o tamanho do produto
Std::string _tipo: representa o tipo(blusa ou moletom)

E as seguintes funções, além das herdadas de Produto:

Chat getTamanho : retorna o tamanho do produto
Std::string getTipo : retorna o tipo(blusa ou moletom)
Void imprimeProduto : imprime o os atributos do produto

2.2 Atributos, Funções e Procedimentos - Usuário

Para modelagem de **Usuário**, utilizou-se uma classe que simula um usuário, cujos atributos são:

std::string _nome: representa o nome do usuário
std::string _email: representa o email do usuário
std::string _senha: representa a senha do usuário

Para modelagem de **Usuário**, utilizou-se também as seguintes funções:

std::string getNome(): retorna o nome do usuário
std::string getEmail(): retorna o email do usuário
std::string getSenha(): retorna a senha do usuário

2.2.1 Atributos, Funções e Procedimentos - Usuário - Comprador

Para modelagem de **Comprador**, utilizou-se uma classe que simula um usuário do tipo comprador, ou seja, modelou-se a classe comprador como uma subclasse de usuário cujos atributos são:

int _numeroComprasHistorico: representa o número de itens no histórico do comprador
int _numeroComprasCarrinho: representa o número de itens no carrinho do comprador
int _numeroAvaliacoes: representa o número de avaliações realizadas pelo comprador

float _totalCarrinho: representa o valor total dos produtos no carrinho do comprador
float _dinheiro: representa a quantidade de dinheiro que o comprador possui

std::string _CPF: representa o cpf do comprador

std::string _endereco: representa o endereço do comprador

std::vector <Produto> carrinho: representa o carrinho do comprador

std::map <std::string, Produto> historico: representa o histórico do compradores

Para modelagem de **Comprador**, utilizou-se também as seguintes funções:

std::string getNome(): retorna o nome do usuário
std::string getEmail(): retorna o email do usuário
std::string getSenha(): retorna a senha do usuário

2.2.2 Atributos, Funções e Procedimentos - Usuário - Administrador

Para modelagem de **Administrador**, utilizou-se uma classe que simula um usuário do tipo administrador, ou seja, modelou-se a classe administrador como uma subclasse de usuário cujos atributos são:

int _qntdade_de_requisicoes: guarda o número de requisições de aumento de saldo feitas por compradores
std::vector<BlusasEMoletom> _bluemols: recebe do arquivo “produtos.csv” as blusas e moletons em estoque
std::vector<Caneca> _cans: recebe do arquivo “produtos.csv” as canecas em estoque
std::vector<Acessorio> _aces: recebe do arquivo “produtos.csv” os acessórios em estoque
std::vector<Comprador> _shoppers: recebe do arquivo “usuarios.csv” os compradores cadastrados
std::map<std::string, float> _requisicoes: recebe do arquivo “requisicoes.csv” as requisições de aumento de saldo feitas por compradores

Para modelagem de **Comprador**, utilizou-se também as seguintes funções:

Administrador(): Construtor da classe Administrador, sempre inicia o objeto com o nome ADMIN, email admin1@gmail.com e senha 123tasalvo
void produtoCsvToVector(): Classe que pega as informações sobre os produtos do arquivo csv no qual eles estão e insere-as em um vector para fácil manuseio
void usuarioCsvToVector(): Classe que pega as informações sobre os usuarios do arquivo csv no qual eles estão e insere-as em um vector para fácil manuseio
void reqCsvToMap(): Classe que pega as informações sobre as requisições do arquivo csv no qual eles estão e insere-as em um map para fácil manuseio
int removeItem(std::string nome_do_produto): Função para remover um item do estoque da loja
void exibirPerfil(): Imprime na tela as informações sobre o administrador
void adicionaPedido(std::string email, float valor): Função chamada pelos usuários quando desejam requisitar aumento em seu saldo
void mostraPedidos(): Imprime na tela as requisições de aumento de saldo
int aprovaPedido(std::string email): Função para aprovar uma requisição de aumento de saldo
int excluiUsuario(std::string email): Exclui um usuário
~Administrador(): Destrutor da classe

2.3 Atributos, Funções e Procedimentos - Ecommerce

Para modelagem de **Ecommerce**, utilizou-se uma classe que simula o ecommerce de maneira geral, ou seja, as interações entre as classes bem como com o usuário:

std::vector <Usuario> usuarios: representa os usuários cadastrados no ecommerce
std::vector <Comprador> compradores: representa os compradores cadastrados no ecommerce
std::vector <Produto> produtos: representa os produtos cadastrados no ecommerce
std::vector <Caneca> canecas: representa os produtos da categoria caneca cadastrados no ecommerce
std::vector <Acessorio> acessorios: representa os produtos da categoria acessórios cadastrados no ecommerce
std::vector <BlusasEMoletom> blusasEmoletons: representa os produtos da categoria caneca cadastrados no ecommerce
std::string userLogged = "": representa o login de um usuário no ecommerce

Para modelagem de **Ecommerce**, utilizou-se também as seguintes funções:

void cadastrarComprador (std::string n, std::string em, std::string s, std::string cpf, std::string endereco, int numCarr, int numHist, int numAval, double din): cadastra um comprador dada as informações fornecidas pelo mesmo

void cadastrarCaneca(int cod, float preco, float mediaAvaliacoes, std::string nome, std::string cor, std::string descricao, std::string material, float diametro): Cadastra uma caneca
void cadastrarAcessorio(int cod, float preco, float mediaAvaliacoes, std::string nome, std::string cor, std::string descricao, std::string material, std::string tipo): Cadastra um acessório
void cadastrarBlusasEMoletom(int cod, float preco, float mediaAvaliacoes, std::string nome, std::string cor, std::string descricao, std::string material, char tamanho, std::string tipo): Cadastra uma blusa ou moletom

bool procurarUsuario(std::string email): localiza um usuário dado o email do mesmo
bool procurarComprador(std::string em): localiza um comprador dado o email do mesmo

void imprimirUsuarios(): imprime a lista de usuários que consta no vector
void imprimirCompradores(): imprime a lista de compradores que consta no vector
void imprimirProdutos(): imprime a lista de produtos que consta no vector

bool validaAlfanumerica(std::string s): checa se o nome e a senha do usuário seguem o formato definido
bool checaEmail(std::string em): checa se o email do usuário segue o formato definido

int buscaIndiceCaneca(int cod): Procura uma caneca pelo seu código de identificação no vector canecas e retorna o índice.

int buscaIndiceAcessorio(int cod): Procura um acessório pelo seu código de identificação no vector acessorios e retorna o índice.

int buscaIndiceBlusasEMoletom(int cod): Procura uma blusa ou moletom pelo seu código de identificação no vector blusasemoletons e retorna o índice.

int buscaIndiceProdutos(int cod): Procura um produto pelo seu código de identificação no vector produtos e retorna o índice.

Produto buscaProdutos(int cod):

Comprador buscaComprador(std::string em):

int tamanhoArquivo(const char* file_name): Recebe o nome de um arquivo como argumento e retorna o tamanho dele.

void addCarrinho(): Adiciona produtos no carrinho do usuario e retorna a pagina anterior.

void limparTela(): limpa a tela do ecommerce.

void impCarrinho(): Cria um menu de opçoes para compradores na pagina do carrinho.

void impHistorico(): Cria um menu de opçoes para compradores na pagina do historico.

void impProdutos(): Cria um menu de opçoes para compradores na pagina de produtos.

void mostraProdutos(): Cria um menu de opçoes para o administrador na pagina de produtos.

void mostraUsuarios(): Cria um menu de opçoes para o administrador na pagina de usuarios.

void listaUsuarioArquivo(): lista os usuários registrados no arquivo

void gravaUsuarioArquivo(): registra os usuários do arquivo no vector de usuários

long int geradorCod():

bool checaCodigo(int cod): Verifica se um produto ja esta no vector pelo código dele.

void listaProdutosArquivo(): Lista os produtos de um arquivo csv.

void gravaProdutosArquivo(): Grava os produtos em um arquivo csv.

void listaComentariosArquivo(): Lista os comentários de um arquivo csv.

void gravaComentariosArquivo(): Grava os comentários em um arquivo csv.

void adicionaComentario(int cod, std::string coment): Atribui um comentário a um produto pelo seu código de de identificação.

void loginUsuario(): Realiza o login de um usuário no ecommerce.

void logoutUsuario(): Realiza logout de um usuario no ecommerce.

void inicio(): Cria o menu inicial do programa.

void menuUsuario(): Cria um menu para administradores.

void menuComprador(): Cria um menu para compradores.

void dadosComprador(): Pede os dados do comprador a ser cadastrado e chama função a de castro.

void dadosProdutos(): Pede os dados do produto a ser cadastrado e chama função a de castro.

3. Análise Teórica do Custo Assintótico

3.1 Análise Teórica do Custo Assintótico de Tempo

Considerando que o algoritmo receberá um número N, a leitura do mesmo ocorrerá linearmente e a complexidade da entrada de dados se dá por O(1). Realizar o caminhamento na espiral quadrada exige que durante a iteração do algoritmo todos os pontos menores ou igual a N sejam percorridos, portanto, a iteração se dá em O(N). Deste modo, a complexidade algorítmica é O(n).

4. Análise de Experimentos

A analise experimental da implementação foi realizada considerando diferentes valores de n, variando entre 0 e o maior número inteiro que pode ser representado em uma variável inteira na linguagem C. A contagem do tempo demandado por cada execução foi realizada utilizando-se a biblioteca “<time.h>”, registrando-se a média da quantidade de clocks demandada em cada um dos testes. Os testes foram realizados em uma máquina com Intel® Core™ i7-2630QM CPU @ 2.00GHz × 8, Memória RAM de 4GB com frequência de 1333Mhz e sistema operacional Ubuntu 18.04 LTS.

Como a espiral quadrada não possui limites de restrição, apresentando inúmeras possibilidades de pontos para os quais as coordenadas deverão ser calculadas, o algoritmo foi construído tendo como limite único a entrada de dados padrão. Dessa forma, o custo de tempo depende exclusivamente do número fornecido por meio da entrada de dados padrão, um número inteiro n maior ou igual a zero e menor ou igual ao maior inteiro positivo representável através de variável inteira na linguagem C++, sendo o custo algorítmico da O(n).

5. Implementação

Utilizou vectors, maps e classes para representação dos elemento que compêm o ecommerce.

6. Conclusão

Após a implementação e análise do algoritmo foi possível notar que o algoritmo funciona bem próximo ao ótimo, sendo suas análises comprovadas a partir de testes experimentais. Portanto, o algoritmo resolve o problema proposto e ainda promove o aprendizado de lógicas para extração de informações dado o problema.