Universidade Estadual do Oeste do Paraná – UNIOESTE
Campus de Cascavel
Colegiado de Ciência da Computação

Estrutura de Dados
1º Trabalho 2017

Descrição

Este trabalho visa possibilitar ao aluno abordagens a novos problemas além dos vistos em sala, criando assim um pensamento mais abrangente, preparando-o para uma gama maior de solução de problemas. Através desta implementação busca-se também mostrar de que forma algoritmos podem ser utilizadas em aplicações externas à computação.

Especificações

a) O trabalho pode ser desenvolvido individualmente ou em duplas. Trabalhos com mais de dois alunos serão desconsiderados e receberão nota zero.
b) O valor total do trabalho representará 50% da nota do segundo bimestre.
c) A linguagem utilizada no desenvolvimento deve ser C++, porém o compilador pode ser escolhido pelo grupo.
d) Devem ser entregues os arquivos contento código fonte. Este deve ser entregue em formato digital em mãos. Não serão aceitos trabalhos enviados via e-mail.
e) Cópias de trabalhos ou soluções prontas serão penalizadas com a perda total do valor do trabalho.
f) A data final de entrega e apresentação do trabalho é 27/07/2018 até as 17:00.
g) As datas e horários da entrega/apresentação deverão ser combinadas com antecedência com o professor da disciplina.

LINHAS

O jogo proposto consiste em formar uma matriz bidimensional *m* x *n*. As posições da matriz podem assumir dois estados. O primeiro, representado pelos pontos brancos na Figura 1, correspondem às posições a serem preenchidas. Já o segundo estado representa as posições de início ou término de uma linha (células coloridas e numeradas).

Na imagem exibida a seguir, pode-se verificar a presença de valores numerados de um a sete. Cada valor é duplicado. A ideia é que os dois valores sejam ligados por uma única linha que não seja sobreposta por nenhuma outra linha. Ou seja, uma posição x,y só pode ser ocupada por uma linha ou elemento numerado.

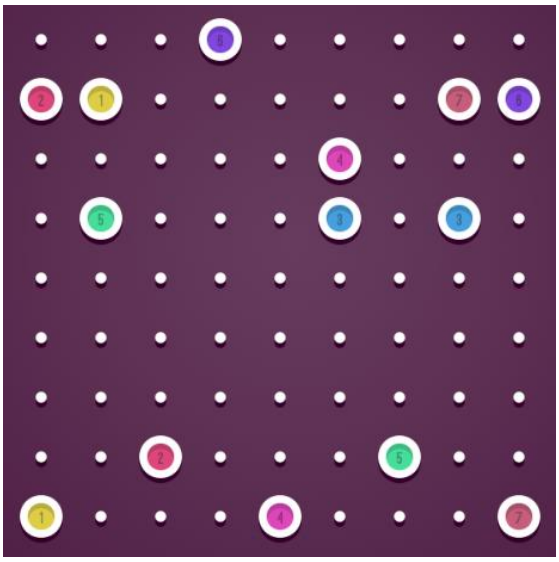

Figura 1. Exemplo de matriz inicial para o jogo Linhas.

Na Figura 2, abaixo, são apresentadas as ligações entre as duas posições com valor 2 e as duas posições com valor 6. Cada ponto marcado com um valor pode ser tanto inicial como final, ou seja, uma linda pode, por exemplo, sair da posição [1,4] até a posição [2,9] ou pode ser traçada no sentido contrário.

A decisão de onde a linha tem início e onde termina, bem como o caminho que ela fará fica a encargo do jogador e não do programador.

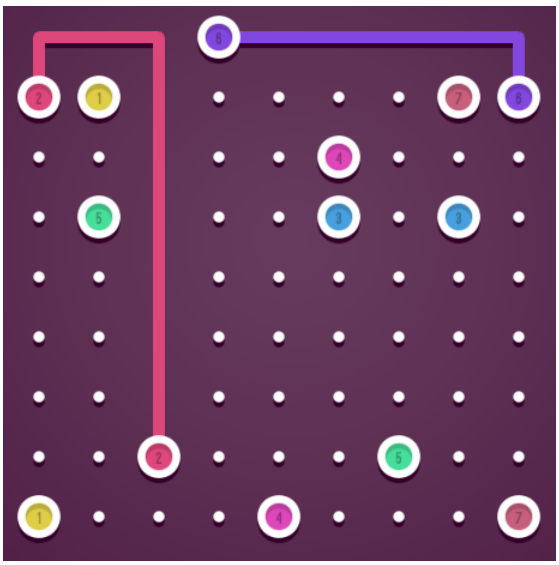

Figura 2. Exemplo de conexão entre os pares de pontos.

O processo de ligação entre as posições numeradas é realizado para cada um dos pares presentes na matriz, conforme apresentado nas Figuras 3 e 4. No caso da figura 4, pode-se ver que todos os pares numerados são conectados.

O objetivo do jogo é conectar todos os pares de posições numerados e, ao mesmo tempo, passar por todas as posições marcadas com pontos brancos, conforme mostrado na Figura 4. O resultado alcançado com sucesso é visto na Figura 5, em que todos os elementos da matriz foram “visitados” e nenhuma linha cruzou com outra.

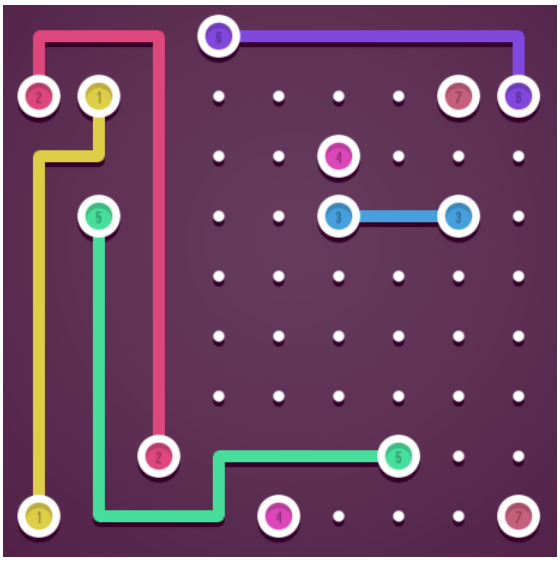

Figura 3. Linhas conectando um conjunto maior de pontos.

Figura 4. Ligações entre praticamente todos os pares.

Caso o jogador escolha caminhos que conectem todos os pares de nós sem que todas as posições da matriz sejam preenchidas, o jogo continua. Algum dos caminhos poderá ser desfeito e refeito na tentativa de preencher todas as coordenadas.

O que deve ser implementado na solução do exercício:

a) A solução deve, obrigatoriamente, usar orientação a objetos. Para tanto, pode-se, por exemplo, se criar uma classe do tipo lista e outra do tipo nó;
b) Uma vez que cada conjunto de elementos ligados pode ser percorrido nos dois sentidos, a abordagem construída deve utilizar listas duplamente encadeadas.
c) A implementação deve consultar o usuário sobre o tamanho da matriz a ser utilizada e permitir que o usuário determine cada um dos pontos de início e fim. Para tanto, deve-se construir um procedimento chamado ConstroiMatriz que receberá por parâmetro as

dimensões especificadas, fará a alocação dinâmica do espaço e fará a determinação das posições iniciais (informadas pelo usuário).

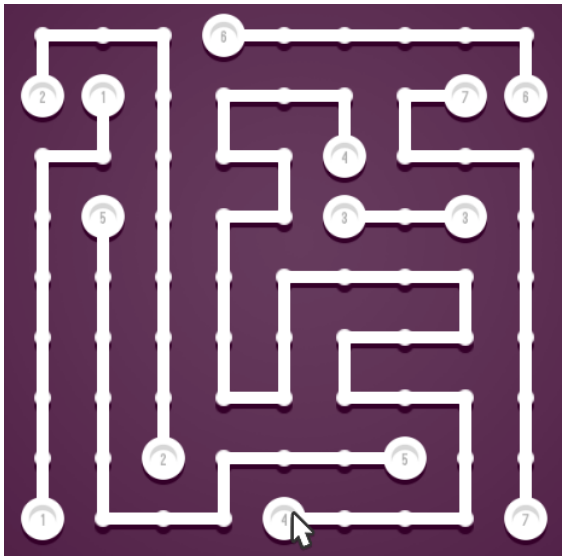

Figura 5. Resultado concluído com sucesso.

Este procedimento deve questionar também quanto ao número de pares que se farão presentes na matriz (deve, obrigatoriamente haver pelo menos 1 par e na máximo (n x m)/2 pares. Tal verificação deve ser realizada.

d) Depois de preparada a matriz, o programa deve permitir ao usuário (através da função ConstroiCaminho) que ele selecione uma determinada célula para começar a traçar os percursos. Estas células necessariamente são elementos numerados e, portanto, deve-se fazer a verificação.

e) Assim que a posição iniciar for escolhida, o usuário vai informando a direção que o caminho deve seguir: cima, baixo, direita, esquerda. O programa deve então ir marcando o caminho e adicionando as posições à lista correspondente.

f) O processo de marcação deve fazer algumas verificações no momento em que o usuário informa uma posição:

a) É uma posição válida? O usuário não está tentando ir para uma direção impossível? Por exemplo, ir para a coluna -1;

b) O usuário não pode passar por uma posição preenchida com um valor que seja diferente do inicial. Por exemplo, ele começou pelo índice “3” na matriz. Durante o caminho, ele não poderá passar pela posição que contém o valor “2”, apenas pelo valor inicial. Para tanto, deve-se implementar a função ChecaPosicao;

c) O usuário pode escolher um caminho que já foi usado por outra lista. Quando isso ocorrer, a lista anterior deve ser apagada pois espera-se que a solução informada agora seja a correta. Para tanto, deve-se implementar a função ChecaCaminho, que verifica se a posição escolhida se sobrepõe a um caminho anterior.

g) Deve ser implementada uma função ChecaFinal que verifica se todas as listas realmente conectam cada um dos pares de valores da matriz e deve checar também se todas as posições da matriz fazem parte de algum caminho. Caso as duas verificações sejam válidas, o programa deve exibir mensagem informando que o jogo foi concluído com sucesso.

MEMÓRIA

O Jogo, que é bastante simples, funciona da seguinte forma: Inicialmente é construída uma matriz de dimensões m x n.

Todas as células são apresentadas com o mesmo aspecto, como apresentado na Figura 6 a seguir.

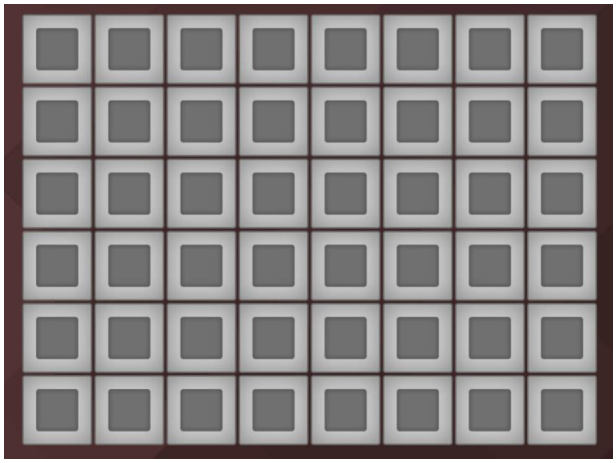

Figura 6. Exemplo de matriz inicial.

Em seguida, é apresentada uma sequência de posições com coloração em destaque, como pode ser visto na Figura 7 abaixo. As posições que fazem parte da sequência não precisam, necessariamente, ser vizinhas. Por exemplo, a primeira posição escolhida pode ser a [3,4], a segunda, [2,7], a terceira, [2,8] e assim sucessivamente.

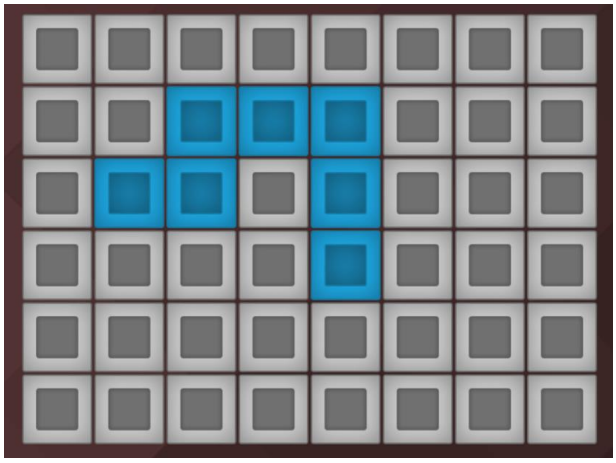

Figura 7. Posições destacadas na matriz original.

No entanto, além de marcar um conjunto de posições em destaque, elas devem ser mostradas, uma a uma, em uma sequência fixa, como mostrado na Figura 8.

Essa sequência não é exibida toda de uma vez. Apenas uma posição é destacada por vez, ou seja, primeiro é destacada a posição 1, então ela volta à representação original e a posição 2 é destacada. O processo é repetido para toda a sequência.

A solução consiste no usuário “assistir” à sequência que é apresenta e então, apontar exatamente quais foram as posições destacadas e corretamente na ordem em que foram exibidas. Por exemplo, no Figura 8, se o usuário informar que a posição [2,3] (que foi a terceira a ser exibida) aparece antes da posição [3,1] (primeira exibida), o

jogo deve acusar um erro dizendo que a ordem está fora do determinada. O processo então começa todo novamente.

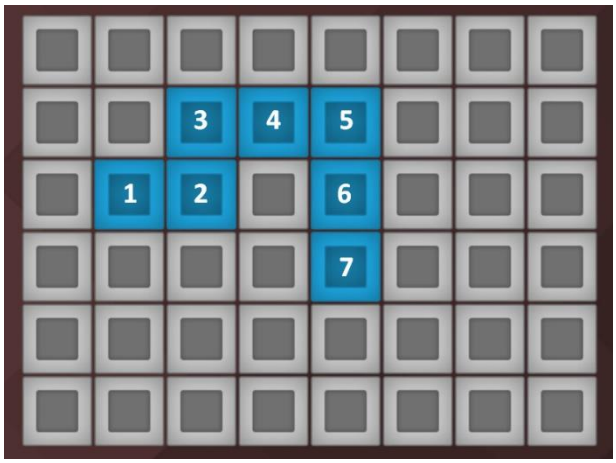

Figura 8. Exemplo de sequência marcada na matriz.

Outro erro que pode ocorrer é quando o usuário informa uma posição fora daquelas que foram exibidas na sequência, como exibido na Figura 9 (posição [4,6]) onde o usuário escolheu uma coordenada que não foi destacada. Neste caso, o jogo apresenta um erro, dizendo ao jogador que ele escolheu uma posição incorreta e o processo começa todo novamente.

Este processo é repetido até que o jogador seja capaz de dizer com precisão quais foram as coordenadas destacadas pelo jogo e exatamente em que sequência elas apareceram. Quando isso ocorrer, o jogador vence.

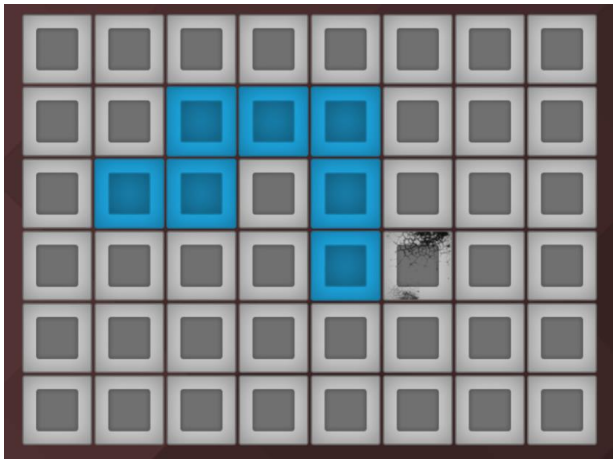
Figura 9. Exemplo de posição incorreta informada.

O que deve ser implementado na solução do exercício:

a) A solução deve, obrigatoriamente, usar orientação a objetos. Para tanto, pode-se, por exemplo, se criar uma classe do tipo fila e outra do tipo nó;
b) Uma vez que o caminho fornecido pelo usuário deve seguir obrigatoriamente a mesma sequência representada na tela, deve-se implementar uma estrutura de fila, de forma que as posições sejam armazenadas na ordem correta.

c) Deve ser construída um método chamado FormaLista que receberá como parâmetro o número de posições que a lista deverá conter. Este procedimento deve construir a lista de posições para que será apresentada ao usuário. Lembrando que uma mesma posição não pode ser percorrida mais de uma vez, ou seja, em um percurso sempre estarão presentes elementos distintos.

Dentro deste método deve ser chamado o método de inserção dos elementos na lista.

d) Construção do método ExibeLista. Este método receberá como parâmetro o endereço do primeiro elemento da lista formada em FormaLista e o tempo (em milissegundos) de exibição de cada elemento em tela. O método exibe apresenta ao usuário cada uma das posições da lista durante certo tempo, de forma que ele possa memorizar as posições em sequência.

e) Construção do método ValidaEntrada. Este método deve receber do usuário a posição informada. Para cada posição informada o método deve verificar se:

1) A posição informada está na lista;

2) Após confirmar que a posição está na lista, deve checar se está na ordem correta.

Caso o usuário escolha uma posição incorreta (não presente na lista), o método deve exibir mensagem informando o erro cometido e iniciar o processo de exibição novamente.

Caso o usuário escolha uma posição que faz parte da lista, mas está em ordem incorreta, o método deve avisar a ocorrência deste erro e iniciar o processo de exibição novamente.

O método ValidaEntrada é executado até que o usuário informe corretamente a sequência fornecida na ordem correta.

MAHJONG

O objetivo deste exercício é desenvolver um jogo de pedras similar ao jogo chinês Mahjong. A ideia consiste em inicialmente distribuir um conjunto de pedras em um tabuleiro de forma aleatória e ir removendo-as até que não restem mais peças.

O dominó é formado por um conjunto de pedras distintas onde cada uma possui um número par de cópias (geralmente 4 peças iguais). Assim, para se remover cada uma, é preciso selecionar um par de peças iguais. Para tanto, é preciso que estas peças estejam "liberadas".

No jogo tradicional, para uma peça estar liberada é preciso que duas bordas conjuntas estejam voltadas para fora. Por exemplo, uma peça que possui o topo e borda esquerda voltadas para fora pode ser escolhida. Já uma peça que possui o topo e a base livres não pode ser escolhida.

Assim como ocorre nos jogos de cartas, inúmeras abordagens do jogo foram desenvolvidas ao longo dos anos. Neste trabalho o problema será tratado da seguinte forma: o tabuleiro do jogo será representado por uma matriz retangular m x n composta por um grupo de peças de número par (não haverá peças sem repetições). A figura 10 ilustra um cenário possível para esta proposta. A matriz é composta por 36 pedras (6 x 6) e, o conjunto de pedras é composto de 18 pares.

Figura 10. Tabuleiro de Mahjong.

Neste cenário, cabe ao usuário ir selecionado par a par até que todas as peças sejam removidas do tabuleiro. Caso seja possível chegar ao final das peças sem que o jogo "trave" o usuário é dito vencedor.

Neste trabalho as pedras poderão ser escolhidas de uma forma mais simples: caso a pedra possua uma borda lateral para fora, ela poderá ser escolhida. Tomemos por exemplo a ilustração da Figura 11. Neste caso as duas pedras destacadas em verde mostram que o usuário selecionou duas posições aceitáveis: [2,6] que tem a borda direita para fora e [3,1] que tem a borda esquerda para fora.

Por outro lado, a Figura 12 demonstra um caso onde há problema na seleção das peças. Neste caso, a posição [2,1] é uma opção válida, pois possui a borda esquerda para fora. Já a pedra posicionada em [1,4] não pode ser escolhida uma vez que suas duas bordas laterais estão bloqueadas. Esta escolha caracteriza então um movimento inválido e que não deve ser permitido.

Além do tratamento de se escolher apenas peças desbloqueadas o jogo deve verificar se o usuário escolheu dois elementos similares, ou seja, se as peças formam um par.

O que deve ser implementado na solução do exercício:

a) Cada pedra deve ser tratada como um objeto individual, contendo sua posição X e Y, seu valor ("Not bad", "Impossibru", "Mother of God", "Why?"...) e se a peça já foi excluída ou não;
b) A matriz deve ser uma estrutura (que pode ser estática) capaz de armazenar um objeto em cada uma de suas posições. Para se construir a matriz deve ser implementado um procedimento que receba as dimensões da matriz e reserve a memória necessária para todo o armazenamento.

1) Deve-se verificar se os valores informados pelo usuário resultado em uma multiplicação de resultado par. Por exemplo, não é possível montar um tabuleiro com dimensões 5 x 5.

Figura 11. Exemplo de movimento permitido neste trabalho.

Figura 12. Exemplo de movimento não permitido.

2) As posições de todas as peças devem ser escolhidas de forma aleatória;

b) Deve implementar para cada objeto pedra os seguintes métodos:

1) Construtor default;
2) Métodos Get e Set para modificar de posição X e Y;
3) Construtor explícito;
4) Remove Pedra: responsável por retirar do tabuleiro a pedra informada (X,Y);
5) Verifica remoção: recebe por parâmetro o objeto a ser excluído e retorna o valor 1 se puder ser excluído e valor caso, caso contrário.

c) Caso o usuário informe uma posição não permita o sistema deve repetir o processo de escolha até que duas peças válidas sejam escolhidas.

d) Deve-se implementar uma função VerificaPossibilidade que analisa todas as peças restantes e retorna se ainda há movimentos possíveis ou se o jogo está travado.

e) Caso o jogo esteja travado, deve-se executar o procedimento Embaralha que pegará todas as pedras restantes e as redistribuirá nas posições restantes de forma a dar continuidade ao jogo. Neste procedimento deve-se atualizar as posições X e Y gravadas em cada objeto.

f) Deve-se implementar a função VerificaFim que, a cada iteração válida do usuário, verifica se todas as pedras foram removidas. Caso tenham sido, apresenta mensagem em tela informando o usuário da vitória. Caso contrário executa VerificaPossibilidade e continua o processo.

CC CONSTRUÇÕES S/A

Em uma empresa de comércio de itens de construção há a divisão dos funcionários em quatro categorias distintas:

a) Motorista, que é responsável por fazer o transporte dos produtos comprados da fábrica onde são produzidos até a loja física onde são vendidos aos clientes. Além de ter a incumbência de efetuar as entregas aos consumidores.

b) Responsável de estoque: funcionário que tem como função descarregar e carregar os veículos de entrega e gerenciar o estoque da empresa.

c) Vendedor: tem a responsabilidade de atender aos clientes e efetuar a venda dos produtos em estoque.

d) Gerente: esta categoria tem a responsabilidade cuidar da empresa como um todo, fazendo a ligação entre os setores e garantindo o bom funcionamento da entidade.

1) Construa uma superclasse chamada Funcionário que armazene os campos comuns a todas as categorias presentes na hierarquia da empresa. Sabe-se a priori que são armazenados o Nome, Sobrenome, Categoria, Código (valor inteiro), CPF, Tempo de Empresa (em meses), Salário Base, Salário Final e Número de Filhos.

Em seguida, construa subclasses (herdadas da primeira) que contenham as informações específicas de cada uma das quatro categorias de funcionários. O modificador usado para a criação das subclasses deve ser public.

2) Elabore um método membro de classe que recebe como parâmetro um objeto do tipo

subclasse criado (cada categoria terá um método distinto) e então exiba em tela quanto tempo falta para que o funcionário possa entrar em férias.

3) Elabore um método na classe principal chamado CalculaSalário. Esta função deve receber como argumento o tipo de função desempenhada pelo salário e o seu código. Com base nos valores deve calcular o valor do salário daquele funcionário e retornar como um float. No entanto, cada categoria possui uma forma de cálculo de trabalho. **Cada subclasse deve implementar o método próprio sobrepondo o método original da superclasse.**

3.1 O salário de um motorista consiste no salário base mais bonificações pelo desempenho ao longo do mês. Cada funcionário desta categoria armazena um número de "corridas" efetuadas, tanto para buscas produtos na indústria quanto para entrega-los nos clientes. Caso ele consiga efetuar mais do que 100 "corridas" em um mês ele recebe uma bonificação de 10% sobre o salário base. Outro fator que influencia no salário final é a quantidade de horas em trânsito. Se o funcionário possui habilitação do tipo "D" e ficou em trânsito por mais de 160 horas ele recebe um bônus de 50% sobre o salário base e, caso o mesmo possua habilitação na categoria "C" e esteve em trânsito por mais de 180 horas, seu bônus será de 0.4 em relação ao salário base.

3.2 O salário do responsável pelo estoque depende, além do valor base, da quantidade de mercadoria carregada e descarregada em estoque. Para cada tonelada vinda da indústria que é descarregada na empresa ou carregada em direção aos consumidores o funcionário recebe um adicional de 3% sobre o valor base. Além disso, caso efetue mais do que 250 cargas e descargas durante o mês, haverá um bônus de 35% sobre o salário base.

3.3 O salário de um responsável por vendas consiste na soma do salário base, acrescidos de 15% de comissão sobre o valor total de produtos vendidos. Para estimular um melhor atendimento aos clientes, o gerente garantiu aos membros desta categoria que, caso efetuem mais do que 300 vendas no decorrer do mês, haveria um bônus de 5% sobre o valor base acrescido da comissão.

3.4 O salário do cargo de gerente consiste em seu salário base mais um adicional de 3% para cada funcionário (das outras categorias) que trabalha na unidade em que gerencia.

4) Construa um procedimento não membro chamado Compra que receba como parâmetro o código de do funcionário que buscou a mercadoria e o tempo gasto (em minutos) para efetuar o processo de transporte. O procedimento deve então atualizar o total de horas em trânsito e o número de "corridas" efetuadas.

5) Elabore um método membro chamado EfetuaVenda que, com base no código do funcionário e no valor total da venda efetuada, atualiza o valor total vendido por aquele funcionário e o número de vendas efetuadas.

6) Construa um procedimento membro chamado CargaDescarga que recebe como parâmetro um objeto da subclasse "Reponsável de estoque" e o peso da carga a ser descarregada ou carregada. O procedimento deve atualizar então o total de Kg manipulados

7) Construa um procedimento que imprima em tela, de forma organizada, todas as informações de cada funcionário. Como cada categoria possui informações específicas, **este procedimento membro deve ser escrito de forma a sobrescrever o método presente na classe principal**.