UFMG

Algoritmos 1 – Trabalho Prático 1

Casamento Estável

Nome: Arthur Pontes Nader

Matrícula: 2019022294

1. Apresentação

Para organizar melhor as promoções da Black Friday durante o período de pandemia, deve-se alocar certo número de clientes a uma loja com base no seu estoque de um produto, evitando assim aglomerações. Certos clientes tem prioridade sobre outros e cada cliente sempre irá preferir uma loja que esteja mais perto dele, caso essa loja lhe faça uma proposta de agendamento para compra do produto.

Desse modo, esse trabalho teve como objetivo a realização da modelagem e implementação computacional de um programa capaz de resolver de forma eficiente o problema exposto acima. Para isso, utilizou-se o algoritmo Gale-Shapley, responsável por encontrar um casamento estável entre clientes e lojas.

2. Modelagem computacional

Para implementação do programa, criou-se dois tipos abstratos de dados, de tal forma que pudessem representar as lojas e os clientes envolvidos no problema a ser solucionado. A seguir, há uma breve descrição das TADs desenvolvidas, algumas observações sobre os principais atributos e métodos de cada uma delas e informações sobre a “main”. O algoritmo responsável pela solução do problema será tratado detalhadamente no próximo tópico.

- **Classe Cliente:**

  A classe Cliente tem como principal papel o armazenamento dos dados de cada cliente. Esses dados serão guardados em atributos, como: idade, número de identificação, localização, loja atual que o cliente está alocado e a distância até ela.

A seguir, encontra-se uma breve descrição dos principais métodos da classe.

void calcularTicket(): com base nos parâmetros passados para o construtor, esse método calcula o ticket que determinará a ordem de preferência do cliente.

int calcularDistancia(Loja *loja): com base nas coordenadas do cliente e da loja passada como parâmetro, esse método determina a distância do cliente até a loja.

void analisarProposta(Loja* LojaComProduto, int* numeroTotalDeProdutos): esse método é responsável por processar a proposta feita ao cliente, determinando se esse deve ou não mudar de loja. Mais detalhes sobre a implementação serão fornecidos no próximo tópico.

double getTicket(): retorna o valor do atributo ticket do cliente.

- **Classe Loja:**

A classe Loja é responsável por guardar informações sobre a loja, como sua localização, o número de produtos em seu estoque, e os clientes alocados a ela.

Os principais métodos implementados para manipulação de atributos de objetos dessa classe são:

int getNumProdutos(): retorna o número de produtos atual que a loja possui em estoque.

int getProximoCliente(): retorna o número de identificação do próximo cliente que a loja deverá fazer uma oferta.

void imprimirLojaMaisClientes(): imprimi o número de identificação da loja e o número de identificação de todos os clientes alocados a ela.

- **Main:**

A leitura dos dados de cada cliente e loja ocorrerá na “main” do programa, com a utilização de dois laços de repetição “for”. Após isso, o vetor de clientes é ordenado, a função responsável pelo casamento estável é chamada e, por fim, imprime-se os clientes alocados a cada uma das lojas.

3. Descrição da solução

A seguir, encontra-se o pseudocódigo resumido das principais funções utilizadas para resolução do problema. A primeira função, responsável pelo casamento estável, faz uma chamada a segunda função, que fará a análise da proposta por parte do cliente.

```
StableMatch(clientes, lojas, numero_de_produtos)
    lojasPendentes = lojas
    enquanto tamanho(lojasPendentes) > 0
        enquanto numero_de_produtos(loja) > 0 e loja não tiver proposto a todos os clientes
            proximo_cliente.analisarProposta(loja, lojasPendentes)
        exclua lojasPendentes[0]
    retorne lojas

analisarProposta(loja, lojasPendentes)
    se cliente.alocado == false
        cliente.alocado = true
        atualize atributos de loja
        cliente.lojaAtual = loja
    senão:
        se cliente.distanciaAtual > cliente.distancia(loja)
            remova cliente de cliente.lojaAtual.clientesAlocados
            atualize atributos de cliente.lojaAtual e de loja
            adicione cliente.lojaAtual.id no vetor lojasPendentes
            troque lojaAtual para a loja passada como parâmetro
```

O método StableMatch é responsável pela implementação da lógica do algoritmo de Gale-Shapley. O código desenvolvido termina a execução quando todas as lojas já tiverem proposto a todos os clientes ou não tiverem mais produto em estoque. Quando um cliente aceita a proposta de uma loja, o atributo da loja que determina o próximo cliente a ser feito uma proposta é

aumentado. Quando o estoque de uma loja acaba ou a loja já propôs a todos os clientes, muda-se a loja ofertante. Assim, o algoritmo sempre termina em algum momento. Para analisar isso, dividiremos em duas situações possíveis:

- Há mais produtos do que clientes: nesse caso, depois que todos os clientes estiverem alocados, as lojas que ainda possuem o produto em estoque continuarão a ofertar para os clientes, mesmo que esses já estejam alocados. Se necessário, isso é feito repetidas vezes, até que chega um momento em que lojas com estoque maior do que zero já propuseram agendamento a todos os clientes possíveis. Assim, a execução termina.
- Há uma quantidade menor ou igual de produtos do que de clientes: nessa situação, o algoritmo termina a execução pois haverá algum momento em que todas as lojas terão estoque igual a zero.

Já o método analisarProposta aloca o cliente à primeira loja que o fizer uma proposta de agendamento. Além disso, caso ele já esteja alocado, a loja atual só será alterada se uma outra loja mais perto desse cliente também lhe fizer uma proposta, o que garante que ao final do processo o cliente estará alocado, dentre todas as lojas que lhe propuseram um agendamento, àquela mais perto de sua localização. Nesse caso em que há uma mudança de loja, o estoque da nova loja diminuirá, enquanto o da loja antiga será acrescentado em uma unidade. Dessa maneira, a loja antiga possui produto em estoque, logo, ela deverá retornar à lista de lojas que devem fazer propostas aos clientes.

Suponha uma instabilidade do tipo: um cliente c_1 está agendado para uma determinada loja l_1, entretanto, existe uma loja l_2 mais próxima de c_1 que possui estoque para atender sua demanda ou possui agendamento para um cliente c_2 de ticket menor. Isso é uma contradição, pois, se c_1 tem maior prioridade do que c_2, então a loja l_2 com certeza lhe fez uma proposta antes de propor a c_2. Se c_1 já estivesse alocado a l_1, ele trocaria para l_2. Caso não estivesse, ele aceitaria a proposta de l_2 e rejeitaria uma possível futura proposta de l_1, pois l_1 é mais distante do que l_2.

Agora, suponha uma outra instabilidade da seguinte forma: há um cliente c_1 sem agendamento, todavia, ainda existe uma loja l_1 que possui estoque disponível para atendê-lo. Isso também é uma contradição, pois enquanto a loja possuir o produto em estoque, ela deve continuar propondo agendamentos até que ofereça o produto a todos os clientes. Assim, em algum momento, a loja l_1 terá que ter oferecido o produto a c_1, que como está sem agendamento, aceitará a proposta.

4. Análise da complexidade de tempo

Para realização da análise assintótica de cada método, define-se duas variáveis: “n” será o número de clientes e “m” o número de lojas. Os principais métodos chamados durante a execução são analisados a seguir, para que se possa concluir sobre a complexidade de tempo da solução final.

- **Métodos da Classe Cliente:**

Os métodos dessa classe expostos no tópico de modelagem computacional realizam diversas operações aritméticas, comparações de valores, mudanças de ponteiros, alteração de atributos e variáveis, adição de elemento a vetores, entre outros. Tudo isso é feito em tempo constante. Logo, esses métodos são todos O(1).

- **Métodos da Classe Loja:**

Os métodos dos objetos da classe Loja que retornam ou alteram o valor de um atributo são O(1).

O método de impressão é no pior caso O(n). Entretanto, esse método não participa diretamente da solução do problema, sendo usado posteriormente para exibição dos resultados obtidos.

- **Função sort da biblioteca padrão:**

A função de ordenação da biblioteca padrão utiliza um algoritmo chamado IntroSort, um algoritmo híbrido usado para aumentar a eficiência. Sua complexidade é O(n log n). Assim, essa será a complexidade de tempo para a ordenação dos clientes de acordo com suas prioridades.

- **StableMatch:**

A função que realiza o casamento estável de clientes e lojas faz chamadas apenas a procedimentos que são O(1). Assim, no pior caso, as m lojas terão que fazer propostas a cada um dos n clientes, sendo que dessa forma, a complexidade será O(mn).

- **Conclusão**

Dessa maneira, a complexidade assintótica do programa como um todo será O(mn + n log n).