TP1 - Documentação

Nome: Rodrigo Ferreira Araújo — Matrícula: 2020006990

Algoritmos 1 - Novembro 2021

1 Modelagem Computacional

Para a implementação desse trabalho, criamos um tipo de dado "Cliente", que representa o cliente do problema e armazena apenas informações necessárias para a implementação do problema: id, localização no grid, o id da loja para o qual foi alocado, a distância para essa loja e o seu valor de ticket. Decidimos não armazenar sua idade, estado e forma de pagamento mais frequente porque são atributos usados apenas no cálculo do ticket, que é feito durante a leitura dos dados da entrada padrão. Note que o id da loja do tipo Cliente é inicializado com 0 para simbolizar que ainda não houve proposta para esse cliente.

Em seguida, definimos o tipo "Loja", que faz função similar ao tipo cliente, mas em relação à loja do problema em questão. Nesse sentido, esse tipo armazena o id, a capacidade (estoque atual), localização no grid, o número de clientes que a loja já ofertou até então e o vetor de clientes alocados para a loja.

Para o desenvolvimento do problema, uma loja pode admitir um ou mais clientes, que corresponde exatamente à quantidade armazenada pelo atributo "capacidade" do tipo. Os clientes só podem, no entanto, serem alocados para uma loja. Assim, centramos a resolução do problema de acordo com a lógica do algoritmo de Gale-Shapley para retornar um casamento estável entre lojas e clientes, adaptado para uma lógica "muitos para um", uma vez que uma loja é considerada "totalmente casada" sse não há mais espaço para clientes. Note que o casamento será ótimo do ponto de vista das lojas, uma vez que são elas as proponentes.

2 Algoritmos e Estruturas de Dados

2.1 Estruturas de Dados

Tipo Cliente: Armazena informações imprescindíveis ao clientes.
Inteiros: id, cliente_x, cliente_y (localização), id_loja, distancia_loja_alocada.
Double: ticket (valor do ticket em ponto flutuante de precisão dupla)

Tipo Loja: Armazena informações imprescindíveis às lojas.
Inteiros: id, capacidade (estoque), loja_x, loja_y (localização), clientes ofertados.
Vector(STL): clientes (vetor de clientes admitidos).

vector<Cliente> Clientes: Vetor (STL) principal com todos os clientes.

vector<Loja> Lojas: Vetor (STL) principal com todas as lojas, de modo que a loja com o id **i** está em **Lojas[i]**.

2.2 Funções/Algoritmos

2.2.1 cmpCliente

Função booleana auxiliar para controlar a forma que a ordenação de um vetor de clientes deve ocorrer. Esta função é usada como parâmetro no procedimento **sort** do C++, que admite um procedimento do tipo para reger a ordenação. Assim, segundo essa função, um vetor de clientes deve ser ordenado por ordem decrescente dos números de ticket e, para uma sequência de 2 ou mais tickets iguais, os ids dos clientes deve estar em ordem crescente.

Desse modo, uma vez que a lista de preferência é a mesma para todas as lojas (pois é baseada nos números de ticket), as proposições serão feitas linearmente no vetor principal de clientes ordenado dessa maneira, que trata o caso de exceção em que uma loja tem que escolher o cliente com o menor id para clientes com tickets iguais. Além disso, essa ordenação também é usada durante a impressão das listas de clientes das lojas conforme esperado.
Complexidade de tempo: O(1).

2.2.2 findIndexVector

Função usada no algoritmo de casamento principal para encontrar (retornar), a partir de seu id, o índice de um cliente específico num vetor de clientes, a fim de removê-lo desse vetor, pois escolheu uma loja mais perto. Complexidade de tempo: O(n), onde n é o número de clientes e procurar um id é uma operação relevante.

2.2.3 findIndexVector

Função usada no algoritmo de casamento principal para encontrar a distância entre um cliente e uma loja no grid. Essa distância é obtida calculando o máximo entre os módulos das diferenças das coordenadas x e y da loja e do clientes. Referência: Distância de Chebyshev = $\max(|x_2 - x_1|, |y_2 - y_1|)$. Complexidade de tempo: O(1).

2.2.4 alocacao

Função usada no algoritmo de casamento principal (tanto no caso quando é a primeira oferta de um cliente quanto no caso em que o cliente escolhe uma loja mais perto do que foi alocado) que realiza todas as operações necessárias para representar a alocação de um cliente a uma loja.

Nesse sentido, os parâmetros **id_loja** e **distancia_loja_alocada** do cliente são, respectivamente, atualizados com o id da loja proponente e a distância daquele cliente para essa loja. Além disso, o cliente é acrescentado no vetor de clientes da loja e o parâmetro **capacidade** da loja é decrementado de 1. Complexidade de tempo: O(1).

2.2.5 casamento_estavel

Função principal que realiza o casamento estável esperado entre lojas e clientes sob a lógica do algoritmo de Gale-Shapley. Para uma loja específica, o loop principal checa se ainda há estoque disponível e essa loja ainda não ofereceu para todos os clientes, isso significa que ela está apta a oferecer vaga aos clientes. O cliente aceitará a oferta se ele não recebeu nenhuma ainda, ou a distância para a loja proponente é menor do que a distância para a loja alocada ou essas distâncias são iguais mas o id da loja proponente é menor do que o id da loja alocada, tratando, assim, os casos de exceção.

Como a lista de preferência das lojas pelos clientes é a mesma, toda proposta que uma loja faz (aceita ou rejeitada) incrementa o contador de clientes ofertados dessa loja, que é usado como índice para a próxima proposta a ser feita. Desse modo, uma loja não propõe para um cliente mais de uma vez. Além disso, caso uma loja "roube" o cliente de outra loja, o cliente é removido da loja antiga, sua capacidade é incrementada de 1 e o procedimento é chamado recursivamente com o índice da loja antiga para que ela "tape o buraco feito", isto é, proponha para todos os clientes ou esgote o seu estoque, o que dá a loja a condição de "totalmente casada", assim como o exemplo dado em aula.

Complexidade de tempo: O(m · n), onde **m** é o número de lojas e **n** é o número de clientes, uma vez que, com a implementação recursiva usada, uma loja não propõe para o mesmo cliente mais de uma vez e, no máximo, todas as lojas proporão para todos os clientes, isto é, **m** · **n**.

2.3 Programa Principal (main)

Em primeiro lugar, todas as informações necessárias são lidas da entrada padrão e armazenadas nos vetores de acordo (o cálculo e armazenamento do valor de ticket é feito no loop de leitura de dados do cliente. Em seguida o vetor de clientes é ordenado conforme a lógica da função **cmpCliente** descrita anteriormente usando a função **sort** (STL) do C++, que, nesse caso, possui complexidade assintótica O(n · log n), onde **n** é o número de clientes.

Após a ordenação de vetores dos clientes, realizamos o casamento entre Lojas e Clientes com um loop while e um índice **i**: O índice **i** controla o índice da loja no vetor Lojas. Desse modo, dada a explicação da função **casamento_estavel**, ao final de uma iteração desse loop while, a loja **i** e todas as anteriores a ela no vetor estarão com seus "buracos tapados", ou seja, propuseram para todos os clientes ou esgotaram seu estoque de acordo com as proposições feitas até a loja **i**. Desse modo, ao final desse loop while, a última loja e todas as anteriores estarão casadas estavelmente com os clientes.

Por fim, para todas as lojas, ordenamos seus respectivos vetores de clientes sob a lógica de **cmpCliente** descrita anteriormente para a impressão dos

resultados conforme esperado. Como temos uma ordenação do mesmo tipo da usada no começo do programa principal, temos uma complexidade de tempo de pior caso O(m · a · log a), onde **m** é o número de lojas e **a** é a quantidade de clientes da loja que admitiu o maior número de clientes, uma vez que uma ordenação é feita para cada loja.

Análise de Complexidade de Tempo Assintótica: A complexidade de tempo assintótica do programa como um todo é a complexidade do programa principal. Assim, temos a complexidade das ordenações de vetores: O(n · log n) + O(m · a · log a) e a complexidade do algoritmo que realiza o casamento estável: O(m · n). Logo, a complexidade total é O(m · n + n · log n + m · a · log a). Note que os processos de leitura e impressão de dados não foram considerados relevantes para a complexidade.

2.4 Pseudocódigo

casamento_estavel (Lojas, Clientes, indice_loja)
while(há alguma loja com estoque disponível e ainda não ofereceu para todos os clientes)
| **if**(o cliente não recebeu nenhuma proposta)
——| cliente aceita a proposta da Lojas[indice_loja]
| **else if**(o cliente recebeu uma proposta de uma loja mais perto ou de mesma distância mas com um id menor)
——| cliente removido da Lojas[indice_loja_antiga]
——| casamento_estavel(Lojas, Clientes, indice_loja_antiga)
——| cliente aceita a proposta da Lojas[indice_loja]
| **else**
——| cliente rejeita a proposta da Lojas[indice_loja]

i = 0
while(i < numero de lojas)
| casamento_estavel(Lojas, Clientes, i)
| i recebe i + 1