Documentação
Algoritmos 1 - TP1

Aluno: Guilherme Ferreira Costa
Matrícula: 2020006590

Introdução

O principal problema a ser resolvido nesse TP é o casamento estável entre lojas e clientes, de modo que eu tenha uma solução mais favorável para as lojas(poderia ser o inverso também).

De forma resumida, temos que a preferência das lojas por um cliente, se dá por uma expressão que calcula o ticket do cliente, e essa expressão depende da idade do cliente, o estado onde o cliente reside e o método de pagamento mais utilizado por esse cliente. E as preferências do cliente são relacionadas a distância do cliente para uma loja Li, sendo i o conjunto de todas as lojas. Em caso de empates no ticket ou na distância, a loja ou usuário que ganham preferência são os com id mais baixo.

Estruturas de dados

Foram utilizadas diversas estruturas para resolver o problema em questão, como:

- Classe Loja
- Classe Cliente

Foram duas classes extremamentes para guardar as informações de cada instância do problema.

Também foi utilizado diversas estruturas de dados da stl, como:

- map ou dicionário (chave -> valor)
- queue (filas)
- vector (vetores)
- pair (struct com dois elementos)

Além de combinações desses tipos de estruturas…

Algoritmos

Foram utilizadas muitas funções da stl durante todo o processo de implementação da solução do problema, como:

- sort() para ordenar os vetores de acordo com suas prioridades
    - Criei funções de comparação separadas para cada execução do problema a fins de testes, mas poderia ser usado o stable_sort() que sairia direto

- outros métodos de inserção e remoção de vetor ou fila

Descrição sucinta do TP em pseudo-código junto com a modelagem do problema:

Cria dois maps (score_estado, score_pagamento)

Pega os dados do Grid da entrada padrão, juntamente com o número de lojas e clientes

Itera m vezes para colher as informações das m lojas e coloca no vetor Lojas
Itera n vezes para colher as informações dos n clientes e coloca no vetor Clientes

// começamos a calcular as prioridades das listas
itera n vezes:
    para cada cliente calcula seu ticket

    adiciona um par {id_cliente, ticket} em um vetor de pares temporário(ListaPriorioridadeLojaTemp)

Ordenamos a lista temporária de pares
Adicionamos essa lista em uma fila // para termos O(1) lá na frente
// listas de preferência de cada cliente
Criamos um vetor de vetores de pares (ListasdeListasDePrefDosClientes)

itera n vezes:
    cria um vetor de pares para o cliente c
    itera m vezes:
        Calcula todas as distâncias do cliente c
        Coloca essas distâncias em um par {id_loja, distância}
        Aloca o par na lista do cliente em questão

    // saímos da iteração do cliente
     Ordena o vetor de pares (já fica na ordem correta)
     E coloca na Lista de Listas De Prev Dos Clientes

// agora ajeitamos as informações para rodar o Gale–Shapley

// invertendo as listas de preferencia dos clientes
Cria um vetor de maps (VetorDeHashes)
itera n vezes:
    Cria um map (myMap)
    itera m vezes:
        preenche o map com os valores da lista de de pref. de c como chave
        e como valor a sua posição na lista de preferência

// para sabermos pelo id do cliente, em qual loja ele está
Cria um map (LojaMap) tendo como chave os ids dos clientes
e como valor -1

// para sabermos pelo id da loja, os clientes alocados para ela
Cria uma lista de listas ClientesMap

Criamos um vetor (ListasDeFilasDePrioridadeDasLojas) com m filas de prioridade das lojas(as filas são idênticas) // para guardar o estado da iteração das lojas no gale-shapley

Criamos uma fila (LojasNaoVisitadas) com todos os ids das lojas

```
//Gale-shapley
enquanto houver lojas não visitadas:

        Pega  a primeira loja da fila (LojasNaoVisitadas)
        Retira a primeira loja da fila (LojasNaoVisitadas)

        enquanto (houver vagas no estoque da loja) e (a lista de prioridade da
                                                        não está vazia)

                Pega o primeiro cliente da lista de prioridade

                se ele ainda não está em nenhuma loja:
                        adiciona ele a loja
                        diminui o estoque da loja
                        adiciona o cliente ao vetor ClientesMap da loja

                se ele já tem uma loja associada mas prefere a loja atual:
                        retira ele do vetor ClientesMap da loja associada
                        aumenta o estoque da loja associada em 1
                        coloca ele no vetor ClientesMap da loja atual
                        diminui o estoque da loja atual

                        coloca a loja associada novamente na fila de lojas não visitadas

                se ele prefere a loja associada:
                        continue
```

Complexidade Assintótica

Observando as iterações como importantes na complexidade assintótica, temos vários loops O(n) e O(m), sendo n o número de clientes e m o número de lojas.

Mas a loop com maior iteração(que é o que é considerado) é:

- O(n* (m + mlog(m)))
- foi usado para criar a listas de listas de preferência dos clientes e depois ordená-las

Explicação:

Iteramos n vezes:

A cada iteração é calculado a distância do meu cliente para todas lojas (ou seja, m iterações)

Ordenamos a lista de preferência do cliente com um sort que é O(mlog(m))

Assim, temos:

- O(n * (m + mlog(m))) = O(nm * (1 + logm))