Trabalho Prático 1
Problema da BlackFriday

Jean Lucas Almeida Mota

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

jeanlk@ufmg.br

1. Introdução

O problema a se enfrentado possui natureza fictícia com base na época comercial Black Friday. Neste problema, dado as lojas e os clientes que moram na região, deve-se alocar os clientes nas lojas. Sob as seguintes condições:

1. As lojas dão prioridade para os clientes com maior Score.
2. As lojas possuem capacidade máxima de clientes.
3. Os clientes devem ser alocados para a loja mais próxima a eles.
4. Se um cliente não for alocado para a loja mais próxima a ele, não deve haver naquela loja vagas ou clientes com um Score menor que o dele.
5. Caso exista um empate, a condição de desempate será o Id do cliente ou da loja.

2. Implementação

O programa foi desenvolvido na linguagem C++, compilado pelo compilador G++.

2.1 Estrutura de dados

A implementação do programa teve como base o problema do casamento estável. Dessa forma a modelagem foi feita a fim de facilitar a implementação do algoritmo de Gale-Shapley. O qual foi modificado para se adequar ao problema proposto. Pseudocódigo:

PAR INSTÁVEL: um cliente c1 é alocado em uma loja l1, mas há uma loja l2 mais próxima que está alocando um cliente c2 com um score menor que c1. Ou seja, c1 prefere l2 e l2 prefere c1 aos pares aos quais foram alocados.

INCIALIZE M como vazio.
WHILE (existir uma loja com vaga E essa loja não propôs para todos os clientes)
    A loja l1 propõe para o primeiro cliente c1 da sua lista de preferencias.
    IF (c1 não tiver alocado)
        c1 é alocado para l1
    ELSE IF (c1 prefere l1 ao seu atual par l2)
        c1 é alocado para l1

```
            l2 libera uma vaga
        ELSE
            c1 rejeita l1
RETURN O casamento estável M
```

Assim foi usada a estrutura Fila Encadeada, a fim de otimizar o algoritmo e facilitar a implementação.

Observação: Está sendo levado em conta o Matching ótimo para as lojas.

2.2 Classes

A fim de modularizar a implementação, o TAD fila encadeada foi implementado de forma independente e uma classe de mais alto nível faz uso dela.

Classe FilaEncadeada: É a implementação do TAD fila encadeada. Além dos métodos básicos de Enfileira(), Desenfileira() e Limpa(). Foi implementado o método Imprimir(), usado para imprimir a fila completa e facilitar a corretude.

Classe Cliente: É a classe que modela as informações referentes aos clientes, possui: id, estado, pagamento, idade, ticket e posição geográfica.

Classe Loja: É a classe que modela as informações referentes às lojas, possui: id, capacidade e posição geográfica.

Classe SistemaPref: É uma classe de alto nível, que faz uso de todas as classes anteriores. Possui os métodos:

- Leitura(), faz a leitura dos dados via linha de comando;
- CalculaTicket(), faz o cálculo do ticket de todos os clientes cadastrados.;
- OrdenaPrefLojas(), baseado nos tickets dos clientes, cria a lista de preferência das lojas em relação aos clientes.
- OrdenaPrefClientes(), baseado na distância entre os clientes e as lojas, cria listas de preferencia para cada cliente e ao fim, cria um rank de lojas para cada um dos clientes, a fim de otimizar a comparação entre lojas preferidas.
- Imprimir(), faz a impressão na tela das lojas e dos clientes.

Os métodos de ordenação usam uma versão do algoritmo de ordenação por Inserção. O método de inserção foi escolhido por sua boa velocidade e baixa complexidade de implementação.

Classe Matching: É a classe de mais alto nível, a qual se utiliza das listas de preferencias criadas pela classe SistemaPref para criar o casamento estável entre lojas e clientes. Possui os métodos:

- GaleShapley(), que implementa o pseudocódigo apresentado anteriormente, a fim de resolver o problema do casamento estável.
- ImprimirMatching(), que imprimi o resultado já formatado.

3. Análise de Complexidade

3.1 Tempo

A entrada do sistema pode ser interpretada da seguinte forma: Sendo **n** o número de clientes e **m** o número de lojas.

O sistema é pode ser dividido em 4 métodos principais de alto nível, logo somando a complexidade de tempo deles, teremos a complexidade total do sistema. Os métodos são:

Leitura(): Ao inicializar percorremos todas as lojas e todos os clientes. Temos o custo de inicialização:

$$\boldsymbol{O(n+m)}$$

OrdenaPrefLojas(): Custo para montar as listas de preferência das lojas:

Melhor caso $\boldsymbol{O(n)}$

Pior caso $\boldsymbol{O(n^2)}$

OrdenaPrefClientes(): Custo para montar as listas de preferência dos clientes:

Melhor caso $\boldsymbol{O(n \cdot m)}$

Pior caso $\boldsymbol{O(n \cdot m^2)}$

GaleShapley(): Implementação do algoritmo de Gale-Shapley, temos:

$$\boldsymbol{O((n+m)^2)}.$$

Dessa forma, a complexidade total do sistema é dada por:
Melhor caso:

$$\boldsymbol{Max(O(n+m), O(n), O(n \cdot m), O((n+m)^2) = O((n+m)^2)}$$

Pior caso:

$$\boldsymbol{Max(O(n+m), O(n^2), O(n \cdot m^2), O((n+m)^2) \ = \ O((n+m)^2)}$$

4. Conclusão

Após a leitura do problema proposto, nota-se o uso essencial da modelagem por casamento estável, o qual proporciona facilidade e velocidade na hora da implementação. Nota-se também que o algoritmo de Gale-Shapley possui

complexidade O(n²), no entanto, na hora de adapta-lo para o problema proposto, o mesmo ficou com uma complexidade maior de $\boldsymbol{O}((\boldsymbol{n}+\boldsymbol{m})^{\boldsymbol{2}})$. Talvez com um estudo mais detalhado da implementação seja possível diminuir a complexidade do sistema para $\boldsymbol{O}(\boldsymbol{n}^{\boldsymbol{2}})$.

Ao finalizar o sistema, ficou evidente a necessidade de se conhecer bem as estruturas de dados básicas, como filas, pilhas, lista, arvores e grafos. Uma vez que, o bom uso delas pode melhorar muito a implementação e o tempo de execução do sistema.

5. Execução

O programa foi desenvolvido no Windows 10 e testado no Windows 10 e no Linux.

Para execução no Windows, entre no diretório onde está o Makefile e execute os seguintes comandos:
mingw32-make
tp01

Para execução no Linux entre no diretório onde está o Makefile e execute os seguintes comandos:
make
./tp01

Observação: Os testes deixados pelo professor foram testados usando o comando:
./tp01 < caso_teste_01.txt

References

Chaimowicz, Luiz e Prates, Raquel. Slide Lista Arranjo e Lista Encadeada de Pesquisa (2020). Disciplina Estrutura de Dados DCC205 UFMG.

Almeida, Jussara. Slide Algoritmos 1 Stabgle Matching (2021). Disciplina ALgortimos 1 UFMG.

Kleinberg, Jon. Livro Algorithm Design.

Post sobre a estrutura map. Disponível em https://www.delftstack.com/pt/howto/cpp/how-toiterate-over-map-in-cpp/ 01 nov. 2017. Acesso em: 18 nov. 2021.

TAD-ORDENACAO. GitHub: Jean Lucas Almeida Mota, 10 set. 2020. Disponível em: https://github.com/jeanhardzz/TAD-Ordenacao . Acesso em: 18 nov. 2021.