Trabalho Prático 3 – Algoritmos I

Problema do custo mínimo para chegar ao destino final

Gilliard Gabriel Rodrigues
Matrícula: 2019054609

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

gilliard2019@ufmg.br

1. Introdução

O problema proposto foi implementar um sistema que, dadas as informações sobre o número de escalas a serem pegas, a quantidade máxima de escalas com descontos cumulativos possíveis, o intervalo máximo para a aplicação dos descontos cumulativos, assim como os valores de desconto de cada escala, além do preço de cada uma e o tempo necessário para o translado entre elas, retornasse o custo mínimo para que João da Silva viajasse até o seu destino final. Baseado nisso, será apresentada nesta documentação uma solução que resolva o problema, fazendo uso de programação dinâmica.

2. Modelagem do problema

A fim de implementar uma solução para o problema, foram criadas duas classes: "Escala" e "Sistema", sendo a primeira para representar uma escala, com custo e tempo, já a segunda, para representar o sistema que trabalha com os vários métodos utilizados para receber e tratar as entradas, além de calcular o custo mínimo para alcançar o destino final. Para o algoritmo que calcula o custo mínimo, uma abordagem *top-down* de programação dinâmica foi utilizada. Seja *i* o índice da escala atual e *j*, o índice da escala em que o desconto começou, OPT(i, j) representa o custo de ir da escala atual até a escala final dado que seu desconto começa em *j* e a equação de recorrência pode ser visualizada na figura 1.

$$\mathrm{OPT}(i, j) = \begin{cases} 0\text{, se } i \text{ representa a escala final.} \\ \infty\text{, se } (i - j) \geq D \text{ ou } (\text{tempoAcumulado}[i] - \text{tempoAcumulado}[j]) \geq T. \\ (\text{fatorMultiplicativoDeDesconto}[i - j] * \text{vetorDeEscalas}[i].\text{obter\_custo}()) + \min(\mathrm{OPT}[i + 1][i + 1], \mathrm{OPT}[i + 1][j])\text{, caso contrário.} \end{cases}$$

Figura 1. Equação de recorrência

3. Implementação

O programa foi desenvolvido na linguagem C++, compilada pelo compilador G++ da GNU Compiler Collection.

2.1 Estrutura de Dados e Algoritmos

A estrutura de dados utilizada foi a *std::vector* da biblioteca <*vector*> do C++, a fim de armazenar as escalas, os descontos recebidos da entrada, o fator multiplicativo de desconto, o tempo e o tempo acumulado. Quanto aos algoritmos, foram utilizados o *setprecision()*, o *partial_sum()* e o *min()* das bibliotecas <*iomanip*>, <*numeric*> e

<*algorithm*> do C++, respectivamente. Além, é claro, dos métodos implementados no TP.

2.2 Classes e seus métodos

A fim de seguir as boas práticas de programação e implementar um código modularizado, o programa foi dividido em 5 arquivos: "escala.h", "escala.cpp", "sistema.h", "sistema.cpp" e "main.cpp".

A classe *Escala* possui 2 atributos privados, são eles: um inteiro *tempo* (para representar o tempo gasto no translado da escala) e outro inteiro *custo* (representando o custo do bilhete dessa escala). Além disso, há também um construtor (público), que recebe como parâmetro dois inteiros e inicializa os atributos *tempo* e *custo* com esses parâmetros. Por fim, essa classe também possui 2 métodos *getters* (públicos), que retornam o valor de cada atributo da classe.

Já a classe *Sistema* possui 8 atributos privados, são eles: um inteiro *N* (representando a quantidade de escalas a serem percorridas até chegar ao destino final), um inteiro *D* (representando a quantidade máxima de escalas com descontos cumulativos no intervalo T), um inteiro *T* (representando o intervalo máximo para aplicação dos descontos acumulativos), um *vector* do tipo *Escala* chamado *vetorDeEscalas* (que, como o próprio nome já diz, armazena as escalas), um *vector* de inteiros chamado *desconto* (que armazena os D descontos recebidos como entrada), um *vector* de *double* chamado *fatorMultiplicativoDeDesconto* (que armazena o valor pelo qual o preço da escala *i* deve ser multiplicado para obter o desconto acumulado), um *vector* de inteiros chamado *tempoAcumulado* (que, como o próprio nome já diz, armazena o tempo de translado acumulado até a escala *i*) e, por fim, um ponteiro para uma matriz de *double* com nome *matrizDeMemoria* (que representa a matriz de custos).

Além dos 8 atributos privados, a classe *Sistema* possui um construtor público, que inicializa os inteiros *N*, *D* e *T* de acordo com os parâmetros recebidos, além de alocar memória dinamicamente para a matriz de custos, que deve ser $(N+1)$ x $(N+1)$. Além disso, há também 5 métodos públicos, todos sem parâmetros, que serão explicados separadamente a seguir.

O primeiro método chama-se *recebeDescontos* e trata-se de um método *void*, que popula o vetor *desconto* com a entrada do usuário.

O segundo método chama-se *calculaComplementoDoDescontoAcumulado* e trata-se de um método *void*, que popula o vetor *fatorMultiplicativoDeDesconto* da seguinte forma: o método *std::partial_sum* (da biblioteca <*numeric*>) é chamado e monta um vetor com a soma acumulada dos descontos e, em seguida, o complemento de cada percentual de desconto é atribuído às posições do vetor a ser populado.

O terceiro método chama-se *instanciaEscala* e trata-se também de um método *void*, que popula o vetor de escalas com a entrada do usuário.

O quarto método chama-se *calculaTempoAcumulado* e trata-se de um método *void*, que popula o vetor *tempoAcumulado* com as somas acumuladas dos tempos até cada escala (novamente usando o *std::partial_sum*).

O quinto método chama-se *calculaCustoMinimo* e trata-se de um método *double*, que de fato calcula o custo mínimo para que João da Silva chegue ao seu destino final.

Dentro desse método são chamados os métodos *calculaTempoAcumulado()* e *calculaComplementoDoDescontoAcumulado()*, em seguida é definida a precisão de 2 casas decimais e há dois *for*'s aninhados, que populam a matriz de custos de acordo com as condições da equação de recorrência apresentada anteriormente, caminhando de trás para frente e somente na parte triangular inferior da matriz, uma vez que somente essa parte precisa ser populada e, portanto, percorrê-la dessa forma economiza tempo. Após o processo de preenchimento da matriz triangular inferior acabar, o método retorna o conteúdo de *matrizDeMemoria[0][0]*, que nada mais é do que o custo mínimo do qual João da Silva precisa.

Por fim, há também um destrutor que desaloca a memória alocada dinamicamente para a matriz de custos.

Na figura 2 é possível visualizar o pseudocódigo do método que calcula o mínimo.

```
calculaCustoMinimo()
________________________________________________________________________________

PARA i = N ATÉ 0
    PARA j = i ATÉ 0
        SE ((número de escalas com desconto ≥ D) || (tempo acumulado desde que o desconto começou ≥ T))
            M[i][j] ← ∞
        SENÃO SE (i = N)
            M[i][j] ← 0
        SENÃO
            M[i][j] ← custo com desconto + min {M[i+1][i+1], M[i+1][j]}
RETORNA M[0][0]
________________________________________________________________________________
```

Figura 2. Pseudocódigo do método que calcula o custo mínimo

2.3 Estrutura do main

No *main()*, ocorrem as declarações de variáveis, além da leitura da primeira linha, composta pelo *N*, *D* e *T*, a instanciação de um objeto do tipo *Sistema*, que recebe como parâmetro os inteiros lidos da primeira linha, seguido pelas chamadas dos métodos *recebeDescontos*, *instanciaEscala* e *calculaCustoMinimo*, com o resultado da chamada desse último método sendo atribuída à variável *resultado* e sendo exibido na saída.

2.5 Instruções de compilação e execução

Para compilar o trabalho, acesse o diretório dos arquivos do TP via terminal e digite "*make*". Em seguida, para executar o código, digite "*./tp03*".

Logo após, na primeira linha, insira a quantidade de escalas a serem percorridas até chegar ao destino final (inteiro N), seguida pela quantidade máxima de escalas com descontos cumulativos no intervalo T (inteiro D) e, por fim, o intervalo máximo para a aplicação dos descontos cumulativos (inteiro T), separados por espaço. Já na segunda linha, insira os D inteiros representando o desconto percentual cumulativo para cada escala em diante no valor do bilhete, até a D-ésima escala, separados por espaço e, nas próximas N linhas, insira 2 inteiros separados por espaço, representando o tempo do translado até a escala i e o custo do bilhete dessa mesma escala.

Para ler a entrada de um arquivo, basta digitar "*./tp03 < nomeDoArquivo.txt*".

4. Análise de complexidade de tempo

Para chegar a uma conclusão sobre o custo de complexidade do programa, serão analisadas as chamadas da função *main()*, passo a passo. Seja *N* o número de escalas a serem percorridas e D, a quantidade máxima de escalas com descontos cumulativos dentro do intervalo T.

A primeira chamada trata-se do método *recebeDescontos()*. Esse método possui apenas um *for* que itera D vezes lendo a entrada do usuário. Portanto, sua complexidade de tempo é da ordem de $O(D)$.

A segunda chamada trata-se do método *instanciaEscala()*. Esse método possui um *for* que itera N vezes lendo as entradas do usuário. Portanto, sua complexidade de tempo é da ordem de $O(N)$.

A terceira chamada trata-se do método *calculaCustoMinimo()*. Dentro desse método são chamados os métodos *calculaTempoAcumulado* e *calculaComplementoDoDescontoAcumulado*. O primeiro deles possui complexidade de tempo da ordem de $O(N)$, uma vez que o custo é dominado pelo *for* que itera N vezes, já o segundo possui complexidade de tempo da ordem de $O(D)$, pois o custo é dominado pelo for que itera D vezes. Voltando à análise das outras linhas de *calculaCustoMinimo()*, temos um *for* aninhado com outro *for*, que embora percorra apenas metade da matriz, sua complexidade assintótica de tempo é da ordem de $O(N^2)$. Portanto, o método é dominado pela complexidade do processo de popular a matriz e isso faz com que a complexidade de tempo seja da ordem de $O(N^2)$.

Portanto, analisando todos esses métodos, conclui-se que a complexidade assintótica de tempo que domina o programa é a do método que calcula o custo mínimo, da ordem de $O(N^2)$.

5. Conclusão

Os pontos mais importantes para conseguir implementar o sistema foram, sem dúvidas, descobrir a melhor forma de armazenar e tratar cada informação, encontrar a equação de recorrência e, assim, conseguir implementar um algoritmo que a seguisse.

No decorrer desse trabalho, ficou evidente a importância de fazer testes para ter certeza de que o algoritmo estava funcionando de forma correta. Por fim, foi dada uma atenção especial ao processo de refatoração e modularização do código a fim de atender às boas práticas de programação e facilitar a identificação e correção de erros.

Em suma, foi possível praticar o uso de programação dinâmica, complementando o aprendizado adquirido durante as aulas.