TP03 - O novo metrô da Big Apple

Rita Rezende Borges de Lima - 2020065317

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

ritaborgesdelima@dcc.ufmg.br

1. Introdução

O problema proposto nesse trabalho consiste em encontrar o menor valor necessário para percorrer cada uma dos trechos de metrô passados pelo usuário. Contudo, existem outros dados a serem considerados devido a um sistema de desconto cumulativo para as D escalas consecutivas que se encontram dentro de um limite de tempo T. Assim dado o preço de cada escala e possíveis descontos aplicadas a cada uma, temos que achar o menor valor possível dos preços somados com descontos aplicados.

2. Modelagem

O problema pode ser reduzido para a seguinte escolha: em cada escala o passageiro pode continuar o trajeto indo para a próxima escala ou esperar até que o tempo do ciclo de descontos termine e outro ciclo se inicie. Esta escolha faz diferença no preço final pois cada trecho possui um preço diferente de forma que descontos maiores são preferíveis em escalas mais caras.

3. Implementação

3.1. Algoritmo

A príncipio a primeira coisa feita é encontrar a porcentagem do valor que será pago ao aplicarmos os descontos. Duas iterações no vetor ocorrem, uma aplica o desconto cumulativo:

$$desconto[i] = desconto[i] + desconto[i-1]$$

Caso esse desconto ultrapasse cem por cento, apenas igualamos este à cem. Depois calculamos qual a porcentagem do valor inicial que sobrará ao aplicarmos o desconto da seguinte maneira:

$$desconto[i] = (100.0 - desconto[i])/100.0$$

Realizando estas contas basta multiplicar o valor do respectivo desconto ao preço e teremos o valor de quanto cada pessoa tem que pagar naquele trecho de trem.

Com o intuito de encontrar o menor custo acumulado, é utilizada uma tabela contendo tuplas compostas pelo custo intermediário e tempo gasto após cada trecho para cada um dos possíveis descontos. Estes valores são armazenados em uma matriz M onde cada linha representa uma escala e cada coluna um desconto. Cada célula da tabela, (i, j) armazenará o valor diagonal superior esquerdo (i-1, j-1) adicionado dos respectivos tempos e custos (com desconto) daquela posição. A única exceção a regra é a primeira coluna que armazena no tempo valor nulo e o custo da posição com desconto somado ao menor preço da linha anterior. Assim iniciando da posição i = 0 e j = 0, preenchemos toda a matriz com base em valores anteriores. Ao final teremos D opções de preço para a soma detodos os trajetos na última linha, de forma que basta retornar o menor destes.

3.2. Exemplo da Matriz de Tuplas e sua Respectiva Entrada

Entradas:
3 3 5 0 50 0 4 5 3 10 5 9

Preço \Desconto	0	50/100	50/100
5	(4, 5.00)	(0, 0.00)	(0,0.00)
10	(3, 15.00)	(7,10.00)	(0,0.00)
9	(5,19.00)	(8,19.50)	(-1, ∞)

3.3. Pseudocódigo da solução

Algorithm 1: Busca menor valor para percurso

input : Descontos: desc[d], Escalas: esc<tempo,custo>[n], Tempo Máximo: t
output: Menor preço: p

```
memo<tempo, custo>[n][d], menorVal[n]
memo[0][0] ← <esc[0].f, esc[0].s * desc[0]>
menorVal ← 0

for i = 1; i < n; i++ do
    memo[i][0] ← <esc[i].f, esc[i].s * desc[0] + menorVal[i-1]>
    for j = 1; j < d and j ≤ i; j++ do
        if memo[i-1][j-1].f < t and memo[i-1][j-1].f ≥ 0 then
            memo[i][j] ← memo[i-1][j-1] + <esc[i].f, esc[i].s * desc[j]>
            if memo[i][menorVal[i]].s > memo[i][j].s then
                menorVal[i] ← j
        else memo[i][j] = < −1, ∞ >
return menorVal[n-1]
```

4. Análise de Complexidade

4.1. Tempo

Existem três operações no projeto, a leitura, o reajuste dos descontos e o cálculo de menor valor a ser pago.

- **Leitura:** A leitura dependerá exclusivamente da quantidade de escalas, N, e de descontos, D. Primeiro, a leitura de N, D e T ocorre seguida da leitura de d inteiros em uma linha. Por fim n linhas com dois inteiros são processadas. Dessa forma é possível afirmar que a complexidade da função de leitura é linear e da forma $O(N + D)$

- **Reajuste dos descontos:** Com o intuito de já ter em mãos o valor que deve ser multiplicado pelo preço, iteramos duas vezes no vetor de descontos, a primeira transforma este em cumulativo e a segunda armazena a diferença entra 1 e o valor armazenado. O vetor de descontos possui tamanho D, dessa forma, são feitas 2*D iterações de maneira que a complexidade é linear e da forma $O(D)$.

- **Cálculo de menor valor:** A função que calcula o menor valor possui dois laços aninhados que em pior caso vão de 1 à N e 1 à D, assim sua complexidade é da forma $O(N * D)$. A função inclusive utiliza da técnica *Bottom-Up* da programação dinâmica, e podemos escrever a seguinte equação de recorrência para descrevê-la:

$$\text{OPT(e, d, t)} = \begin{cases} 0 & \text{se } t \geq T \text{ ou } d = D \text{ ou } e = N. \\ \text{Menor Custo da Escala Passada + e*d} & \text{se } t = 0. \\ \text{min(OPT(e + 1, d+1, t + t}_e\text{), OPT(e+1, 0, 0))} & \text{Caso contrário.} \end{cases}$$

A equação de recorrência é em função da escala, desconto e o tempo de cada iteração. Primeiro precisamos analisar se o tempo máximo foi ultrapassado ou se já percorremos todas as escalas ou descontos, caso contrário olhamos se o tempo acabou de ser zerado ou se continuamos em um mesmo ciclo de descontos.

Por fim analisando a complexidade do algoritmo como um todo observamos que na função main, chamamos cada uma das operações descritas apenas uma vez, o que ocasiona na complexidade geral: $O(n + d) + O(d) + O(n * d)$. Assim, a complexidade pode ser reduzida para a forma $O(n * d)$.

4.2. Espaço

No algoritmo todas as estruturas dependem de duas variáveis, a quantidade de pontos de escalas, n e a quantidade de descontos em uma escala, d.

- **Vetor de descontos em uma escala**: seu tamanho é sempre d, de maneira que sua complexidade espacial é da forma: $\theta(d)$.
- **Vetor de tuplas compostas pelos tempos e custos de cada escala**: seu tamanho é sempre n, de maneira que sua complexidade espacial é da forma: $\theta(n)$.
- **Matriz contendo as tuplas, tempo custo:** esta possui uma linha para cada possível escala e uma coluna para cada possível desconto, logo tem tamanho nxd, assim sua complexidade espacial é da forma: $\theta(n * d)$.
- **Um vetor contendo o índice do menor preço de cada linha da matriz de memorização:** este possui o tamanho da quantidade de linhas da matriz, n, de maneira que sua complexidade espacial é da forma: $\theta(n)$.

5. Conclusões

A partir da implementação do trabalho foi possível colocar em prática os conhecimentos adquiridos na disciplina a respeito de programação dinâmica, especialmente utilizando a abordagem *Bottom-Up*. Por meio do algoritmo proposto foi possível analisar todas as possibilidades de decisão a respeito de quanto tempo esperar em cada terminal de trem e encontrar o menor custo total dos preços de bilhete de acordo com os respectivos descontos. Também foi possível perceber a grande diferença de performance entre um algoritmo utilizando programação dinâmica, que nesse caso teve complexidade n*d, comparada a uma implementação força bruta para o mesmo problema, que teria complexidade exponencial.

6. Bibliografia

- Ziviani, N. (2006).Projetos de Algoritmos com Implementações em Java e C++ Editora Cengage
- Kleinberg, Jon; Tardos, Eva. Algorithm Design, Pearson Education India, 2006