Universidade Federal de Ouro Preto
PCC104 - Projeto e Análise de Algoritmos

Lista 5 - Programação Dinâmica

Aluno: Lucas Andrade Freitas

Questão 3

Problema: Implemente um algoritmo para o problema do troco (Change-making problem (Seção 8.1)) utilizando programação dinâmica.

Para cada implementação, apresentar a análise de complexidade de tempo do algoritmo.

Solução:

1) Código:

```cpp
int coinChange(std::vector<int>& coins, int amount) {
    std::vector<int> dp(amount + 1, INT_MAX);
    dp[0] = 0;

    for (int count = 1; count <= amount; count++) {
        for (int coin : coins) {
            if (count - coin >= 0) {
                dp[count] = std::min(dp[count], dp[count - coin] + 1);
            }
        }
    }

    if (dp[amount] == INT_MAX) {
        return -1;
    }

    return dp[amount];
}
```

2) Operação básica:

A operação básica seria a comparação para encontrar a menor quantidade de moedas para o troco. Definida por:

```
dp[count] = std::min(dp[count], dp[count - coin] + 1);
```

3) Equação de recorrência:

Seja C(i) o custo mínimo para formar a quantia i, e seja M um conjunto de moedas disponíveis.

C(i) = min { C(i - m) + 1 | para cada m em M e i - m ≥ 0 }

Nesta expressão, C(i - m) + 1 representa o custo mínimo para formar a quantia i - m (valor anterior) adicionado ao custo de incluir uma moeda m na solução.

A função min{} retorna o valor mínimo entre todas as combinações de C(i - m) + 1 para todas as moedas m em M, desde que i - m seja maior ou igual a zero.

Para entender melhor essa relação, podemos simular o funcionamento do código para um **Amount = 7** e com as seguinte moedas disponíveis: 1, 5, 10, 20.

Caso base: F(0) = 0
F(1) = min {F(1-1)} + 1 = 1
F(2) = min {F(2-1)} + 1 = 2
F(3) = min {F(3-1)} + 1 = 3
F(4) = min {F(4-1)} + 1 = 4
F(5) = min{F(5 -1), F(5 - 5)} + 1 = 1 Obs: Esse caso se diferencia dos outros porque existe opção de pegar o valor antigo que seria o referente ao valor do amount 4 ou o uma moeda de 5, pois no nosso conjunto existe uma moeda de 5, que será a escolhida por resulta no valor mínimo.

F(6) = min {F(6-1)} + 1 = 2
F(7) = min {F(7-1)} + 1 = 3

Logo a resposta para um **amount** 7, será 3 moedas.

4) Complexidade:

Complexidade de tempo: O(m * n), onde n = amount, m = size(coins)
Complexidade de espaço: O(n)

- Melhor caso: Solução ótima em somente uma operação, por exemplo, troco = 1 com 1 moeda.
- Pior caso: Pior solução, por exemplo, troco = 4 com 4 moedas.

Pertence à classe P, pois pode ser resolvido em tempo polinomial.

Questão 4

Problema: Problema: Implemente um algoritmo para o problema de coleta de moedas (Coin-collecting problem (Seção 8.1)) utilizando programação dinâmica.

Para cada implementação, apresentar a análise de complexidade de tempo do algoritmo.

Solução:

1) Código:

```
int collectCoins(std::vector<std::vector<int>>& grid, int row, int col) {
    // Verifica se está fora dos limites da grade
    if (row >= grid.size() || col >= grid[0].size()) {
        return 0;
    }

    // Verifica se chegou à célula de destino (canto inferior direito)
    if (row == grid.size() - 1 && col == grid[0].size() - 1) {
        return grid[row][col];
    }

    // Calcula a quantidade máxima de moedas coletadas ao mover para baixo e para a direita
    int coinsDown = collectCoins(grid, row + 1, col);
    int coinsRight = collectCoins(grid, row, col + 1);

    // Retorna o valor máximo acumulado ao escolher o caminho com mais moedas
    return grid[row][col] + std::max(coinsDown, coinsRight);
}
```

2) Operação básica:

```
// Retorna o valor máximo acumulado ao escolher o caminho com mais moedas
return grid[row][col] + std::max(coinsDown, coinsRight);
```

A operação básica seria a comparação entre se o caminho à direita ou a esquerda que traz o maior valor de moeda, junto do somatório da posição atual.

3) Equação de recorrência:

Seja C(i, j) a quantidade máxima de moedas coletadas ao chegar na posição (i, j) na grade.

C(i, j) = grid[i][j] + max(C(i+1, j), C(i, j+1))

Essa relação de recorrência indica que a quantidade máxima de moedas coletadas ao chegar na posição (i, j) é igual à quantidade de moedas na posição (i, j) somada ao máximo entre a quantidade máxima de moedas coletadas ao chegar na posição abaixo (i+1, j) e a quantidade máxima de moedas coletadas ao chegar na posição à direita (i, j+1).

4) Complexidade:

Complexidade de tempo: O(n^2), dado que a matriz seja quadrada ou O (m * n)
Complexidade de espaço:O(n^2), dado que a matriz seja quadrada ou O (m * n)

Creio que pior caso e melhor caso não se aplique neste algoritmo, pois de qualquer maneira ele vai passar por toda matriz.

Pertence à classe P, pois pode ser resolvido em tempo polinomial.

Questão 5

Problema: Implemente um algoritmo para o problema de coleta de moedas (Coin-collecting problem (Seção 8.1)) sem utilizar programação dinâmica.

Solução:

```cpp
int collectCoins(vector<vector<int>>& grid) {
    int rows = grid.size();
    int cols = grid[0].size();

    // Cria uma matriz para armazenar os resultados intermediários
    vector<vector<int>> dp(rows, vector<int>(cols, 0));

    // Inicializa o último elemento da matriz dp com o valor da última célula da grade
    dp[rows-1][cols-1] = grid[rows-1][cols-1];

    // Preenche a última coluna da matriz dp
    for (int i = rows - 2; i >= 0; i--) {
        dp[i][cols-1] = dp[i+1][cols-1] + grid[i][cols-1];
    }

    // Preenche a última linha da matriz dp
    for (int j = cols - 2; j >= 0; j--) {
        dp[rows-1][j] = dp[rows-1][j+1] + grid[rows-1][j];
    }

    // Preenche o restante da matriz dp de baixo para cima e da direita para a esquerda
    for (int i = rows - 2; i >= 0; i--) {
        for (int j = cols - 2; j >= 0; j--) {
            dp[i][j] = grid[i][j] + max(dp[i+1][j], dp[i][j+1]);
        }
    }

    // Retorna o valor máximo acumulado na posição inicial (0, 0)
    return dp[0][0];
}
```

2) Operação básica:

```cpp
dp[i][j] = grid[i][j] + max(dp[i+1][j], dp[i][j+1]);
```

3) Equação de recorrência:

$$F(i,\ j) = max\{F(i-1,\ j),\ F(i,\ j-1)\} + c_{ij} \quad \forall\ 1 \leq i \leq n,\ \ 1 \leq j \leq m$$
$$F(0,\ j) = 0 \quad \forall\ 1 \leq j \leq m \quad e \quad F(i,\ 0) = 0 \quad \forall\ 1 \leq i \leq n,$$

onde $c_{ij} = 1$ se houver uma moeda na célula (i, j), e $cij = 0$ caso contrário.

4) Complexidade:

Complexidade de tempo: O(n^2), dado que a matriz seja quadrada ou O (m * n)
Complexidade de espaço:O(n^2), dado que a matriz seja quadrada ou O (m * n)

Creio que pior caso e melhor caso não se aplique neste algoritmo, pois de qualquer maneira ele vai passar por toda matriz.

Pertence à classe P, pois pode ser resolvido em tempo polinomial.