Aluno: Gustavo Martins Ribeiro

Disciplina: Projeto de Algoritmos (FGA-UNB)

Tema: Programação Dinâmica

EXERCÍCIOS – PROGRAMAÇÃO DINÂMICA

1210) Produção Ótima de Ótima Vodka (USP)

O problema trata da produção de Vodka na cidade de São Petersburgo, que usa um destilador com vida útil limitada. O objetivo é determinar o custo mínimo de produção de Vodka ao longo de N anos, considerando o custo de manutenção anual do destilador e o valor de venda do destilador em diferentes idades. O destilador pode ser trocado no início de cada ano, e a solução deve indicar em quais anos a troca foi feita. A entrada inclui as informações sobre N, I, M, P, custo de manutenção e valor de venda do destilador em diferentes idades. A saída deve ser o custo mínimo e a sequência de trocas de destiladores.

>> Sobre a solução apresentada (Arquivo 1210_ProducaoOtimaDeOtimaVodka.c)

A solução de código apresentada implementa o algoritmo de Programação Dinâmica. Isso é evidente pela construção de uma tabela (**S**) de soluções e outra (**T**) de decisões, que são preenchidas através da resolução de subproblemas. O objetivo é encontrar a solução ótima para o problema por meio da combinação dessas soluções subótimas. Além disso, é possível ver a utilização do recurso de Memorização, pois as soluções para subproblemas são armazenadas para evitar o cálculo repetido dessas soluções.

1517) Maçãs (UTFPR)

Rafael está na fazenda de seu tio e vai fazer uma torta de maçã. Seu primo vai derrubar maçãs da árvore e Rafael tem que pegar o máximo possível delas antes que elas quebrem no chão. Rafael pode se mover na área abaixo da árvore, que tem N linhas e M colunas, e cada maçã tem uma posição específica [i, j] e um tempo específico tem que Rafael tem que estar naquela posição para pegar a maçã. A entrada vai ser informada com N, M e K, que são a quantidade de linhas e colunas da área e o número de maçãs, respectivamente. Em seguida, haverá as informações de posição e tempo de cada maçã, e a posição inicial de Rafael. A saída será o número máximo de maçãs que Rafael consegue pegar.

>> Sobre a solução apresentada (Arquivo 1517_Maçãs.cpp)

A solução apresentada usa a programação dinâmica para resolver o problema de encontrar o número máximo de maçãs que podem ser pegos num grid dado a posição inicial e a posição e tempo de colocação das maçãs. A programação dinâmica é realizada através do uso de uma matriz tridimensional (dp[i][j][k]) que armazena o número máximo de maçãs que podem ser pegos a partir da posição (j,k) no tempo i. A solução itera pelo tempo em ordem decrescente e, para cada posição no tempo atual, verifica as posições adjacentes no próximo tempo e escolhe o maior valor. Além disso, a solução também verifica se há uma maçã na posição (linha, coluna) no tempo atual e adiciona +1 ao resultado se houver.

1522) Jogo das Pilhas

Claudio criou um jogo chamado "Jogo das Pilhas" e quer enviá-lo para um concurso de jogos. O jogo é jogado com três pilhas, cada uma com o mesmo número de cartas, e cada carta tem um valor numérico de 0 a 9. O jogador pode ver o valor de qualquer carta a qualquer momento, mas só pode jogar com as cartas no topo das pilhas. A cada rodada, o jogador deve remover cartas cuja soma seja múltipla de 3. O jogo é vencido quando todas as cartas são removidas. Se alguma carta não puder ser removida, o jogo é perdido. O programa deve ler a quantidade de cartas em cada pilha e os valores das cartas em cada pilha, e determinar se o jogador pode vencer ou não.

>> Sobre a Solução apresentada - (Arquivo 1522_JogoDasPilhas.cpp)

A solução apresentada utiliza a técnica de programação dinâmica para resolver o problema de maximizar o número de maçãs que podem ser retiradas de pilhas, desde que o total retirado seja múltiplo de três. A função **solve** é a função responsável por realizar este cálculo. A solução usa três vetores para armazenar as pilhas, **mys[0]**, **mys[1]** e **mys[2]**, e uma matriz tridimensional, **dp[i][j][k]**, para armazenar os valores já calculados, evitando assim o cálculo repetido. A solução também utiliza uma macro, **FOR**, para facilitar a escrita de laços de repetição.
