Três Problemas Básicos

Coin-row

Há uma fila de n moedas de valores são inteiros positivos C1, C2,...,Cn, não necessariamente distintos. O objetivo é pegar a quantidade máxima de dinheiro com restrição de que duas moedas adjacentes na fila inicial não podem ser escolhidas.

Seja F(n) a quantidade máxima que pode ser escolhida da fila de n moedas. Para derivar uma recorrência para F(n), divide-se todas as seleções permitidas em dois grupos: as que têm a última moeda e as sem ela. A maior quantidade que podemos obter do primeiro grupo é Cn + F(n - 2). A quantidade máxima do segundo grupo é igual a F(n - 1) pela definição de F(n).

Assim, temos a seguinte recorrência sujeita às condições iniciais:

- F(n) = max{Cn + F(n - 2), F(n - 1)} para n > 1,
- F(0) = 0, F(1) = C1.

Podemos calcular F(n) preenchendo a tabela de uma linha da esquerda para a direita.

Change-making

Dar troco para o valor n usando o menor número possível de moedas de denominações: d1 < d2 < ... < dm. Consideramos um algoritmo de programação dinâmica para o caso geral, assumindo uma quantidade ilimitadas de moedas, onde d1 = 1.

Seja F(n) o número mínimo de moedas cuja soma é n. F(0) = 0. O valor n só pode ser obtido adicionando uma moeda de denominação dj ao valor n - dj para j = 1, 2,...,m, desde que n ≥ dj. Portanto, selecionamos aquela que minimiza F(n - dj) + 1. Uma vez que 1 é uma constante, podemos encontrar o menor F(n - dj) primeiro e então adicionar 1 a ele.

Assim, temos a seguinte recorrência para F(n):

- F(n) = min{F(n - dj) + 1 : n ≥ dj} para n > 0,
- F(0) = 0.

Podemos calcular F(n) preenchendo uma tabela de uma linha da esquerda para a direita, mas calcular uma entrada da tabela requer encontrar o mínimo de até m números.

Coin-collecting

Várias moedas são colocadas em células de um tabuleiro n × m, com no máximo uma moeda por célula. Um robô, localizado na célula superior esquerda, precisa coletar o maior número possível de moedas e levá-las para a célula inferior direita. Em cada etapa, o robô pode mover-se para a direita ou para baixo. Quando o robô visita uma célula com uma moeda, ele a recolhe. Para encontrar o número máximo de moedas que o robô pode coletar e o caminho para fazer isso.

Seja F(i, j) o maior número de moedas que o robô pode coletar e trazer para a célula (i, j). Ele pode alcançar essa célula tanto a partir das célula adjacentes (i - 1, j) e (i, j - 1). Os maiores números de moedas que podem ser trazidos para essas células são F(i - 1, j) e F(i, j - 1), respectivamente.

Como não há células adjacentes acima das células na primeira linha e à esquerda das células na primeira coluna. Para elas, assumimos que F(i - 1, j) e F(i, j - 1) são iguais a 0, para seus vizinhos inexistentes. Logo, o maior número de moedas que o robô pode trazer para a célula (i, j) é o máximo desses dois números mais uma possível moeda na própria célula (i, j). Temos a seguinte fórmula para F(i, j):

- F(i, j) = max{F(i - 1, j), F(i, j - 1)} + Cij para 1 ≤ i ≤ n, 1 ≤ j ≤ m,
- F(0, j) = 0 para 1 ≤ j ≤ m e F(i, 0) = 0 para 1 ≤ i ≤ n,

onde Cij = 1, se houver uma moeda na célula (i, j), senão Cij = 0.
Assim, podemos preencher a tabela n × m de valores de F(i, j)

Programação Dinâmica

O problema da mochila

Dado n itens com pesos conhecidos w1, ... ,wn e valores v1, ... ,vn e uma mochila com capacidade W, encontre o subconjunto mais valioso dos itens que se cabem na mochila.

Vamos considerar uma instância definida pelos primeiros i itens, 1 ≤ i ≤ n, com pesos w1, ... ,wi, valores v1, ... ,vi e capacidade da mochila j, 1 ≤ j ≤ W. Seja F(i, j) o valor do subconjunto mais valioso dos primeiros i itens que cabem na mochila de capacidade j. Podemos dividir todos os subconjuntos dos primeiros i itens que cabem na mochila de capacidade j em duas categorias: aqueles que não incluem o i-ésimo item e aqueles que incluem.

Observe o seguinte:

- Entre os subconjuntos que não incluem o i-ésimo item, o valor de um subconjunto ótimo é, por definição, F(i - 1, j).
- Entre os subconjuntos que incluem o i-ésimo item, um subconjunto ótimo é composto por este item e um subconjunto ótimo vi + F(i - 1, j - wi) dos primeiros i - 1 itens que cabem na mochila de capacidade j - wi.

Funções de Memória

Este método resolve um problema dado com top-down, e mantém uma tabela do tipo que seria usada por um algoritmo de programação dinâmica bottom-up. Inicialmente, todas as entradas da dela são têm um símbolo especial "NULL" que indica que ainda não foram calculadas.

Sempre que um novo valor precisa ser calculado, primeiro ele verifica a entrada correspondente na tabela: se a entrada não for "NULL", ela é recuperada da tabela, senão, é calculada pela chamada recursiva.

Em geral, não podemos esperar mais do que um ganho de fator constante ao usar o método da função de memória para o problema da mochila, porque sua eficiência de tempo é da mesma classe que a do algoritmo top-down.

Algoritmos de Aproximação para problemas NP-difíceis

Uma maneira de lidar com problemas de otimização difíceis é resolvê-los aproximadamente por meio de um algoritmo rápido.

Um algoritmo de aproximação em tempo polinomial é dito ser um algoritmo de c-aproximação, onde c ≥ 1, se a razão de precisão da aproximação não excede c para qualquer instância do problema em questão: r(Sa) ≤ c.

O menor valor de c é chamado de razão de desempenho do algoritmo e é denotado por RA.

Algoritmos de aproximação para o problema do caixeiro viajante

Os algoritmos de aproximação mais simples para o problema do caixeiro viajante são baseados na técnica gananciosa.

- O algoritmo do vizinho mais próximo.
- Algoritmo Multifragment-heuristic.

Algoritmos de Aproximação para o Problema da Mochila.

Algoritmos Gananciosos para o Problema da Mochila:

- Passo 1: Calcular as razões valor-peso ri = vi/wi, i = 1, . . . , n, dos itens dados.
- Passo 2: Ordenar os itens em ordem decrescente das razões calculadas.
- Passo 3: Repetir isso até que nenhum item esteja na lista ordenada: se o item atual na lista couber na mochila, colocá-lo na mochila e prosseguir para o próximo item, senão, apenas prosseguir para o próximo item.