Trabalho Prático 3 - Algoritmos I
Centro de Distribuição

Wallace Eduardo Pereira - 2021032013

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

wallaceep@ufmg.br

1. Introdução

O objetivo deste trabalho é otimizar o processo de distribuição de um centro de distração de ligas metálicas. O papel desse centro é separar as encomendas de cada cliente para o despache da transportadora. Para otimizar esse processo, é feito um algoritmo que faça com que o número de ligas usadas para suprir uma demanda seja a mínima possível, levando em conta os tamanhos de liga disponíveis no momento.

No caso específico desse problema, com a abordagem de programação dinâmica será possível otimizar o processo de distribuição de ligas metálicas, encontrando a quantidade mínima de ligas necessárias para atender a demanda de cada cliente. Isso resultará em uma redução de custos e uma melhoria na eficiência operacional do centro de distribuição.

Depois de implementar o algoritmo, é perceptível que o mesmo fica consideravelmente mais lento para ser executado com entradas significativamente maiores e tipos mais diversos. Isso ocorre porque o problema é do tipo NP-completo. A segunda parte desse trabalho é avaliar a complexidade de tal algoritmo e provar que se trata de um problema NP-completo.

2. Implementação

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

2.1. Programação Dinâmica

Para resolver o problema proposto, foi implementado um algoritmo utilizando programação dinâmica. A programação dinâmica se aplica quando os subproblemas se sobrepõem, isto é, quando os subproblemas compartilham subproblemas. Um algoritmo de programação dinâmica resolve cada subproblema só uma vez e depois grava a resposta em uma tabela, evitando assim, o trabalho de recalcular a resposta toda vez que resolver cada subproblema.

O desenvolvimento do algoritmo de programação dinâmica segue uma sequência de quatro etapas:

1. **Caracterizar a estrutura de uma solução ótima**

A solução ótima para o problema dado é determinar o número mínimo de ligas para suprir uma demanda, considerando as ligas disponíveis. Para representar tal solução, foi utilizado o

vetor dp, onde dp[i] armazena o número mínimo de ligas metálicas necessárias para atender uma demanda de tamanho i. Assim, o algoritmo preenche esse vetor de forma iterativa, calculando os valores ótimos para cada subproblema

2. **Definir recursivamente o valor de uma solução ótima**

A definição recursiva não é usada diretamente neste código, pois a abordagem escolhida é a de programação dinâmica de baixo para cima (iterativa). No entanto, a ideia principal é que o número mínimo de ligas metálicas necessárias para atender a uma demanda i é igual ao mínimo entre o valor já armazenado em dp[i] e o valor obtido ao usar uma liga metálica de tamanho ligas[j] para atender à demanda restante i - ligas[j] (caso seja possível usar a liga j).

3. **Calcular o valor de uma solução ótima, normalmente de baixo para cima**

A parte central da programação dinâmica é o loop duplo na função minLigas. Esse trecho percorre todos os valores possíveis de demanda i, de 1 até W, e para cada valor i, percorre todos os tipos de ligas disponíveis. Ele verifica se é possível usar a liga j para atender à demanda i, e caso seja possível, atualiza o valor de dp[i] para o mínimo entre seu valor atual e o valor obtido ao usar a liga j.

4. **Construir uma solução ótima com as informações calculadas**

No final do processo de cálculo, o vetor dp estará preenchido com o número mínimo de ligas metálicas necessárias para atender a cada demanda i, sendo dp[W] a resposta para o caso de teste atual. Esse valor representa o número mínimo de ligas metálicas necessárias para atender à demanda do cliente W, conforme solicitado no enunciado. O resultado é então impresso na saída.

O algoritmo é implementado utilizando programação dinâmica porque envolve a resolução de um subproblema de forma repetida e, em seguida, utiliza as soluções dos subproblemas para resolver o problema original.

A ideia da programação dinâmica aqui é encontrar o número mínimo de ligações para cada valor de demanda de 0 a W, armazenando essa informação no vetor dp. O valor em dp[i] representa o número mínimo de ligações necessárias para atender a demanda i.

3. NP-completude

Quando dizemos que um problema é NP-completo, fica subentendido que trata-se de um problema difícil de resolver (ou ao menos que achamos que é difícil), e não de um problema fácil de resolver. Não estamos tentando provar a existência de um algoritmo eficiente, mas que provavelmente não existe nenhum algoritmo eficiente.

Muitos problemas de interesse são problemas de otimização, para os quais cada solução possível (isto é, "válida") tem um valor associado e para os quais desejamos encontrar uma solução viável com o melhor valor. Porém, a NP-completude não se aplica diretamente a problemas de otimização, mas a problemas de decisão, para os quais a resposta é simplesmente "sim" ou "não" (ou, em linguagem mais formal, "1" ou "0").

Para provarmos que o problema em questão é um problema NP-completo, vamos reduzi-lo a um problema já conhecido NP-completo. Para tal, iremos utilizar o problema da mochila 0/1, reduzindo-o ao problema de ligas metálicas.

No problema da mochila, recebemos N itens onde cada item tem algum peso e lucro associado a ele. Também recebemos um saco com capacidade W, [ou seja, o saco pode conter no máximo W peso]. O objetivo é colocar os itens na bolsa de forma que a soma dos lucros associados a eles seja a máxima possível. Nota: A restrição aqui é que podemos colocar um item completamente na bolsa ou não podemos colocá-lo [Não é possível colocar uma parte de um item na bolsa].

Para reduzir o problema da mochila (Knapsack Problem) ao problema das ligas metálicas, devemos mostrar que uma instância do problema da mochila pode ser mapeada para uma instância do problema das ligas metálicas de tal forma que uma solução para o problema das ligas metálicas também é uma solução para o problema da mochila.

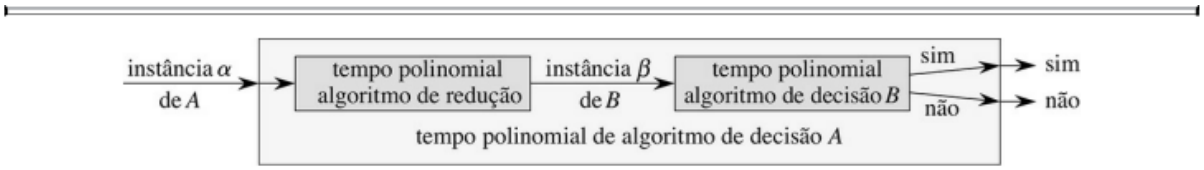

Figura 34.1 Como usar um algoritmo de redução de tempo polinomial para resolver um problema de decisão A em tempo polinomial, dado um algoritmo de decisão de tempo polinomial para um outro problema B. Em tempo polinomial, transformamos uma instância a de A em uma instância b de B, resolvemos B em tempo polinomial e usamos a resposta para b como a resposta para a.

Para reduzir o problema da mochila ao problema das ligas metálicas, devemos transformar os itens da mochila com seus respectivos pesos em uma liga com seus respectivos tamanhos e a capacidade da mochila na demanda dos clientes. Assim, o objetivo é encontrar o menor número de ligações metálicas necessárias para suprir a demanda, que é o tamanho da mochila.

Ao encontrarmos uma solução para o problema das ligas metálicas usando a função minLigas, podemos facilmente mapeá-la para uma solução válida para o problema da mochila. A liga metálica escolhida para atingir a demanda corresponderá ao subconjunto de itens que serão colocados na mochila a fim de maximizar o seu valor, mantendo o peso total dentro da capacidade da mochila.

4. Análise de Complexidade

Vamos analisar a complexidade da função minLigas. O loop externo executa W interações, e o loop interno executa N operações, sendo W a demanda e N o número de elementos no vetor ligas. As demais operações da função são de tempo constante. Portanto, a complexidade da função é de O(N*W).

Já na função main temos o loop que executa “numTestes” iterações, e um loop interno que executa uma operação para cada tipo de liga, denominado por “tipoLiga”. As demais operações são de custo constante, logo, a complexidade da função main é de O(numTestes * W).

O algoritmo é considerado pseudo-polinomial porque, embora a complexidade temporal do algoritmo seja expressa como O(N*W), ela não é estritamente polinomial em relação ao tamanho do conjunto de entrada (N e W).

A complexidade O(N*W) surge da abordagem de programação dinâmica utilizada, onde o algoritmo preenche um vetor de tamanho W+1, e em cada célula realiza uma quantidade constante de operações.

No entanto, a complexidade não depende apenas do valor de W e N, mas também do valor dos elementos presentes na entrada. No caso desse algoritmo, o tamanho da entrada é determinado por N (o número de tipos de ligações) e W (a demanda), mas o valor de cada elemento do vetor ligas também importa, pois eles influenciam diretamente a quantidade de iterações necessárias.

Assim, se os valores dos elementos em ligas forem muito grandes em comparação com N e W, a complexidade do algoritmo pode se tornar bastante alta, tornando-o não estritamente polinomial em relação ao tamanho da entrada. Isso contrasta com um problema polinomial típico, onde a complexidade é puramente determinada pelo tamanho da entrada e não pelos valores numéricos envolvidos.

Considerando agora a complexidade de espaço, na função minLigas temos o vetor de tamanho W+1 para armazenar os valores de programação dinâmica. Dessa forma, a complexidade de espaço da função é de O(W).

Na função main, a complexidade espacial é dada pelo vetor size, de tamanho da entrada tipoLiga, assumindo assim a complexidade da função O(tipoLiga).

5. Conclusão

O problema do centro de distribuição que consistia em otimizar o processo de distribuição, separando as encomendas de cada cliente para o despache da transportadora, de forma que o número de ligas usadas para suprir uma demanda fosse a mínima possível, foi resolvido com a utilização de programação dinâmica, de forma que evita recálculos redundantes por armazenar as soluções dos subproblemas em uma estrutura de dados.

Dado que o problema da mochila pode ser reduzido ao problema das ligas metálicas de forma polinomial (tempo polinomial é usado para transformar a entrada de um problema no outro), e uma solução para o problema das ligas metálicas é uma solução para o problema da mochila, concluímos que o problema das ligas metálicas é pelo menos tão difícil quanto o problema da mochila. Portanto, o problema das do centro de distribuição é NP-completo.

6. Referências

T. Cormen, C. Leiserson, R. Rivest e C. Stein. Introduction to Algorithms (Third Edition). The MIT Press.
https://www.geeksforgeeks.org/dynamic-programming/
https://www.geeksforgeeks.org/introduction-to-np-completeness/
https://www.geeksforgeeks.org/0-1-knapsack-problem-dp-10/