Trabalho Prático 3 da disciplina de Algoritmos 1

Alex Eduardo Alves dos Santos
Matrícula: 2021032145

Departamento de Ciência da Computação -
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil
alexeduardoaeas@ufmg.br

1. Introdução

O código apresentado implementa um algoritmo para otimizar o processo de distribuição de ligas metálicas em um centro de distribuição de uma fábrica. O objetivo é determinar o número mínimo de ligas necessário para atender a uma determinada demanda de cada cliente, levando em conta os tamanhos de ligas disponíveis no momento.

O algoritmo utiliza uma abordagem de programação dinâmica conhecida como "Troco de Moedas" (ou "Coin Change"). Ele constrói uma tabela para armazenar soluções intermediárias e determina o número mínimo de ligas para cada valor de demanda de forma iterativa. O algoritmo percorre os tamanhos de ligas disponíveis e verifica se é possível utilizar uma liga para suprir parte da demanda atual. Caso seja possível e essa opção resulte em um número menor de ligas do que o registrado na tabela, o valor mínimo é atualizado.

Essa abordagem garante que o número mínimo de ligas seja encontrado de maneira eficiente, evitando desperdício de tempo e recursos no centro de distribuição. Ao utilizar esse código, é possível agilizar o processo de separação das encomendas e o despacho da transportadora, reduzindo o total de ligas utilizadas para atender às demandas dos clientes. Ele fornece uma solução precisa e confiável para otimizar o fluxo de distribuição de ligas metálicas na fábrica, economizando recursos e maximizando a eficiência operacional.

2. Implementação/Modelagem

A função minimumLigas recebe a demanda do cliente (em metros) e um vetor ligas_disponiveis com os tamanhos de ligas que o centro de distribuição tem. Ela retorna o menor número de ligas que satisfazem a demanda.

Na função minimumLigas, criamos um vetor solucoes com demanda + 1 posições para guardar as soluções parciais. Todas as posições começam com INT_MAX para indicar que não temos solução ainda.

O caso base é: para demanda 0, o menor número de ligas é 0. Então, solucoes[0] = 0.

Usamos um loop duplo para preencher o vetor ‘solucoes’ usando programação dinâmica. O loop de fora vai de 1 até a demanda. O loop de dentro vai pelos tamanhos de ligas disponíveis.

Checamos se o tamanho da liga atual (ligas_disponiveis[j]) é menor ou igual à demanda atual (i). Se for, quer dizer que essa liga pode ser usada para atender parte da demanda.

Pegamos o menor número de ligas para atender à demanda que sobra (solucoes[i - ligas_disponiveis[j]]) e guardamos na variável subproblema.

Checamos se o subproblema é diferente de INT_MAX, o que significa que dá para atender à demanda que sobra. Além disso, checamos se subproblema + 1 (contando a liga atual) é menor que o valor atual em solucoes[i]. Se for, atualizamos solucoes[i] com esse novo valor mínimo.

Depois de preencher a tabela ‘solucoes’, o valor em solucoes[demanda] representa o menor número de ligas para atender à demanda total.

3. Análise de Complexidade e NP-Completude

3.1 Complexidade pseudo-polinomial

No algoritmo proposto, utilizamos programação dinâmica para preencher a tabela “solucoes”, o que nos leva a um tempo de execução de aproximadamente O(demanda * número de ligas disponíveis), onde "demanda" representa o valor numérico da demanda (em metros) e "número de ligas disponíveis" é a quantidade de tamanhos diferentes de ligas que temos no centro de distribuição.

No pior caso, onde a demanda é muito grande, o tempo de execução será proporcional à demanda. Isso ocorre porque, para cada valor de demanda de 1 a "demanda", percorremos todos os tamanhos de ligas disponíveis para verificar a combinação que resulta no número mínimo de ligas.

No entanto, ao considerar a representação binária da demanda, o tamanho de bits necessário para representar um número "demanda" cresce de forma logarítmica em relação ao próprio valor de "demanda". Isso significa que a quantidade de iterações necessárias para preencher a tabela “solucoes” aumenta exponencialmente com o tamanho de bits da entrada, ou seja, O(n*2^log(w)).

Portanto, concluímos que o algoritmo de distribuição de ligas metálicas é pseudo-polinomial, o que implica que sua complexidade é exponencial no tamanho de bits da entrada. Isso significa que, à medida que o valor numérico da demanda aumenta, o tempo de execução do algoritmo cresce de forma exponencial devido à quantidade de iterações necessárias para preencher a tabela “solucoes”.

3.1 NP

Para demonstrar que o problema de otimização de distribuição de ligas metálicas é NP (não determinístico polinomial), devemos mostrar que é possível verificar se uma solução proposta é válida em tempo polinomial.

Nesse caso, a solução proposta seria um conjunto de ligas que atende à demanda do cliente. Para verificar se essa solução é válida, basta somar os tamanhos das ligas no conjunto e verificar se a soma é igual ou superior à demanda. Esse processo pode ser feito em tempo polinomial, uma vez que a soma dos tamanhos é uma operação de tempo constante e a verificação também é feita em tempo constante.

Portanto, a verificação de uma solução proposta para o problema de distribuição de ligas metálicas é possível em tempo polinomial, o que classifica o problema como NP.

3.1 NP-Completude

Podemos provar que o problema das ligas metálicas é NP-Completo realizando a redução em tempo polinomial desse problema para um problema já conhecido como NP-Completo, para isso, primeiro vamos provar que o problema das ligas metálicas é equivalente ao problema do troco.

O problema do troco consiste em determinar o menor número de moedas de um sistema monetário que somam um valor desejado. O problema das ligas consiste em determinar o menor número de ligas de diferentes tamanhos que somam uma demanda desejada.

Para mostrar que esses problemas são equivalentes, podemos fazer o seguinte:

Dada uma instância do problema das ligas, com um vetor ligas_disponiveis contendo os tamanhos de ligas disponíveis e uma demanda do cliente, criamos uma instância do problema do troco com o mesmo vetor ligas_disponiveis, mas interpretando os tamanhos das ligas como valores das moedas, e a demanda do cliente como o valor desejado.

O objetivo será minimizar o número de moedas (ligas) que somam o valor desejado (demanda).

Essa equivalência é feita em tempo polinomial, pois basta copiar os valores dos tamanhos das ligas para os valores das moedas, e atribuir a demanda do cliente ao valor desejado.

Se pudéssemos resolver o problema do troco em tempo polinomial, poderíamos resolver o problema das ligas em tempo polinomial também.

Isso mostra que o problema das ligas é pelo menos tão difícil quanto o problema do troco.

Agora sabemos que os problemas do troco e das ligas são equivalentes, logo, para provar a NP-Completude do problema das ligas, basta realizar a redução do problema mais conhecido do troco para o problema que sabemos de antemão que é NP-Completo, nesse caso o **problema da soma de subconjuntos**.

Problema do Subconjunto de Soma (Subset Sum):

Dado um conjunto de números inteiros positivos $\{a_1, a_2, ..., a_n\}$ e um valor-alvo S, a questão é determinar se existe um subconjunto dos números cuja soma é igual ao valor-alvo S.

Problema do Troco:

Dado um conjunto de moedas com valores inteiros positivos $\{c_1, c_2, ..., c_n\}$ e um valor V, a questão é encontrar a menor quantidade de moedas necessárias para representar o valor V em troco.

Redução do Subconjunto de Soma para o Troco:

Dada uma instância do problema do Subconjunto de Soma com um conjunto de números inteiros positivos $\{a_1, a_2, ..., a_n\}$ e um valor-alvo S, vamos criar uma instância correspondente do problema do troco. Perceba que a instância deve conter uma solução ótima para o problema do troco, ou seja, a instância desse problema do subconjunto da soma deve ter pelo menos o valor de cada moeda no problema do troco que alcance o valor alvo, ou seja, se o alvo for 4, temos que ter a instância {1,1,1,1,2,2,3,4} e a solução do problema tem que ser o menor subconjunto formado por esses elementos. Dessa forma podemos perceber que o problema do troco é uma especialização do problema do subconjunto de soma.

Definimos um conjunto de moedas disponíveis, onde cada moeda tem um valor correspondente igual aos elementos do conjunto do Subconjunto de Soma. Ou seja, o conjunto de moedas será $C = \{a_1, a_2, ..., a_n\}$.

O valor do troco que queremos alcançar será igual ao valor-alvo do Subconjunto de Soma, ou seja, o valor do troco é V = S.

Agora, podemos afirmar que uma solução para o problema do Subconjunto de Soma é equivalente a uma solução para o problema do troco.

Se existir um subconjunto dos números cuja soma é igual ao valor-alvo S no problema do Subconjunto de Soma, isso significa que podemos selecionar as moedas correspondentes a esses números no problema do troco para obter um troco com valor V.

Por outro lado, se existir uma solução para o problema do troco com valor V, isso significa que podemos selecionar as moedas correspondentes a essa solução no problema do troco para obter um subconjunto dos números cuja soma é igual a S no problema do Subconjunto de Soma.

Portanto, mostramos que qualquer instância do problema do Subconjunto de Soma pode ser reduzida para uma instância correspondente do problema do troco.

Redução do Troco para o Subconjunto de Soma:

Dada uma instância do problema do troco com um conjunto de moedas $\{c_1, c_2, ..., c_n\}$ e um valor V, vamos criar uma instância correspondente do problema do Subconjunto de Soma.

Definimos um conjunto de números inteiros positivos, onde cada número tem um valor correspondente igual aos elementos do conjunto de moedas do problema do troco. Ou seja, o conjunto de números será $\{a_1, a_2, ..., a_n\}$.

O valor-alvo que queremos alcançar no problema do Subconjunto de Soma será igual ao valor do troco do problema do troco, ou seja, o valor-alvo é S = V.

Agora, podemos afirmar que uma solução para o problema do troco é equivalente a uma solução para o problema do Subconjunto de Soma.

Se existe uma solução para o problema do troco com valor V, isso significa que podemos selecionar as moedas correspondentes a essa solução no problema do troco para obter um subconjunto dos números cuja soma é igual a S no problema do Subconjunto de Soma.

Por outro lado, se existir um subconjunto dos números cuja soma é igual ao valor-alvo S no problema do Subconjunto de Soma, isso significa que podemos selecionar as moedas correspondentes a esses números no problema do troco para obter um troco com valor V.

Portanto, mostramos que qualquer instância do problema do troco pode ser reduzida para uma instância correspondente do problema do Subconjunto de Soma.

Com isso, concluímos que o problema do troco e o problema do Subconjunto de Soma são ambos NP-completos, uma vez que cada um pode ser reduzido ao outro de forma polinomial

4. Conclusão

A implementação deste algoritmo de distribuição de ligas metálicas utilizando programação dinâmica foi uma experiência valiosa. Ao construir o algoritmo e aplicá-lo a diferentes instâncias do problema, pude aprofundar meu conhecimento em programação dinâmica e compreender melhor sua aplicação em otimização de processos.

Durante a implementação, enfrentei desafios relacionados à manipulação das estruturas de dados, como a tabela “solucoes”, e à correta formulação dos subproblemas e transições de estado. Foi necessário garantir que o algoritmo fosse eficiente e produzisse a solução ótima em tempo pseudo-polinomial.

Além disso, a implementação destacou a importância da redução de problemas para a conclusão da NP-Completude. Ao mostrar que o problema de distribuição de ligas metálicas pode ser reduzido a partir de um problema já conhecido como NP-completo, como o problema da soma de subconjuntos, pude estabelecer sua pertinência na classe dos problemas NP-completos.

Essa experiência proporcionou uma compreensão mais profunda da programação dinâmica, da redução de problemas e da complexidade de algoritmos. Aprendi a avaliar a eficiência e a corretude de algoritmos e a realizar análises detalhadas para demonstrar a NP-Completude de um problema.

Em resumo, essa implementação permitiu a aplicação prática dos conceitos de programação dinâmica e redução de problemas, proporcionando um maior entendimento sobre a complexidade de algoritmos e sua relação com a classe NP-completo. Os conhecimentos adquiridos serão valiosos para futuros projetos que envolvam a resolução de problemas de otimização e a análise de sua complexidade.

Referências Bibliográficas

ZIVIANI, Nivio. Projetos de Algoritmos com Implementações em Java e C + +. Editora Cengage, 2006.

CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. Algoritmos: Teoria e Prática. 6ª edição. Editora Elsevier, 2020.

Instruções para compilação e execução

1. Abra um terminal e navegue até o diretório onde o arquivo Makefile está localizado.

2. Execute o comando "make" na linha de comando para compilar o programa. Certifique-se de que não houve erros durante a compilação.

3. Verifique se o arquivo executável "tp02" foi criado com sucesso.

4. Para executar o programa, utilize o comando "./tp02" na linha de comando. Certifique-se de que o arquivo de entrada é passado como entrada padrão, através da linha de comando (por exemplo, $ ./tp02 < casoTeste01.txt) e a saída deve ser gerada na saída padrão (não gerar saída em arquivo).

5. Confira a solução no terminal.