Trabalho Prático 3

Gabriela Tavares Barreto

Universidade Federal de Minas Gerais
Belo Horizonte - MG - Brasil

gbarreto@ufmg.br

1 Introdução

Este trabalho teve como objetivo resolver um problema de otimização relacionado à distribuição de ligas metálicas em um centro de distribuição de uma fábrica. O problema consistia em encontrar a quantidade mínima de ligas necessárias para atender a uma demanda específica de um cliente, levando em consideração os tamanhos disponíveis das ligas no estoque.

No decorrer deste relatório, serão apresentados detalhes sobre a abordagem utilizada para resolver o problema, o algoritmo implementado, sua complexidade computacional e uma análise dos resultados obtidos.

2 Método

O programa foi desenvolvido na linguagem C++, e compilado pelo G++ da GNU Compiler Collection, usando o WSL. O computador utilizado tem como sistema operacional o Windows 11, com 12 GB de RAM.

2.1 Solução

Para resolver o problema de otimização, foi utilizada a abordagem de programação dinâmica. A programação dinâmica é uma técnica que permite decompor um problema em subproblemas menores, resolvendo cada subproblema de forma ótima e combinando as soluções para obter a solução global. Neste trabalho, foi implementado um algoritmo de programação dinâmica que determina a quantidade mínima de ligas necessárias para suprir uma determinada demanda.

2.1.1 Implementação

1. A função `min_ligas` recebe a demanda total e um vetor `ligas` que contém os tamanhos das ligas metálicas disponíveis.
2. Inicializa-se uma variável `n` com o tamanho do vetor `ligas` e cria-se um vetor `v` de tamanho `demanda + 1` para armazenar os resultados intermediários. Inicializa-se todas as posições do vetor `v` com um valor muito grande (representado por `INT_MAX`) para indicar que ainda não se encontrou uma solução.
3. Define-se `v[0] = 0`, pois não são necessárias ligas metálicas para atingir uma demanda de tamanho zero.

4. Inicia-se um loop de 1 até a demanda total (`i`) e um loop de 0 até o número de ligas disponíveis (`j`).
5. Verifica-se se o tamanho da liga `ligas[j]` é menor ou igual à demanda `i`. Se for, isso significa que é possível usar a liga `ligas[j]` para atingir a demanda `i`.
6. Atualiza-se `v[i]` com o valor mínimo entre `v[i]` e `1 + v[i - ligas[j]]`, ou seja, o mínimo entre o número de ligas necessário para atingir a demanda `i` sem utilizar a liga `ligas[j]` e o número de ligas necessário para atingir a demanda `i - ligas[j]` somado de 1 (que representa o uso da liga `ligas[j]`).
7. Repete-se os passos 5 e 6 para todas as combinações possíveis de tamanhos de ligas e demandas.
8. Ao final do loop, o valor de `v[demanda]` será o número mínimo de ligas necessárias para atingir a demanda total.
9. No `main`, o código lê o número de casos de teste `T`.
10. Em cada caso de teste, lê-se o número de tipos de ligas `N` e a demanda `W`.
11. Lê-se os tamanhos das ligas disponíveis e armazena-os no vetor `ligas`.
12. Chama-se a função `min_ligas` passando a demanda `W` e o vetor `ligas` para obter o número mínimo de ligas necessárias.
13. Imprime-se o número mínimo de ligas na saída padrão.
14. Repete-se os passos 10 a 13 para todos os casos de teste.
15. O programa termina retornando 0.

3 Análise de complexidade e NP-Completude

3.1 Complexidade

A complexidade do código pode ser analisada considerando a quantidade de ligas (N) e a demanda total (W).

1. Inicialização:
   - A criação do vetor v com tamanho $demanda + 1$ tem complexidade $O(W)$, onde W é a demanda total.
   - Preencher o vetor v com o valor INF tem complexidade $O(W)$.
2. Loop Externo: O loop externo executa no máximo W iterações, de 1 até a demanda total.
3. Loop Interno: O loop interno executa no máximo N iterações, percorrendo todas as ligas disponíveis.
4. Condição `if`: A verificação `if (ligas[j] <= i)` é executada para cada combinação de i e j. Se a condição for verdadeira, o código dentro do `if` é executado.
5. Atribuição e chamada de função: A atribuição `v[i] = min(v[i], 1 + v[i - ligas[j]]);` e a chamada de função `min_ligas(W, ligas);` têm complexidade constante.

Considerando esses pontos, podemos analisar a complexidade do código:

- A inicialização do vetor v tem complexidade $O(W)$.
- O loop externo executa no máximo W iterações.

- O loop interno executa no máximo N iterações.
- A condição `if` é executada para cada combinação de i e j.
- A atribuição e a chamada de função têm complexidade constante.

Portanto, a complexidade do algoritmo é $O(N \cdot W)$.

4 NP-Completude

O algoritmo proposto utiliza programação dinâmica para encontrar o número mínimo de ligas metálicas necessárias para atender a uma determinada demanda. A ideia principal é construir uma tabela v, em que cada posição v[i] armazena o número mínimo de ligas necessárias para atingir a demanda i. O algoritmo percorre todas as demandas de 1 até a demanda total e, para cada demanda, verifica todas as combinações possíveis de ligas disponíveis.

No entanto, ao analisar a complexidade do algoritmo, observamos que o número de combinações possíveis cresce exponencialmente com o tamanho da entrada. Suponha que a demanda total seja W e o número de tipos de ligas disponíveis seja N. Nesse caso, o número de combinações possíveis é da ordem de $O(N^W)$. Isso ocorre porque, para cada demanda i, o algoritmo avalia todas as N opções de ligas disponíveis.

A complexidade exponencial do algoritmo indica que ele pode se tornar rapidamente impraticável à medida que o tamanho da entrada aumenta. Essa característica é uma indicação de que o problema em questão pertence à classe de problemas NP-completos.

4.1 Redução do Problema

Para demonstrar a NP-completude do problema das ligas metálicas, faremos uma redução a partir do problema da mochila, que já é conhecido como um problema NP-completo.

O problema da mochila consiste em encontrar a melhor combinação de itens para colocar em uma mochila, respeitando a capacidade máxima da mochila e maximizando o valor total dos itens selecionados. Cada item possui um peso e um valor associados.

1. Dado uma instância do problema da mochila com os seguintes parâmetros:
   - Um conjunto de N itens, onde cada item i possui um peso peso[i] e um valor valor[i].
   - Uma capacidade máxima da mochila capacidade.
2. Vamos criar uma instância equivalente do problema das ligas metálicas, com os seguintes parâmetros:
   - Um conjunto de N ligas metálicas, onde cada liga i possui um tamanho ligas[i].
   - Uma demanda total de ligas ‘demanda‘ igual à capacidade máxima da mochila.
3. Agora, para cada item i do problema da mochila, criamos uma liga metálica correspondente i com tamanho igual ao peso do item. Ou seja, ligas[i] = peso[i].
4. Executamos o algoritmo proposto para resolver o problema das ligas metálicas com os parâmetros modificados.
5. Se o número mínimo de ligas retornado pelo algoritmo for igual a demanda, então a resposta para o problema da mochila é "SIM". Caso contrário, a resposta é "NÃO".

Se existe uma solução viável para o problema da mochila (ou seja, uma combinação de itens com valor máximo dentro da capacidade), isso significa que é possível atingir a demanda total de ligas usando ligas metálicas com tamanho igual aos pesos dos itens selecionados. Portanto, o número mínimo de ligas necessário será exatamente igual à demanda total.

Se não existe uma solução viável para o problema da mochila, não será possível atingir a demanda total de ligas usando ligas metálicas com tamanho igual aos pesos dos itens, pois excederia a capacidade da mochila. Portanto, o número mínimo de ligas necessário será maior que a demanda total.

Assim, a redução mostra que o problema das ligas metálicas é NP-completo, pois foi reduzido a partir de um problema NP-completo conhecido, o problema da mochila.

5 Conclusão

Concluindo, este trabalho me proporcionou a oportunidade de enfrentar um desafio interessante no contexto da otimização do processo de distribuição de ligas metálicas. Através do uso de programação dinâmica, fui capaz de desenvolver um algoritmo eficiente para determinar o número mínimo de ligas necessárias para atender a uma determinada demanda.

Ao aplicar o algoritmo em diferentes casos de teste, pude observar sua eficácia na solução do problema proposto. Nesse sentido, este trabalho me permitiu explorar a complexidade do algoritmo desenvolvido, levando em consideração o tamanho da entrada em termos de bits.

Por fim, essa experiência de resolução de um problema realístico e a análise da complexidade do algoritmo enriqueceram minha compreensão sobre a importância da eficiência computacional e dos desafios inerentes à solução de problemas NP-completos.

6 Instruções de compilação e execução

Para compilar o programa, deve-se seguir os seguintes passos:

- Acessar o diretório TP03-Template-CPP;
- Utilizando um terminal, digitar make, de forma a gerar o arquivo executável **tp3**;
- Para a execução, chamar o TP3 seguido de $<$ e passando um arquivo de entrada do tipo .txt com os dados do grafo como parâmetro.