UFMG
UNIVERSIDADE FEDERAL DE MINAS GERAIS

Universidade Federal de Minas Gerais

Turma: Ciência da Computação **Prof.:** Vinícius

Nome: Rubia Alice Moreira de Souza
Matrícula: 2022043507

Centro de Distribuição

Belo Horizonte
2023

1. Introdução

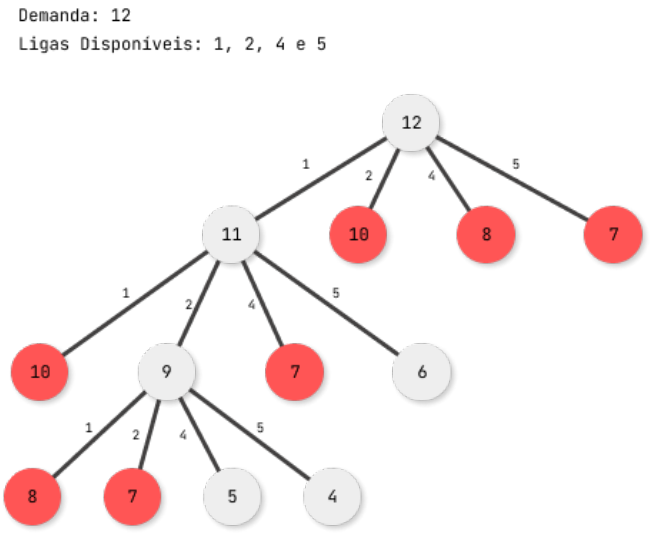
O trabalho consiste no desenvolvimento de um algoritmo para conseguir determinar a menor quantidade de ligas metálicas dentre diferentes tamanhos que somam uma certa quantidade solicitada por um cliente. Os tamanhos de ligas metálicas que podem ser entregues devem ser selecionadas de um grupo específico disponível no estoque.

Esse problema se assemelha muito ao problema da mochila. Contudo, neste caso, temos que a quantidade de itens que podem ser selecionados é ilimitada. Ou seja, para atender uma demanda, podemos selecionar qualquer quantidade dos diferentes tamanhos de ligas metálicas a fim de atender o tamanho solicitado.

Todo o histórico de desenvolvimento, pode ser acompanhado pelos commits neste repositório: UFMG-ALG-Centro-De-Distribuicao. Ele apenas se tornará público após a data final de entrega deste trabalho.

2. Modelagem

Assim como no problema da mochila, encontramos diversas repetições ao dividir o problema original em subproblemas. Por exemplo: neste caso com uma demanda de 12 unidades de ligas metálicas para ser atendida com ligas de tamanhos 1, 2, 4 e 5, teríamos as seguintes possíveis divisões:



É possível notar que assim que resolvemos o subproblema 11, todos subproblemas restantes do problema original 12 já foram computados, que são os vértices marcados em vermelho. Uma vez que os subproblemas se sobrepõem, podemos utilizar Programação Dinâmica para evitar repetições de soluções.

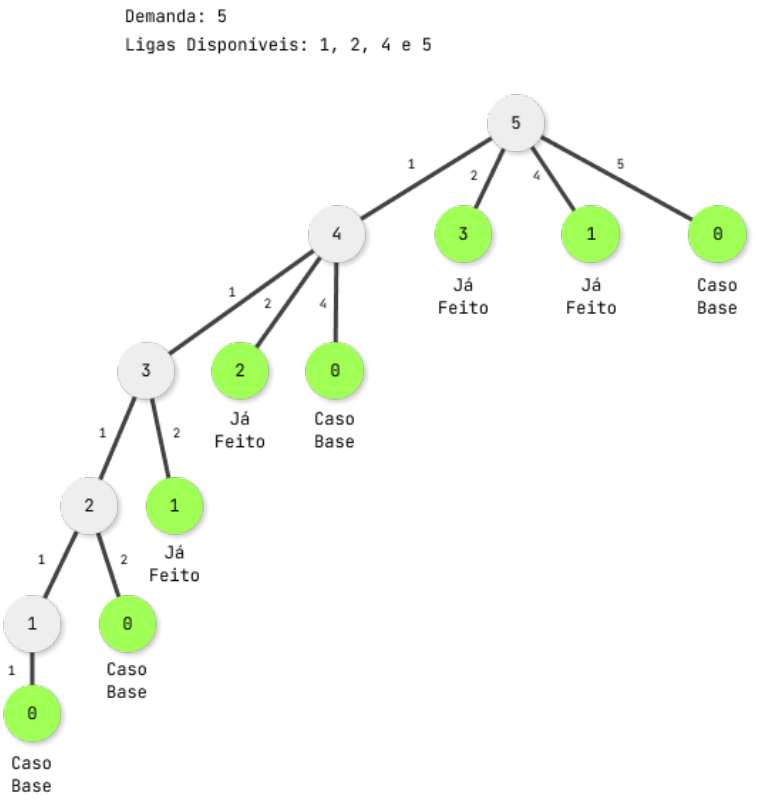
Podemos usar um vetor para armazenar as soluções dos problemas que já foram computados anteriormente. Além disso, podemos identificar um subproblema apenas observando o valor da demanda informada, uma vez que as ligas disponíveis para resolver

cada subproblema sempre serão as mesmas. Assim, o index de uma posição no vetor de soluções será equivalente à demanda do problema definido por ela.

Podemos resolver problemas de duas formas: Top Down e Bottom Up. A única diferença entre as duas soluções consiste na ordem que os subproblemas serão solucionados.

2.1. Solução Top Down

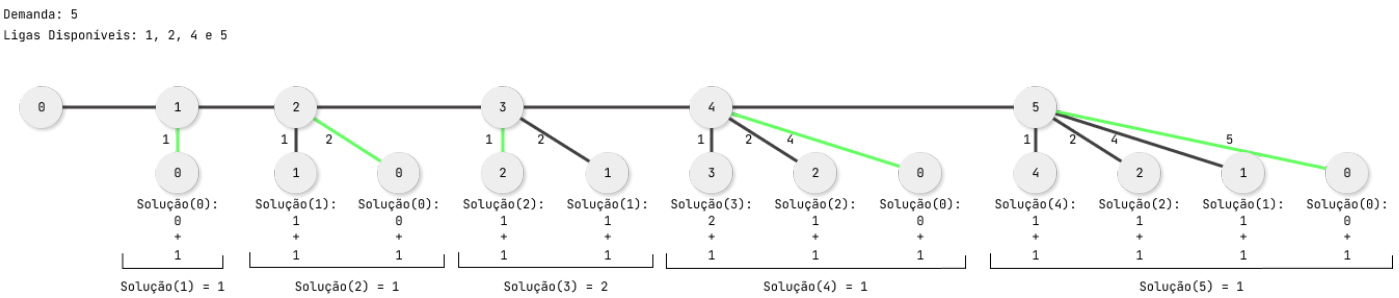
Para a solução Top Down, nós iremos partir da demanda total e subtrair dela o valor de cada liga disponível. Para cada valor resultante não negativo, iremos resolvê-los como um subproblema. Então, basta repetir esse processo até o caso base em que a demanda será 0. Ao armazenar os resultados, a quantidade de chamadas será reduzida de modo considerável:



No entanto, para esta solução, é possível que a profundidade de chamadas seja muito grande e ultrapasse o limite imposto pelo sistema operacional devido o caminho mais à esquerda das chamadas, em que a demanda é sempre reduzida em 1. Isso levará a um erro de StackOverflow. Por esse motivo, mantive como solução principal a Bottom Up, que funciona de forma iterativa.

2.2. Solução Bottom Up

Já a solução Bottom Up será construída gradualmente a partir de zero até o valor da demanda:



Para cada valor de zero até a demanda final, iremos testar todos os tamanhos de ligas metálicas disponíveis para atendê-lo. Cada teste será a liga metálica atual mais a menor quantidade necessária para atender a demanda restante.

Por exemplo: Se estamos no valor 3, teremos dois casos. O primeiro será a liga metálica de tamanho 1, que terá uma quantidade mínima de ligas igual a 1 mais a solução do subproblema em que a demanda era 3 - 1 = 2. Já o segundo será a liga metálica de tamanho 2, que terá uma quantidade mínima de ligas igual a 1 mais a solução do subproblema em que a demanda era 3 - 2 = 1.

Ou seja, além da liga metálica que estamos considerando atualmente, iremos precisar da menor quantidade de ligas para atender a demanda restante, que já foi armazenada no vetor de soluções da programação dinâmica.

Além disso, é certo que já teremos calculado o valor ideal dos subproblemas resultantes da subtração, pois estamos construindo todas as soluções de baixo para cima até a demanda original.

Por fim, dentre todos os resultados encontrados em cada caso de teste, basta selecionar o menor e armazená-lo no vetor de soluções para que possa ser utilizado em soluções futuras.

3. Complexidade

A complexidade de tempo para ambas as implementações é a mesma. Contudo, ela é mais simples de ser analisada na implementação Bottom Up. Ela consiste em 2 loops aninhados. O externo itera de 1 até a demanda do problema original. Já o interno percorre cada um dos tamanhos de ligas metálicas disponíveis. Assim, se temos uma demanda “d” e uma quantidade “n” de ligas metálicas, a complexidade será: O(n * d).

4. NP-Completude

4.1. Prova que está em NP

No entanto, a complexidade é diferente ao avaliarmos o tamanho da entrada em sua representação em bits. Se temos uma demanda com valor “d”, a sua representação será: rd = $\log_2(d)$. Escrevendo “rd” em função de “d”, temos: $2^{rd} = 2^{\log(d)} \rightarrow d = 2^{rd}$. Então, colocando a

complexidade em função da representação em bits, temos: $O(n * 2^{rd})$, sendo “n” a quantidade de ligas metálicas disponíveis e “rd” a representação em bits de “d”.

Ou seja, o tempo de execução do algoritmo, O(n * d), é polinomial para o valor numérico de “d”. No entanto, para a sua representação em bits, em que $d = 2^{rd}$, ele é exponencial, $O(n * 2^{rd})$. Além disso, é possível notar que O(n * d) jamais será um limite superior para $O(n * 2^{rd})$, independentemente de qual constante você multiplicá-lo. Portanto, isso indica que este problema não está em P.

Por outro lado, sabemos que existe um algoritmo para validar uma possível solução para o problema. Se temos um conjunto de ligas metálicas e os seus respectivos tamanhos, podemos iterar sobre ele e ver se a soma delas é igual a demanda. Além disso, se desejamos que a quantidade de ligas seja menor ou igual a certo valor, basta compará-lo a quantidade de elementos no conjunto.

Portanto, se temos “n” ligas metálicas e o maior tamanho entre elas é “t”, a representação em bits de “t” será: $rt = \log_2(t)$. O custo para iterar sobre o conjunto de ligas e obter a soma delas será: O(n * rt). Como as comparações para verificar se a soma é igual a demanda e a quantidade de elementos no conjunto ser menor ou igual a um certo valor são constantes, o algoritmo para validar uma possível solução, tem custo polinomial a representação da entrada. Por esse motivo, temos que este problema está em NP.

4.2. Prova que está em NP-Difícil

Além disso, podemos provar que este problema está em NP-Difícil por meio da seguinte redução: 3-SAT $\leq_P$ Ligas Metálica. Nós podemos definir os dois problemas como problemas de decisão da seguinte forma:

3-SAT	**Ligas Metálicas**
Entrada: Uma fórmula booleana na CNF **Pergunta:** A fórmula é satisfatível?	**Entrada:** ● Um vetor “v” de inteiros com o tamanho das ligas metálicas ● Um inteiro “d”, que é a demanda do cliente ● Um inteiro “k”, que é a quantidade máxima de barras utilizadas **Pergunta:** É possível selecionar um Multiconjunto dentre os elementos de “v” de modo que somem ao valor “d” e possui no máximo “k” elementos?

Ou seja, desejamos encontrar um modo de transformar a fórmula booleana do problema 3-SAT em um vetor de inteiros, que representam os tamanhos das ligas, uma demanda “d” e um máximo de ligas “k”.

Para isso, cada variável e o seu complemento serão mapeadas a um número inteiro decimal que compõem o vetor “v”. Já a demanda do cliente será um valor que restringe

quais literais podem ser selecionados a fim de atender a demanda e serem consistentes na construção da fórmula booleana. Por fim, o valor “k” será a quantidade de números que foram selecionados para compor a soma até o valor “d”.

Primeiramente, iremos representar separadamente uma variável negada e não negada para construir a representação dos números associados a cada uma delas. As duas terão os dígitos mais significativos formados pela mesma potência de 10. Por exemplo, se temos as variáveis X, Y e Z teríamos algo como:

	10^2	10^1	10^0
X	1	0	0
¬X	1	0	0
Y	0	1	0
¬Y	0	1	0
Z	0	0	1
¬Z	0	0	1
Demanda (d)	1	1	1

Com isso, ao definir a demanda com os três primeiros dígitos sendo 1, conseguimos impedir que uma variável não negada e a sua negação sejam selecionadas juntas. Caso isso ocorresse, o valor da soma seria maior que um e não atenderia a demanda. Por outro lado, se nenhuma das duas forem selecionadas, ao somar as linhas o resultado será zero e também não atenderá a demanda. Isso garante que sempre iremos selecionar um literal para compor a resposta e satisfazer as cláusulas da fórmula booleana.

Além disso, cada literal terá mais “n” dígitos, sendo “n” a quantidade de cláusulas presentes na fórmula. Cada dígito D_i, com “i” variando de 1 até “n”, será 0, caso aquele literal não satisfaça a respectiva cláusula, ou 1, caso contrário.

Por exemplo: Se além das variáveis X, Y e Z, temos as seguintes cláusulas:

$$\underset{C_1}{(X \vee Y \vee Z)} \wedge \underset{C_2}{(\neg X \vee Y \vee \neg Z)} \wedge \underset{C_3}{(X \vee \neg Y \vee \neg Z)} \wedge \underset{C_4}{(\neg X \vee \neg Y \vee Z)}$$

Elas serão mapeadas para os dígitos D_1, D_2, D_3, D_4 da seguinte forma:

	10^6	10^5	10^4	D_1	D_2	D_3	D_4
X	1	0	0	1	0	1	0
¬X	1	0	0	0	1	0	1

Y	0	1	0	1	1	0	0
¬Y	0	1	0	0	0	1	1
Z	0	0	1	1	0	0	1
¬Z	0	0	1	0	1	1	0
Demanda (d)	1	1	1	1	1	3	1

Por fim, para atender uma demanda como 1114444, seria impossível apenas com os literais presentes na tabela atualmente. Isso pois mesmo atribuindo os valores que satisfazem a fórmula, como X = V, Y = F e Z = F, eles somarão somente até 1111131. Portanto, a fim de complementar a soma, para cada cláusula podemos adicionar 2 linhas, uma com o valor 1 e outra com o máximo que desejamos da demanda naquele dígito menos 1.

	10^6	10^5	10^4	D_1	D_2	D_3	D_4
X	1	0	0	1	0	1	0
¬X	1	0	0	0	1	0	1
Y	0	1	0	1	1	0	0
¬Y	0	1	0	0	0	1	1
Z	0	0	1	1	0	0	1
¬Z	0	0	1	0	1	1	0
C_1	0	0	0	1	0	0	0
C_1	0	0	0	3	0	0	0
C_2	0	0	0	0	1	0	0
C_2	0	0	0	0	3	0	0
C_3	0	0	0	0	0	1	0
C_3	0	0	0	0	0	3	0
C_4	0	0	0	0	0	0	1
C_4	0	0	0	0	0	0	3
Demanda (d)	1	1	1	4	4	4	4

Assim, além dos números dos literais, podemos pegar os números representados pelas cláusulas para somar a demanda desejada. Dessa forma, podemos interpretar que cada linha é um número no vetor “v”, as linhas em verde somam o valor “k” e a demanda “d” e o subvetor selecionado são os números das linhas em verde.

Essa transformação certamente tem um custo polinomial, pois ela consiste apenas na construção da tabela com base na fórmula do 3-SAT. Além disso, apenas haverá solução para a fórmula caso seja possível somar os números representados pelas cláusulas e pelos literais por construção. Uma vez que, caso um literal que não satisfaça a fórmula seja selecionado ou um que satisfaça não seja selecionado, a soma jamais será igual a demanda.