Trabalho Prático 3
Centro de distribuição

Aline Cristina Pinto
2020031412

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

alinecristinapinto@ufmg.br

1. Introdução

Este trabalho possui como objetivo desenvolver um algoritmo a partir de programação dinâmica capaz de otimizar o processo de distribuição de ligas metálicas de um Centro de Distribuição. Essa otimização consiste em reduzir o número de ligas necessárias para uma demanda considerando os tamanhos das ligas disponíveis. Sendo assim, o algoritmo criado busca encontrar o menor número de ligas necessárias para uma dada demanda. Nessa documentação será abordado a modelagem do algoritmo implementado, sua complexidade e redução ao problema de Subset Sum.

2. Modelagem

2.1. Definição do problema

O problema da Distribuição de Ligas pode ser reescrito como: "Dada uma demanda de ligas metálicas D, e um número infinito de ligas de tamanho T = $\{T_1, T_2, ..., T_n\}$, sendo que sempre haverá ligas de tamanho 1, qual é o número mínimo de ligas necessárias para se obter a demanda D?"

Para modelar a Distribuição de Ligas, o primeiro passo foi escolher as estruturas de dados. Como para cada demanda recebemos a demanda, o número de tipos de ligas (n) e seus tamanhos, foi optado por representar esse problema por um inteiro de demanda (D) e um vetor de inteiros com o tamanho das ligas (ligas[n] = $\{T_1, T_2, ..., T_n\}$).

Esse problema poderia ser resolvido por recursão da seguinte forma:
Caso base: D == 0, então número de ligas = 0

D > 0: então:

$minimoLigas(ligas[0..n-1],\ D) = min\{1 + minimoLigas(D - ligas[i])\}$

$sendo\ i = n - 1\ e\ ligas[i] \leq D$

Entretanto, a complexidade desse algoritmo seria exponencial, visto que se desenharmos a árvore completa de recursão, seria possível observar que muitos subproblemas são resolvidos várias vezes. Então, a solução com programação dinâmica foi usada.

2.2. Programação dinâmica

Para resolver o problema proposto, foi escolhida a abordagem de *Bottom-up dynamic programming*, apenas por considerar a mesma mais intuitiva. Nessa abordagem criamos a tabela como um vetor de inteiros, sendo sua função armazenar a contabilização mínima de ligas necessárias para cada valor (do seu índice) entre 0 e a demanda passada.

- Para o valor 0 (tabela[0]) atribuímos 0, visto que não é necessário nenhuma liga.
- Para os demais valores, o seguinte cálculo é realizado:
    - Para cada valor *i*, são utilizados alguns dos valores armazenados em posições anteriores da tabela (que já são mínimos para o valor de seu índice) + 1 (nova liga necessária). O menor valor dessas somas é adicionado a tabela[i].

A seguir, o pseudocódigo da mesma: ($n = quantidade\ de\ ligas$ e $D = demanda$)

```
Integer tabela [D + 1] <- {}                                    O(1)
tabela [0] = 0                                                  O(1)

for i=1 ... i<=D                                                O(D)
  tabela[i] <- infinite                                         O(1)

  for j=0 ... j<n                                               O(n)

    if ligas[j] <= i                                            O(1)
      Integer subSolucao= tabela[i - ligas[j]] + 1;      O(1)

        if subSolucao < tabela[i]                               O(1)
          tabela[i] = subSolucao;                               O(1)

return tabela[D]                                                O(1)
```

Observa-se que as operações que consomem maior complexidade são os dois *for* que estão em função de n e D. Logo, a complexidade é dada por $O(n * D)$.

3. Análise de Complexidade e NP-Completude:

3.1. Análise de Complexidade

Como discutido na seção anterior, a complexidade encontrada é dada por $O(n * D)$, no qual $n = quantidade\ de\ ligas$ e $D = demanda$. Essa complexidade está em função da entrada (decimal), porém, em problemas aritméticos, os inputs inteiros são codificados em binário. Logo, precisamos fazer uma "conversão" para essa representação:

D em binário $\rightarrow log_2 D$

Supondo que s seja o número de bits necessário para representar D ($s = log_2 D$), então:

$2^s = 2^{log_2 D}$

$2^s = D$

Logo, a complexidade pode ser reescrita como $O(n * 2^s)$, que não é polinomial.

Como já abordado na descrição do trabalho, o problema da Distribuição de Ligas é de fato pseudo-polinomial, ou seja, o seu tempo de execução é polinomial no valor numérico da entrada, mas é exponencial no comprimento da entrada.

3.2. NP-Completude

Algoritmos pseudo-polinomiais: Knapsack, Subset Sum, etc.

Para provar que o problema da Distribuição de Ligas (DL) pertence a NP-Hard vamos partir do pressuposto que o problema de Subset Sum pertence a NP-Completo.

- Problema de Subset Sum (SS): Dado n inteiros não negativos $w_1, w_2, \ldots, w_n$ e uma soma alvo K , existe um subconjunto que contenha uma soma igual a K?

$DL \notin NP$

Não é possível criar um algoritmo verificador para DL que seja polinomial, pois não é possível verificar se o valor encontrado é de fato o mínimo. Outra forma de visualizar isso se deve ao fato desse problema não ser um problema de decisão. Logo, **o problema da Distribuição de Ligas (DL) não pertence a NP (possivelmente é NP-Hard)**.

$SS \leq_p DL$

Dada uma instância genérica do problema de Subset Sum, definida por um vetor v^{SS} de n inteiros e uma soma k, queremos construir uma instância particular para a Distribuição de Ligas definida por uma demanda d, o número de tipos de ligas e o valor das ligas.

-

4. Referências:

https://stackoverflow.com/questions/19647658/what-is-pseudopolynomial-time-how-does-it-differ-from-polynomial-time