UNIVERSIDADE FEDERAL DO SUL E SUDESTE DO PARÁ
FACULDADE DE COMPUTAÇÃO E ENGENHARIA ELÉTRICA
ENGENHARIA DA COMPUTAÇÃO

TRABALHO FINAL DE COMPLEXIDADE DE ALGORITMOS
IMPLEMENTAÇÃO DE ALGORITMOS GULOSOS E PROGRAMAÇÃO DINÂMICA UTILIZANDO C++ E PYTHON

José Arthur Pereira Alves
Luana Batista Araújo

Marabá
2022

UNIVERSIDADE FEDERAL DO SUL E SUDESTE DO PARÁ
INSTITUTO DE GEOCIÊNCIAS E ENGENHARIAS
FACULDADE DE COMPUTAÇÃO E ENGENHARIA ELÉTRICA

JOSÉ ARTHUR PEREIRA ALVES
LUANA BATISTA ARAÚJO

IMPLEMENTAÇÃO DE ALGORITMOS GULOSOS E PROGRAMAÇÃO DINÂMICA UTILIZANDO C++ E PYTHON

Relatório referente ao TRABALHO FINAL "IMPLEMENTAÇÃO DE ALGORITMOS GULOSOS E PROGRAMAÇÃO DINÂMICA UTILIZANDO C++ E PYTHON" como critério de avaliação da disciplina COMPLEXIDADE DE ALGORITMOS do Prof. Dr. Manoel Ribeiro.

Marabá
2022

SUMÁRIO

1. INTRODUÇÃO

Neste trabalho iremos apresentar duas soluções computacionais, a primeira utilizando a estratégia de algoritmos gulosos e a segunda utilizando a programação dinâmica.

Os algoritmos gulosos, também conhecidos por serem “míopes”, tomam decisões com base nas informações disponíveis a cada iteração corrente, eles não olham para as consequências que essas decisões terão no futuro e nunca se arrependem ou voltam atrás, todas as escolhas feitas são definitivas. Sendo assim, em cada iteração o algoritmo escolhe o objeto mais apetitoso que vê pela frente, a é feita de forma “gananciosa”, prestando atenção ao ganho de curto prazo ou local, sem levar em conta se isso levará a uma boa solução de longo prazo ou global. A definição de apetitoso é estabelecida a priori, antes da execução do algoritmo e o objeto escolhido passa a fazer parte da solução que o algoritmo constrói.

Embora algoritmos gulosos pareçam naturalmente corretos, a prova de sua correção é, em geral, difícil. Para compensar, algoritmos gulosos são muito eficientes. Mas os problemas que admitem solução gulosa são um tanto raros.

Já com relação a programação dinâmica, observa-se uma estratégia de recursão com o apoio de uma tabela. Como em um algoritmo recursivo, cada instância do problema é resolvida a partir da solução de subinstâncias da instância original. A principal característica da programação dinâmica é a utilização de uma tabela que armazena as soluções das diversas substâncias. Sendo, o consumo de tempo do algoritmo, proporcional ao tamanho da tabela.

O truque da utilização da programação dinâmica está em resolver os subproblemas na ordem correta para que sempre que a solução de um subproblema seja necessária, ela já esteja disponível em uma tabela, ou seja, em vez de resolver os subproblemas recursivamente, resolva-os sequencialmente armazenado as soluções de cada um.

A Tabela 1 a seguir detalha as principais diferenças entre os algoritmos gulosos e os algoritmos de programação dinâmica.

Tabela 1 - Comparação entre algoritmos gulosos e algoritmos de programação dinâmica

Algoritmo Guloso	Programação Dinâmica
Escolhe a alternativa mais promissora no momento	Explora todas as alternativas
Execução mais rápida	Execução mais lenta
Nunca se arrepende de uma decisão	Se arrepende de decisões tomadas
Não tem prova de correção simples	Tem prova de correção simples.

2. METODOLOGIA

Neste trabalho, para cada um dos algoritmos citados anteriormente será apresentado a implementação de uma solução computacional que resolve determinado problema.

Para os algoritmos gulosos, utilizou-se a linguagem de programação C++ por meio da IDE (Integrated Development Environment) Dev-C++. Nela, será apresentada uma solução gulosa para o algoritmo de quantidade mínima e máxima de dinheiro necessária para obter N tipos de doce de uma Loja de Doces, sendo que, nesta loja há uma promoção onde para cada doce comprado você ganha K outros doces (de tipos diferentes).

Dando continuidade, com relação aos algoritmos de programação dinâmica, a linguagem de programação utilizada foi o Python, sendo o trabalho escrito e compilado por meio do editor de código-fonte VSCode. Nele, será apresentado uma solução de programação dinâmica do problema da mochila booleana, onde é necessário obter o maior valor total possível com base em objetos, seus pesos e valores e na capacidade da mochila.

2.1. Algoritmo Guloso

Primeiramente, ao analisar a solução é preciso compreender o problema que se deseja resolver com o algoritmo. A imagem 1 a seguir mostra um comentário explicativo acerca da apresentação do problema, bem como, específica logo de cara as restrições do algoritmo e a complexidade de tempo, que ficarão claras no decorrer da implementação.

Imagem 1 - Comentário inicial do algoritmo

AlgoritmoGuloso.cpp

```
/* UNIVERSIDADE FEDERAL DO SUL E SUDESTE DO PARÁ
   DISCIPLINA: COMPLEXIDADE DE ALGORITMOS
   PROFESSOR: MANOEL RIBEIRO
   ALUNOS: JOSÉ ARTHUR E LUANA BATISTA

   DEFINIÇÃO DO PROBLEMA:
   Em uma loja de doces existem N tipos de doce e o cliente tem acesso

   Em uma loja de doces com N tipos de doce o cliente observa o preço
   de cada um.
   Nela ocorre uma promoçaão, na compra de de 1 doce você cada K
   outros doces (de tipos diferentes) de graça.

   Agora você tem que responder duas perguntas.
   1 - Qual a quantidade mínima de dinheiro que você tem que gastar
       para obter todos os N tipos de doce?
   2 - Qual a quantidade máxima de dinheiro que você gastaria para
       obter todos os N tipos de doce?

   Para os dois casos se aplica a condição da promoção de que na compra
   de 1 você ganha K doces.

   SOLUÇÃO:
   O algoritmo a seguir é uma solução gulosa para o problema.

   RESTRIÇÕES:
   1 <= T(num de testes) <= 50
   1 <= N(num tipos de doce) <= 1000
   0 <= K(doces ganho na promo) <= N-1
   1 <= Ai <= 100

   COMPLEXIDADE DE TEMPO: O(nlogn)
*/
```

Prosseguindo, temos na Imagem 2 o início da implementação da solução. Nas linhas 46 e 55 pode-se observar a declaração das variáveis utilizadas pelo algoritmo, sendo elas: a quantidade de testes ou execuções que o usuário deseja realizar (`t`), a quantidade de tipos de doce que existem nesta loja (`n`), a quantidade de doces ganhos pela promoção (`k`), o valor máximo (`mx`) que será retornado no final, o valor mínimo (`mn`) que será retornado no final, além de três variáveis auxiliares `i`, `j` e `l`. Já nas linhas 50, 58 e 60 é possível observar a definição de algumas delas pelo usuário, sendo definido respectivamente, as variáveis, `t`, `n` e `k`.

Após determinar as três variáveis principais citadas anteriormente podemos passar para a lógica de execução do algoritmo. Inicialmente, na linha 62 temos a criação de um array ‘`a`’ de tamanho ‘`n`’, neste array será armazenado os valores de cada tipo de doce e na linha 68 há a ordenação dos elementos dos valores deste array de forma crescente.

Imagem 2 - Definição de variáveis e criação de array ordenado

```
36  #include <bits/stdc++.h>
37  #include <locale.h>
38
39  using namespace std;
40
41  int main() {
42
43      setlocale(LC_ALL, "Portuguese");
44      cout << "******************** ALGORITMO GULOSO ********************\n\n";
45
46      int t; // declaração da variável t
47
48      // cin - stream de entrada - realiza a leitura de uma sequência de dados
49      cout << "---------------------- CONFIGURAÇÃO ----------------------" << "\nInsira o
50      cin >> t; // pega a quantidade de testes que se deseja realizar e atribui a t
51
52      // enquanto ainda houver testes
53      while(t--)
54      {
55          int n,k,i,l,j,mn=0,mx=0; //declaração de variáveis
56
57          cout << "\n-------------------- DADOS DO PROBLEMA --------------------" << "\nIn
58          cin >> n; // pega o número de tipos de doce e atribui a variável n
59          cout << "Insira o número máximo de doces que podem ser ganhos: ";
60          cin >> k; // pega o número máximo de doces que pode ser ganho e atribui a k
61
62          int a[n]; // cria um array 'a'de tamanho 'n'
63
64          for(i=0;i<n;i++){ // para cada item do array, ou seja, para cada doce
65              cout << "Insira o preço do doce " << i+1 << ": ";
66              cin>>a[i]; // insere o preço de cada um
67          }
68          sort(a,a+n); // ordena os elementos do array 'a'
```

Com a posse do número de tipos de doce, a quantidade de doces ganhos na promoção e os preços de cada tipo de doce, o programa define o valor da primeira variável auxiliar (`l`), ela receberá o valor da operação `n/(k+1)` arredondada para cima.

Em seguida, ocorre a execução dos dois laços de repetição principais do algoritmo. No primeiro laço, feito utilizando a estrutura de repetição `for` e a variável auxiliar `i`, o array '`a`' é percorrido `i` até `l`, á medida que o array é percorrido a variável `mn` tem o seu valor atual somado ao valor de `a[i]`, em outras palavras, como o array está ordenado, basta verificar por meio de uma equação quantos doces são necessários comprar para obter todos os tipos de doce, esta operação foi feita na atribuição da variável `l` e a partir daí basta pegar os menores valores para esta quantidade de itens.

Já no segundo laço, utilizou-se a estrutura de repetição `while` e a variável auxiliar `j`, nós temos novamente a utilização da variável `l`, mas como novidade temos a variável `j` de valor `n-1`. Neste laço de repetição, conforme `l` decresce a variável `mx` tem o seu valor atual somado a `a[j]`, o que se prestarmos atenção, é basicamente uma operação contrária à do laço anterior, só que desta vez adicionando a variável os maiores valores de doces da quantidade de doces que se pode comprar.

Imagem 3 - Implementação do algoritmo guloso

```
70          l=ceil((double)n/(double)(k+1)); // l = n/(k+1) com arredondamento para cima
71
72          for(i=0;i<l;i++) // de i = 0 até l
73          {
74              mn+=a[i]; // o valor mn será igual ao valor atual + o valor a[i]
75          }
76
77          j=n-1; // j é igual ao número de tipos de doce menos 1
78          while(l--) // enquanto l-- for verdadeiro
79          {
80              mx+=a[j]; // o valor mx será igual ao valor atual + o valor a[j]
81              j--; // decrementação de j
82          }
83          cout << "\n----------------------- RESULTADO -----------------------" << "\nQ
84          cout << mn; // imprime o valor minimo
85          cout << "\nQuantidade máxima de dinheiro necessária: ";
86          cout << mx << endl; // imprime o valor máximo
87
88      }
89      return 0;
90  }
```

2.2. Programação Dinâmica - Mochila booleana

Supondo um conjunto de objetos, cada um com um certo peso e um certo valor. Quais dos objetos devo colocar na minha mochila para que o valor total seja o maior possível, supondo que a mochila tenha uma capacidade de peso limitada? Ou seja, dados os pesos e valores de `n` itens, é necessário colocar esses itens em uma mochila com capacidade `W` para obter o valor total máximo na mochila. Em outras palavras, dados dois arrays inteiros `val [0..n-1]` e `p [0..n-1]` que representam valores e pesos associados a "`n`" itens, respectivamente. Também dado um inteiro `W`, que representa a capacidade da mochila, descubra o subconjunto de valor máximo de `val []` de forma que a soma dos pesos deste subconjunto seja menor ou igual a `W`. Uma observação importante a ser feita e que impacta diretamente no problema é que não é possível quebrar um item, deve-se escolher o item completo ou não escolher.

O problema da mochila booleana tem estrutura recursiva: qualquer solução de uma instância é composta por soluções de suas sub-instâncias. Através dessa estrutura recursiva pode-se aplicar a programação dinâmica.

Como outros problemas típicos de Programação Dinâmica (DP), o recálculo dos mesmos subproblemas pode ser evitado construindo um array temporário `K[][]` de maneira ascendente.

Nesse método trabalha-se considerando os mesmos casos mencionados na abordagem recursiva. Em uma tabela DP `[][]`, considera-se todos os pesos possíveis de '`1`' a '`W`' como as colunas e os pesos que podem ser mantidos como as linhas. O estado DP `[i][j]` denota o valor máximo de '`peso-j`' considerando todos os valores de '`1 a i`'. Portanto, se considerar '`wi`' (peso na linha '`i`'), podemos preenchê-lo em todas as colunas que têm '`valores de peso > wi`'. Com isso, duas possibilidades podem ocorrer:

- Preencher '`wi`' na coluna fornecida.
- Não preencher '`wi`' na coluna fornecida.

Agora surge a necessidade de tomar o máximo dessas duas possibilidades, formalmente se não preenchermos '`i-ésimo`' peso na '`enésima`' coluna, então o

estado DP `[i][j]` será o mesmo que DP `[i−1][j]`, mas se preenchermos o peso, DP `[i][j]` será igual ao valor de '`wi`' + valor da coluna com peso '`j−wi`' na linha anterior. Portanto, usa-se o máximo dessas duas possibilidades para preencher o estado atual. A seguir, na Imagem 4 é possível observar a implementação em código:

Imagem 4 - Implementação do algoritmo de programação dinâmica

```
#UNIVERSIDADE FEDERAL DO SUL E SUDESTE DO PARÁ
#DISCIPLINA: COMPLEXIDADE DE ALGORITMOS
#PROFESSOR: MANOEL RIBEIRO
#ALUNOS: JOSÉ ARTHUR E LUANA BATISTA
# Programação Dinâmica baseada em Python - Programa para o problema da Mochila booleana
# Retorna o valor máximo possível dentro de uma mochila com capacidade W

import sys

def mochila(cap, p, val, n):
    K = [[0 for x in range(cap + 1)] for x in range(n + 1)]

    # Constroi uma tabela K[][] de baixo pra cima
    for i in range(n + 1):
        for w in range(cap + 1):
            if i == 0 or cap == 0:
                K[i][w] = 0
            elif p[i-1] <= w:
                K[i][w] = max(val[i-1] + K[i-1][w-p[i-1]],  K[i-1][w])
            else:
                K[i][w] = K[i-1][w]

    return K[n][cap]
```

Através da função mochila será construída uma tabela de `K[][]` de baixo para cima, de acordo com a explicação fornecida anteriormente. O módulo "`sys`" que pode ser observado na figura é importado para ser utilizado na construção da interface. A implementação da interface e a criação dos parâmetros que são utilizados no problema pode ser observada logo abaixo:

Imagem 5 - Construção da interface

```
# Seção que pede os valores de cada parametro ao usuario, e logo após exibe o resultado.
print("\n*******************************PROBLEMA DA MOCHILA BOOLEANA*******************************\n")

sys.stdout.write("Digite 3 valores: \n")
num1 = input("")
num2 = input("")
num3 = input("")

print("\n-----------------------------------------------------------------------------------------------\t")

sys.stdout.write("Digite 3 pesos para cada valor: \n")
num4 = input("")
num5 = input("")
num6 = input("")

print("\n-----------------------------------------------------------------------------------------------")

print("Digite a capacidade total da mochila: \n")
num7 = input("")
```

Imagem 6 - Variáveis que recebem os parâmetros

```
val = [int(num1), int(num2), int(num3)]
p = [int(num4), int(num5), int(num6)]
cap = int(num7)
n = len(val)

print("\n*******************************RESULTADO*******************************\n")
print("O valor obtido é: ", mochila(cap, p, val, n))
```

De acordo com a Imagem 6 nota-se que a lista criada tem 3 elementos, com isso, será solicitado ao usuário 3 itens, e logo após, seus respectivos valores e pesos. No final a função “mochila” é chamada através de um print para realizar o cálculo e consequentemente exibir o resultado. Foi utilizado as seguintes variáveis para definir cada parâmetro necessário ao problema:

- `val` - valor de cada item
- `p` - peso de cada item
- `cap` - capacidade da mochila
- `n` - número de itens definidos dentro de “`val`”

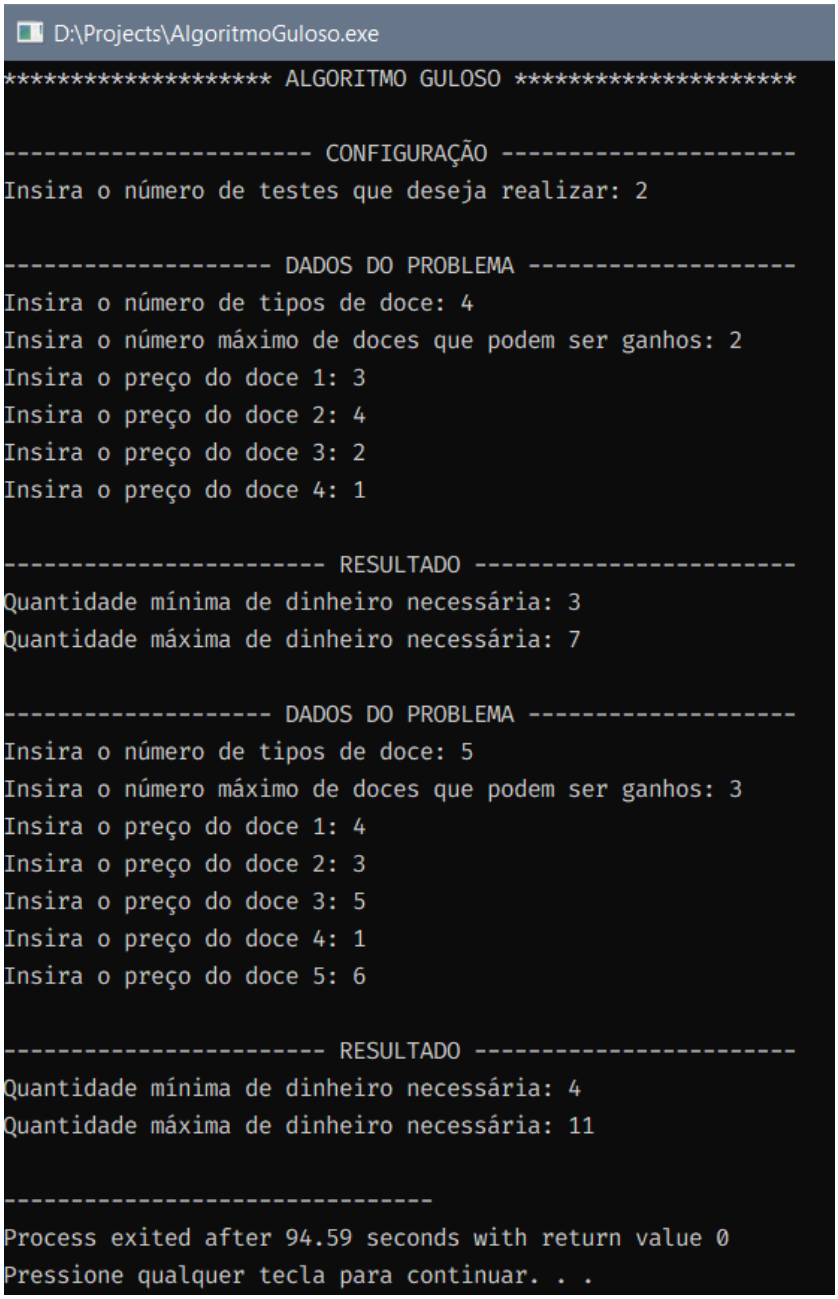
Portanto, a programação dinâmica é eficiente e distinta pelo armazenamento das soluções de várias substâncias em uma tabela, assim o consumo do tempo do algoritmo para resolver a solução que lhe foi determinado acaba sendo proporcional ao tamanho da tabela de soluções.

3. RESULTADOS OBTIDOS

3.1. Algoritmo Guloso

Na Imagem 7 pode-se observar um resultado de execução para o qual no início da execução inseriu-se na seção de "`CONFIGURAÇÃO`" o valor 2, referente ao números de testes que se deseja realizar (valor atribuído à variável `t`); após isso a seção "`DADOS DO PROBLEMA`" é exibida, coletando os valores para as variáveis `n`, `k` e posteriormente para o array '`a`'; depois o resultado da execução do algoritmo é exibido na seção "`RESULTADOS`", mostrando a quantidade máxima e mínima de dinheiro necessária para obtenção dos doces; e por fim as seções "`DADOS DO PROBLEMA`" e "`RESULTADOS`" são exibidas novamente, já que o número de testes definido para realização era 2.

Imagem 7 - Resultados obtidos para o algoritmo guloso



3.2. Programação Dinâmica

Na Imagem 8, inicialmente é pedido ao usuário que seja definido 3 valores, assim como foi mencionado anteriormente. Logo após, é definido um peso respectivo para cada valor que foi descrito na etapa anterior e em seguida a capacidade total de armazenamento da mochila.

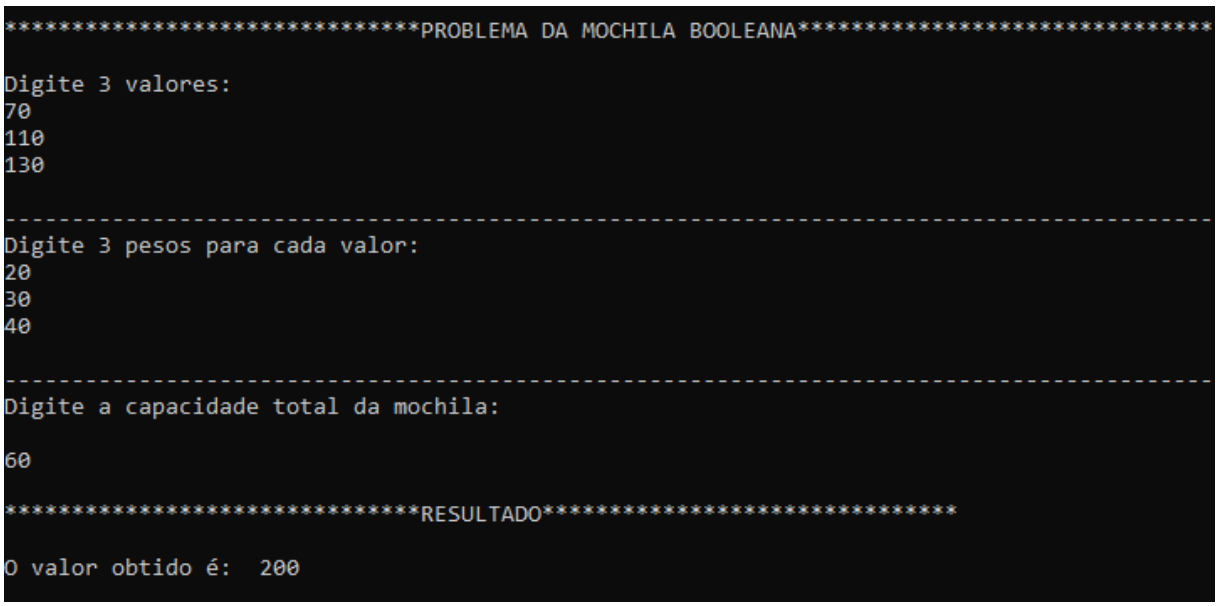
Pode-se observar que a soma dos pesos de cada valor ultrapassa a capacidade total da mochila, pois (20+30+40) > 60, portanto para definir o valor máximo será

utilizado um subconjunto de apenas 2 desses valores para que a capacidade não seja excedida, fazendo a análise é possível perceber que não é possível juntar o `v2` (valor 2) com o `v3` (valor 3), por conta que a soma de seus pesos também excede a capacidade da mochila, (30 + 40) > 60. Portanto, sobram apenas duas possibilidades:

- `v1` com `v2`, onde a soma dos pesos = 50 e valor máximo = 180.
- `v1` com `v3`, onde a soma dos pesos = 60 e valor máximo = 200.

Logo de cara é possível chegar a conclusão que a segunda opção é o valor máximo possível a ser obtido, além de que a soma de seus pesos é igual a capacidade total da mochila, ou seja, está respeitando a capacidade total.

Imagem 8 - Resultado obtido para o algoritmo de programação dinâmica



4. CONCLUSÃO

Após uma sequência de testes que foram realizados para garantir que os algoritmos estão funcionando corretamente, concluiu-se que foi possível aplicar os dois métodos de forma eficiente para resolução de seus respectivos problemas, pois os resultados obtidos estão dentro do esperado.

Além disso, a atividade avaliativa desenvolvida foi de grande valia para aquisição de conhecimentos práticos com base na parte teórica aplicada pelo Docente da disciplina, visto que na construção do código-fonte é necessário a aplicação dos mesmos. O que possibilitou uma compreensão mais extensiva de ambos os métodos de algoritmos (guloso e programação dinâmica).

REFERÊNCIAS

1. FEOFILOFF, Paulo. Método Guloso. IME - USP, 2020. Disponível em: <https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/guloso.html>. Acesso em: 30 de mai. de 2022.

2. UNESP, Programação Competitiva. Algoritmo Guloso / Divisão e Conquista - LPC I 2021. Youtube, 5 de mai. de 2021. Disponível em: <https://www.youtube.com/watch?v=Yq6JlFP520o>. Acesso em: 30 de mai. de 2022.

3. FEOFILOFF, Paulo. Programação Dinâmica. IME - USP, 2020. Disponível em: <https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/dynamic-programming.html#:~:text=A%20característica%20distintiva%20da%20programação,direta%20com%20programação%20de%20computadores>. Acesso em: 30 de mai. de 2022.

4. ROCHA, Anderson. DORINI, Leyza. Algoritmos gulosos: definições e aplicações. 2004. Trabalho Acadêmico. Disponível em: <https://www.ic.unicamp.br/~rocha/msc/complex/algoritmosGulososFinal.pdf>. Acesso em: 30 de mai. de 2022.

5. FEOFILOFF, Paulo. Mochila booleana. IME - USP, 2020. Disponível em: <https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/mochila-bool.html#exr:Mochila-Prog-Din>. Acesso em: 30 de mai. de 2022.

6. MUNARI, Pedro. Programação Dinâmica: Problema da Mochila 0-1, Modelagem, Otimização, Recursão, Pesquisa Operacional. Youtube, 23 de mai. de 2021. Disponível em: <https://www.youtube.com/watch?v=e9vJzakUedY&ab_channel=PedroMunari>. Acesso em: 30 de mai. de 2022.