1- A motivação por trás deste trabalho é resolver um desafio clássico da teoria de algoritmos: o problema de mapeamento comutativo. Esta situação nos obriga a utilizar um conjunto de moedas de diferentes valores para encontrar o número mínimo de moedas necessárias para atingir um valor total predefinido. Uma aplicação prática deste problema é digna de nota em sistemas automatizados de caixa registradora, onde a eficiência na distribuição do troco é crucial.

O objetivo deste estudo é explorar e aplicar métodos eficazes para resolver o problema do mapa de troco, considerando múltiplas abordagens algorítmicas. Pretendemos analisar, implementar e comparar técnicas como programação dinâmica e algoritmos gulosos para encontrar soluções ótimas ou aproximadas. Ao fazê-lo, procuramos compreender não apenas a complexidade computacional associada a estes métodos, mas também obter insights sobre a sua aplicação prática e eficiência no mundo real.

2- O problema é considerado NP, pois verificar se uma solução proposta (um conjunto de moedas ou notas) é válida é fácil, mas encontrar a solução ótima pode exigir a avaliação de muitas combinações possíveis.

3- CÓDIGO EM JAVA:

```
import java.util.List;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Stack;


class CoinChange {


   // C(N,S) = C(N, S') - C(N - Ms, S)

   /*

      Os casos base são:

      C(N, S) = 1, N = 0 --> como o valor é zero, só tem uma solução = não selecionar nenhuma moeda, troco = 0 moedas

      C(N, S) = 0, N < 0 --> não existe solução válida, pois o valor é negativo (inválido)

      C(N, S) = 0, N >=1, S = {} --> não temos moedas para dar o troco, portanto não tem solução
```

```
    */

   static int calculate(int N, List<Integer> moedas, Stack<Integer> sol,
List<Integer[]> conj_solucoes) {
       if ((N<0) || (moedas.size()==0)) return 0;
       if (N==0) {
          // acrescenta o conjunto "sol" as solucoes possiveis
          conj_solucoes.add( sol.toArray(new Integer[sol.size()]));
          return 1;
       }

       Integer moeda = moedas.get(0); // obtem uma novo moeda para ver se faz
parte da solucao
       sol.push(moeda); // acrescento ao conjunto corrente que representa a
solucao
       int v1 = calculate(N-moeda, moedas, sol, conj_solucoes); // 1o termo da
equacao

       sol.pop(); // como vou tentar a 2a opcao, que nao considera a moeda,
retiro do conj. corrente de solucao

       List<Integer> moedas_novo = new ArrayList<>(moedas);
       moedas_novo.remove(moeda);
       int v2 = calculate(N, moedas_novo, sol, conj_solucoes); // 2o termo da
equacao

       return v1 + v2; // retorna a equacao de recorrencia
   }

   public static void main(String[] args) {
       int N = 4; // valor do troco
       List<Integer> moedas = new ArrayList<>(Arrays.asList(1, 2, 3));
       Stack<Integer> sol = new Stack<>();
```

```
        List<Integer[]> conj_solucoes = new ArrayList<>();


        int num = calculate(N, moedas, sol, conj_solucoes);
        System.out.println("Localizadas " + num + " maneiras de obter o troco.");
        System.out.println("Elas sao:");
        for(Integer[] s : conj_solucoes)
            System.out.println("* " + Arrays.toString(s));
    }
}
```

Este é um código Java de exemplo que implementa um algoritmo para encontrar todas as maneiras de dar troco para um determinado valor (N) usando um conjunto de moedas específico. Aqui estão algumas características principais do código:

Função calculate:

Esta função é recursiva e utiliza a programação dinâmica para encontrar todas as combinações possíveis de moedas que somam ao valor desejado N.

Usa a equação de recorrência: C(N, S) = C(N, S') - C(N - Ms, S), onde C(N, S) representa o número de maneiras de obter troco N com as moedas no conjunto S.

Casos Base:

Retorna 0 se N for negativo ou se o conjunto de moedas estiver vazio.

Retorna 1 se N for zero, o que significa que uma solução foi encontrada. O conjunto de solução é adicionado à lista de conjuntos de soluções (conj_solucoes).

Recursão:

A função faz chamadas recursivas para dois casos: considerando a moeda atual (moeda) na solução e não considerando a moeda.

Utiliza uma pilha para manter o conjunto corrente de moedas na solução.

O código usa conceitos avançados de recursão e manipulação de listas em Java para gerar todas as combinações possíveis de moedas.

4- O algoritmo escolhido foi o guloso, desenvolvido por um dos colegas.
Link do código: https://github.com/ThanatosPycaro/mapa-de-troco/blob/main

5- Em relação à classificação Big(O), o algoritmo possui uma complexidade linear, pois o número de operações executadas é proporcional ao tamanho da entrada (valor do salário). Então, a complexidade é O(n), onde n representa o valor do salário.

6- Principais estratégias:

-Utilização de variáveis para armazenar o valor do salário, quantidade de notas e moedas, e o resto do valor após cada cálculo.

-Utilização da classe Scanner para obter o valor do salário digitado pelo usuário.

-Utilização de condicionais if para verificar se o valor do salário é maior ou igual a determinado valor e realizar os cálculos correspondentes.

-Utilização do operador de módulo (%) para obter o resto da divisão entre o valor do salário e o valor de cada nota ou moeda.

-Utilização da classe Math para arredondar valores decimais.

-Utilização de operações matemáticas simples para calcular a quantidade de notas e moedas necessárias.

7-
```
import java.util.Arrays;


public class MapaDeTroco {


    public static int[] calcularTroco(int[] valoresMoedas, int troco) {
        Arrays.sort(valoresMoedas);
        int[] resultado = new int[valoresMoedas.length];


        for (int i = valoresMoedas.length - 1; i >= 0; i--) {
            while (troco >= valoresMoedas[i]) {
                troco -= valoresMoedas[i];
                resultado[i]++;
            }
        }


        return resultado;
```

```
    }

    public static void main(String[] args) {
        int[] valoresMoedas = {1, 5, 10, 25}; // Exemplo de moedas (em centavos)
        int troco = 63; // Exemplo de valor do troco em centavos

        int[] resultadoTroco = calcularTtroco(valoresMoedas, troco);

        System.out.println("Troco mínimo para " + troco + " centavos:");
        for (int i = 0; i < valoresMoedas.length; i++) {
            System.out.println(resultadoTroco[i] + " moeda(s) de " +
valoresMoedas[i] + " centavos");
        }
    }
}
```

No quesito estratégia e complexidade os algoritmos são bem parecidos, usam de NP, além das variáveis e condicionais.

8- A linguagem escolhida para resolução do problema foi o Java, escolhemos ela pois é uma linguagem que já temos um conhecimento sobre ela e algumas bibliotecas.