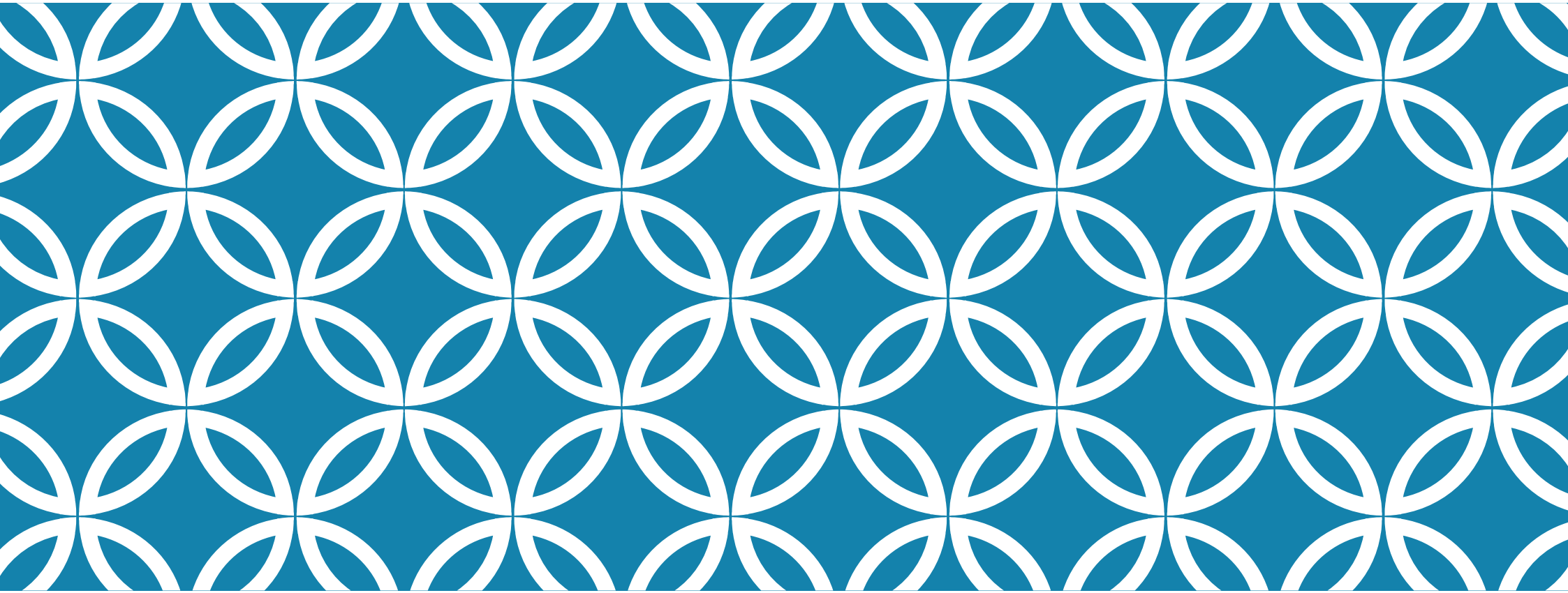


CONCILIAÇÃO DE TRANSAÇÕES SUSPEITAS (SUBSET SUM)

Victor Leonardo Mascarenhas Soares Horta

SUMÁRIO

O PROBLEMA

Uma equipe de **Auditoria/Compliance** investiga possível **"rachadinha"/lavagem** em um órgão. Há um **valor de referência** (um repasse total suspeito) de **R$ 7.358.450,23** que, supostamente, teria sido "montado" somando várias transações menores realizadas ao longo do mês.

A pergunta é objetiva: **existe um conjunto de transações que, somadas, dá exatamente R$ 7.358.450,23?**

Se existir, queremos **listar quais transações** compõem essa soma (para juntar como evidência).

O PROBLEMA

Entrada

Transações $\boldsymbol{S = \{a_1, a_2, \ldots, a_n\}}$ **,com** $\boldsymbol{a_i \in \mathbb{Z}_{\geq 0}}$ (valores **inteiros,** e.g.**, e conter centavos ou não.**)

Um **alvo** $T \in \mathbb{Z}_{\geq 0}$.

Pergunta

Existe um subconjunto $S' \subseteq S$ tal que

$$\sum_{x \in S'} x = T \ ?$$

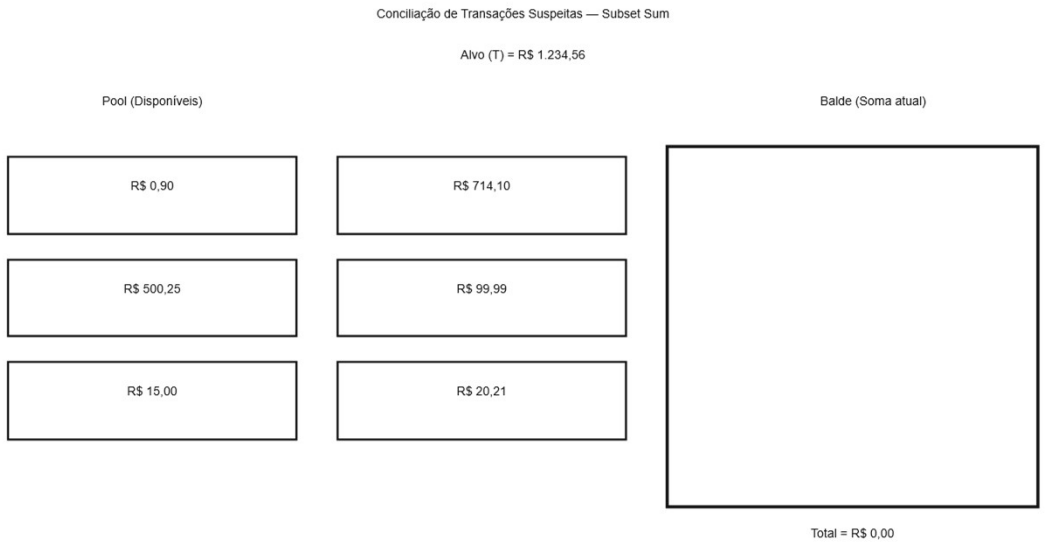
Saída

SIM/NÃO.

- Se **SIM,** também retornamos **quais índices/IDs** compõem S'.

ENTRADAS E EXEMPLOS

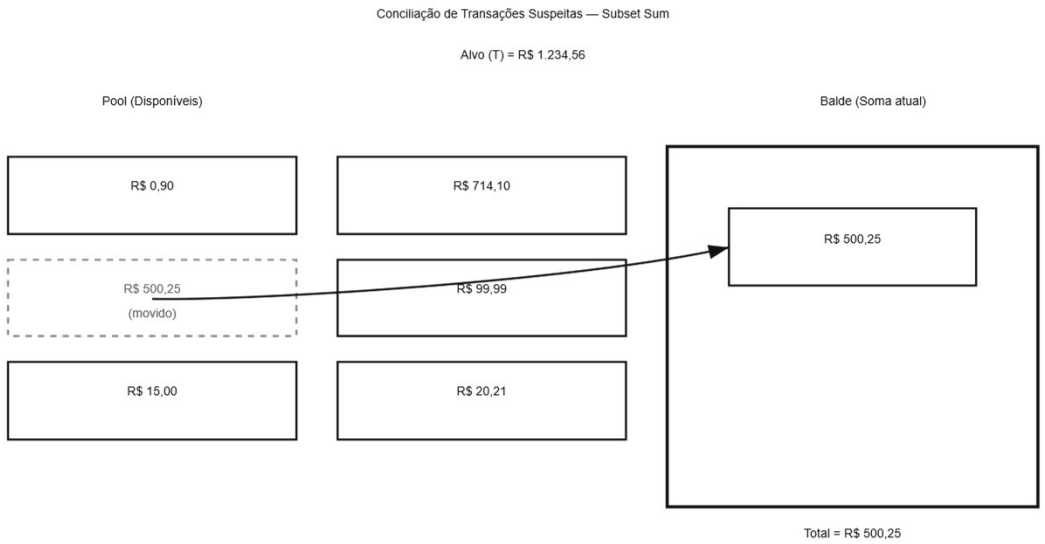
CASO 1: O ALVO É ENCONTRADO



Nesta primeira imagem: todas as transações estão à esquerda e o balde de conciliação está vazio.

ENTRADAS E EXEMPLOS

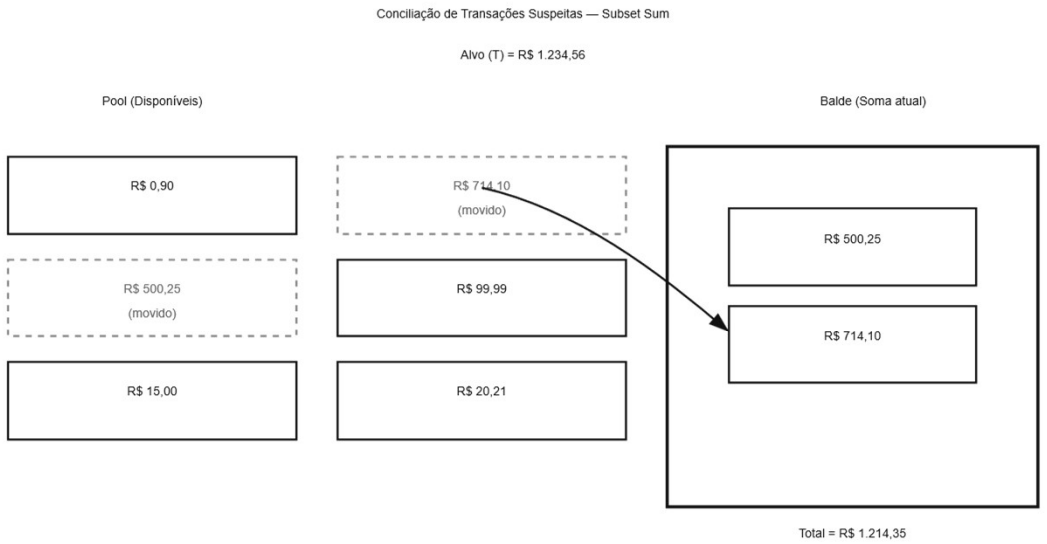
CASO 1: O ALVO É ENCONTRADO



Nesta segunda imagem: a transação de R$ 500,25 foi selecionada e movida ao balde. A soma atual reflete esse valor.

ENTRADAS E EXEMPLOS

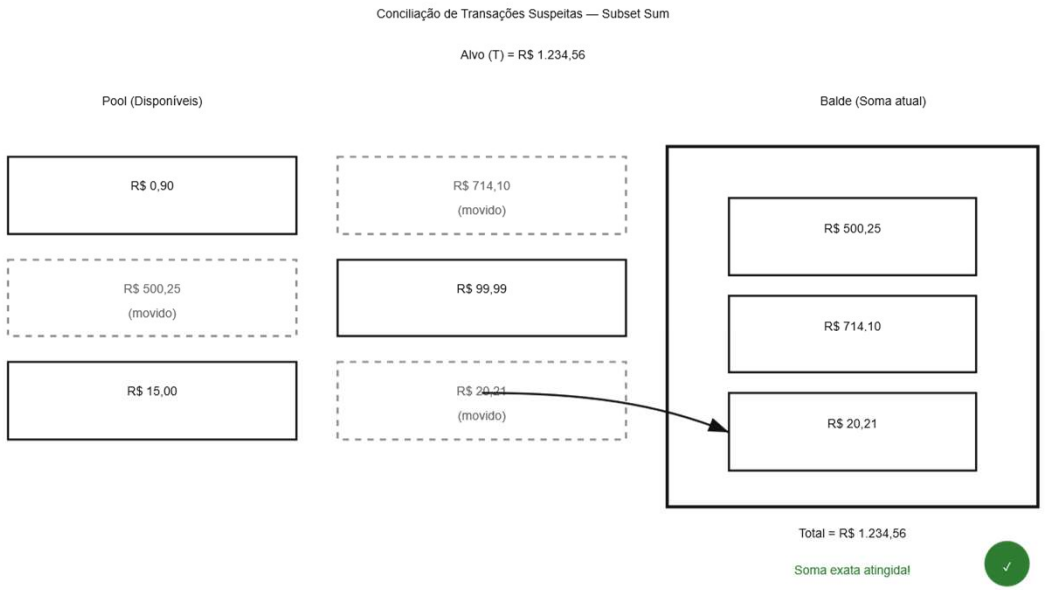
CASO 1: O ALVO É ENCONTRADO



Nesta terceira imagem: R$ 714,10 foi adicionado ao balde junto com R$ 500,25. A soma atual (R$ 1.214,35) se aproxima do alvo.

ENTRADAS E EXEMPLOS

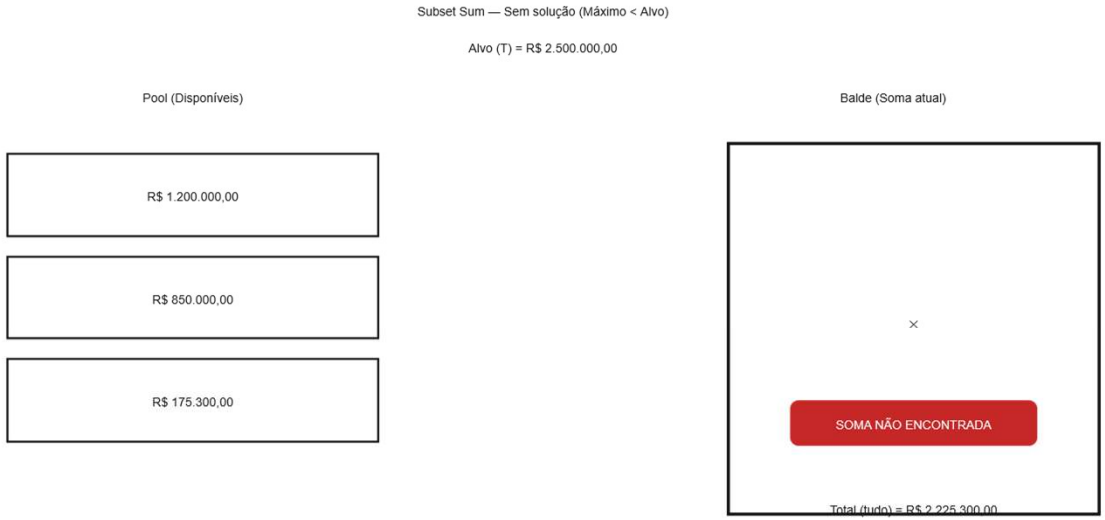
CASO 1: O ALVO É ENCONTRADO



Peça final (R$ 20,21) adicionada. Total igual ao alvo (R$ 1.234,56). Resultado validado com check verde.

ENTRADAS E EXEMPLOS

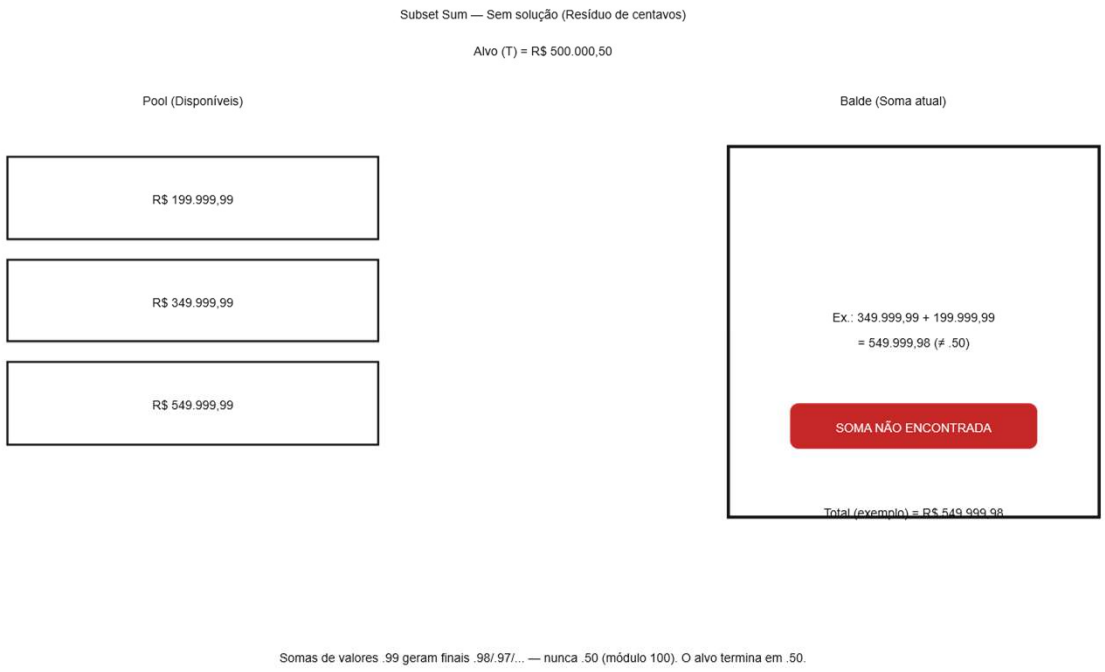
CASO 2: O ALVO É MENOR QUE O CONJUNTO DE TRANSAÇÕES



Mesmo somando tudo: 1.200.000,00 + 850.000,00 + 175.300,00 = 2.225.300,00 < 2.500.000,00.

ENTRADAS E EXEMPLOS

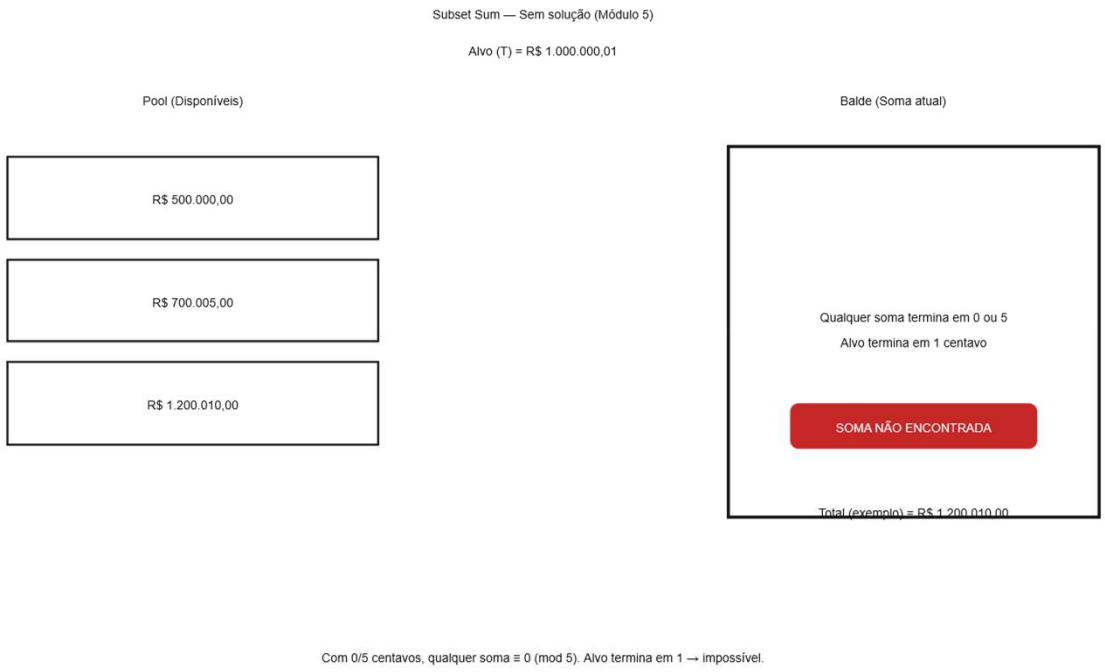
CASO 3: O ALVO É MAIOR QUE O CONJUNTO DE TRANSAÇÕES POR CENTAVOS



Somas de valores .99 geram finais .98/.97/... — nunca .50 (módulo 100). O alvo termina em .50.

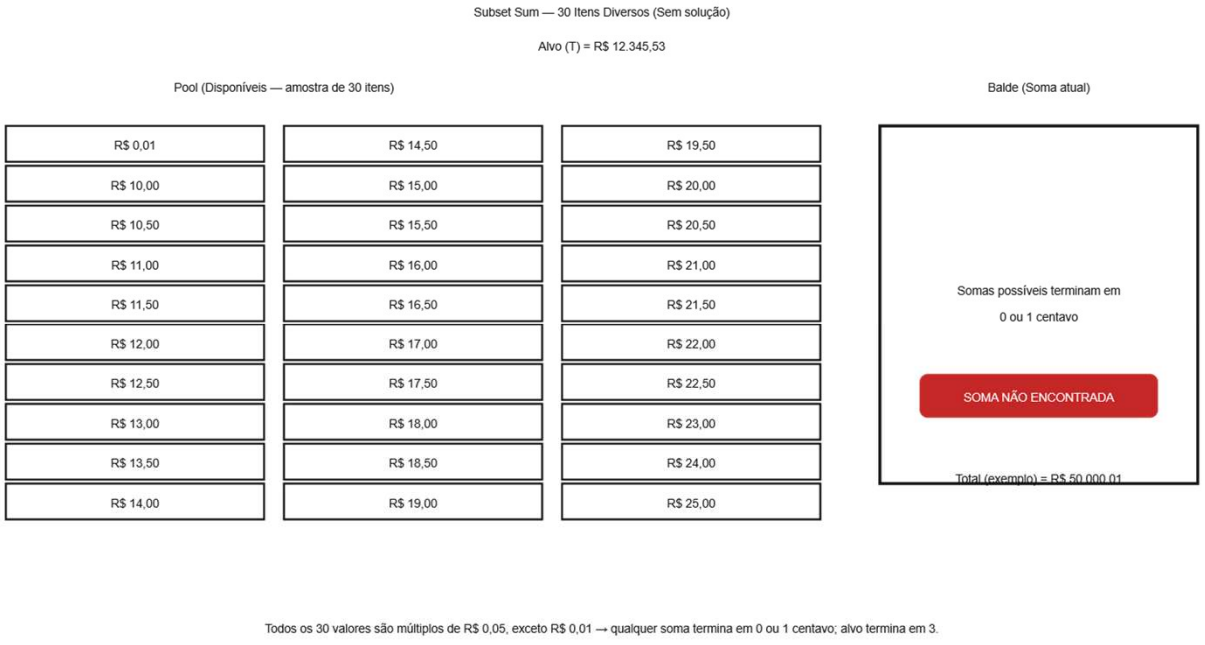
ENTRADAS E EXEMPLOS

CASO 4: O CONJUNTO DE TRANSAÇÕES NÃO POSSUI POSSIBILIDADE DE SOMA = T



ENTRADAS E EXEMPLOS

CASO 4: O CONJUNTO DE TRANSAÇÕES É GRANDE



CLASSIFICAÇÃO DO PROBLEMA

ATÉ AGORA, EXEMPLIFICAMOS COM CASOS QUE PODEM SER FEITOS À MÃO OU COM UMA PEQUENA AJUDA DE UMA PLANILHA.

MAS CONSIDERANDO UM UNIVERSO MAIOR, POR EXEMPLO, DE UM POLÍTICO, COM ASSESSORES, CABO ELEITORAL, FAMILIARES, ETC., O CONJUNTO DE TRANSAÇÕES PODE SER MAIOR.

CLASSIFICAÇÃO DO PROBLEMA

Definição (decisão).
Existe um subconjunto $S' \subseteq S$ tal que

$$\sum_{x \in S'} x = T \ ?$$

QUAL, ENTÃO, É SUA CLASSIFICAÇÃO?

1 - Está em NP (verificador rápido).
Se alguém entrega S' o "certificado"), **verificar** é só somar e comparar com T— custo $O(n)$.

2 - É NP-Completo***

Difícil como os clássicos: problemas conhecidos (ex.: **PARTITION,** e via cadeias de redução a partir de **3-SAT** na lista de Karp) podem ser "traduzidos" para uma única soma-alvo.

Consequência prática: não se conhece algoritmo **geral** que ache a solução **sempre** em tempo polinomial.

CLASSIFICAÇÃO DO PROBLEMA

***Fracamente NP-Completo (pseudo-polinomial existe).**

Há DP em tempo O(n·T)(e memória O(n·T)ou O(T)), eficiente quando T é moderado.

É "pseudo-polinomial" porque depende do valor numérico de T, não só do seu tamanho em bits (logT).

Limites & alternativas.

Se Té muito grande, DP fica inviável → usa-se **meet-in-the-middle** $O\left(2^{n/2}\right)$ou heurísticas.

A versão **otimização** ("ache um subconjunto que chegue o mais perto de T") é **NP-difícil**.

ALGORITMO

Funções auxiliares do código

```python
def fmt_centavos(x: int) -> str:
    s = f"{x/100:.2f}"
    return "R$ " + s.replace(".", ",")

def parse_pool_field(field: str) -> List[int]:
    s = field.strip()
    try:
        if s.startswith("[") and s.endswith("]"):
            lst = ast.literal_eval(s)
            return [int(x) for x in lst]
    except Exception:
        pass
    sep = ";" if ";" in s else ","
    parts = [p.strip() for p in s.split(sep) if p.strip()]
    return [int(p) for p in parts]
```

fmt_centavos(x)
Converte um valor inteiro em centavos para o formato monetário brasileiro.
Ex.: 450093 → "R$ 4.500,93".
Serve apenas para exibir resultados de forma clara.

parse_pool_field(field)
Lê o campo pool do CSV e converte para uma lista de inteiros, aceitando vários formatos:
"[90, 50025, 71410]" (lista Python)
"90,50025,71410"
"90; 50025; 71410"
A função detecta o formato automaticamente e retorna sempre algo como:
[90, 50025, 71410].

ALGORITMO

DP BITSET + RECONSTRUÇÃO

```python
# ------------ DP bitset + reconstrução ------------

def bitset_dp_with_reconstruction(pool: List[int], T: int) -> Tuple[bool, List[int]]:
    n = len(pool)
    layers: List[int] = []
    dp = 1  # bit 0 ligado (soma 0)
    layers.append(dp)
    for v in pool:
        if v <= T:
            dp |= (dp << v)
        layers.append(dp)
    if ((dp >> T) & 1) == 0:
        return False, []
    subset: List[int] = []
    curT = T
    for i in range(n, 0, -1):
        v = pool[i-1]
        prev = layers[i-1]
        if ((prev >> curT) & 1) == 1:
            continue
        if curT >= v and ((prev >> (curT - v)) & 1) == 1:
            subset.append(v)
            curT -= v
    subset.reverse()
    return True, subset
```

Função bitset_dp_with_reconstruction
Implementa o **Subset Sum por Programação Dinâmica (bitset)** com **reconstrução do subconjunto.**

Inicialização:
dp = 1 → apenas soma 0 possível.
layers guarda cada "foto" do bitset após cada valor.

Transição da DP:
Para cada valor v do pool:
dp |= (dp << v)
Isso marca como possíveis tanto as somas antigas quanto as somas obtidas ao adicionar v.

Checagem de solução:
Se o bit T não estiver ligado → **não existe subconjunto.**

Reconstrução:
Percorre os itens **de trás pra frente,** usando layers para decidir:

- se curT já era possível antes → item **não foi usado**
- senão, se curT - v era possível → item **foi usado** e curT -= v

Saída:
Retorna **(True, subset)** se encontrou uma combinação que soma T;
caso contrário, **(False, []).**

ALGORITMO

Execução de 1 estancia

```python
# ---------------- Execução de 1 instância ----------------

def run_instance(name: str, pool: List[int], target: int) -> None:
    print("=" * 80)
    print(f"Instância: {name}")
    print(f"Alvo   : {target} -> {fmt_centavos(target)}")
    print(f"Itens : {len(pool)} valores")
    ok, subset = bitset_dp_with_reconstruction(pool, target)
    if ok:
        subtotal = sum(subset)
        print("\n>>> ENCONTRADO")
        print(f"Soma do subconjunto: {subtotal} -> {fmt_centavos(subtotal)}")
        print("Subconjunto (centavos):", subset)
        print("Subconjunto (R$)      :", [fmt_centavos(v) for v in subset])
    else:
        print("\n>>> NÃO ENCONTRADO")
```

Função run_instance

Executa **uma instância** do Subset Sum usando o algoritmo de DP.

Imprime as informações básicas:

- Nome da instância
- Alvo (em centavos e em R$)
- Quantidade de itens no pool

Chama a função **bitset_dp_with_reconstruction** para verificar se existe subconjunto que soma o alvo.

Se encontrou:

- Calcula subtotal = sum(subset)
- Exibe:
  - "ENCONTRADO"
  - Soma do subconjunto (centavos e R$)
  - Lista de valores usados

Se não encontrou:

- Exibe: **"NÃO ENCONTRADO"**

ALGORITMO

MAIN

```python
def main():
    # 1) Pega caminho do CSV: argumento ou prompt
    if len(sys.argv) >= 2:
        csv_path = sys.argv[1]
    else:
        print("Informe o caminho COMPLETO do CSV gerado (ex.: C:/Users/voce/Documentos/transacoes.csv)")
        csv_path = input("Caminho do CSV: ").strip().strip('"').strip("'")

    # 2) Valida existência; se não existir, pergunta novamente
    while not csv_path or not os.path.exists(csv_path):
        print("Arquivo não encontrado. Verifique o caminho e tente novamente.")
        csv_path = input("Caminho do CSV: ").strip().strip('"').strip("'")

    # 3) Lê e processa
    with open(csv_path, newline="", encoding="utf-8") as f:
        reader = csv.DictReader(f)
        required = {"name", "pool", "target"}
        if reader.fieldnames is None or not required.issubset(set(reader.fieldnames)):
            print("Erro: o CSV deve ter cabeçalho com colunas: name,pool,target")
            sys.exit(1)

        for i, row in enumerate(reader, start=2):
            try:
                name = row["name"]
                pool = parse_pool_field(row["pool"])
                target = int(row["target"].strip())
            except Exception as exc:
                print(f"[Linha {i}] erro ao parsear: {exc}")
                continue
            if target < 0 or any(v < 0 for v in pool):
                print(f"[{name}] Ignorada: valores/target negativos não são suportados.")
                continue
            run_instance(name, pool, target)

if __name__ == "__main__":
    main()
```

Função main()

Obtém o caminho do CSV

Se passado como argumento: usa direto

Caso contrário: pergunta ao usuário

Repetir até encontrar um arquivo válido

Valida o CSV

Abre via csv.DictReader

Exige cabeçalho: name, pool, target

Se faltar campo → encerra com erro

Processa cada linha do CSV

Extrai:

name → nome da instância

pool → convertido em lista de inteiros (parse_pool_field)

target → convertido para inteiro (centavos)

Ignora linhas com valores negativos ou mal formatadas

Executa a instância

Para cada linha válida, chama: run_instance(name, pool, target)

ALGORITMO

DEMONSTRAÇÃO

OBS.: TODO O CODIGO DO SUNSET E O CÓDIGO GERADOR DE TRANSAÇÕES E ALVO, ESTÃO DISPONÍVEIS NO LINK DO GITHUB.

OBRIGADO!