Distribuidor de capacidade (solução)

Tipo de problema: paradigmas, divisão e conquista.

A condição de que o limite de um projeto deve ser proporcional à sua importância implica que devemos calcular um valor **x** que é o **limite por unidade de importância**. O limite de um projeto é $x * importance$ se este valor estiver entre $minLimit$ e $maxLimit$, ou $minLimit$ ou $maxLimit$ caso contrário.

Agora, resolvamos um problema um pouco diferente: em vez de calcular um valor x, suponhamos que x seja dado. Devemos verificar se esse x é válido. Os limites podem ser calculados diretamente: o limite de cada projeto será dado por $limit[i] = \min(\text{maxLimit[i]}, \min(\text{minLimit[i]}, \text{x} * \text{importance[i]}))$. O valor x é válido se e somente se $\sum_{i=0}^{n-1} \min(usage[i], limit[i]) \leq capacity$. Essa verificação pode ser feita em $O(n)$, onde n é o número de projetos.

Voltando ao problema original, devemos calcular o máximo x que seja válido para minimizar o excesso. Percebemos que se um valor x_0 é válido, então todo valor $x \leq x_0$ também é válido. Também percebemos que se um valor x_0 é inválido, então todo valor $x \geq x_0$ também é inválido. Assim, podemos fazer **busca binária** do maior valor de x que é válido. O algoritmo executa em $O(n \log C)$, onde n é número de projetos e C é a capacidade do cliente.

Código:

```
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

public class CapacityDistributor {

    public record Project(int minLimit, int maxLimit, int importance) {}

    static List<Integer> distribute(List<Project> projects, List<Integer>
usage, int capacity) {
        int lo = 0, hi = capacity+1;
        while (hi > lo + 1) {
            int x = (lo + hi) / 2;
            if (isValid(projects, usage, capacity, x)) {
                lo = x;
            } else {
                hi = x;
            }
        }
        List<Integer> limits = new ArrayList<>();
```

```
        for (Project project : projects) {
            limits.add(calculateLimit(project, lo));
        }
        return limits;
    }

    static boolean isValid(List<Project> projects, List<Integer> usage,
int capacity, int x) {
        int combinedUsage = 0;
        for (int i = 0; i < projects.size(); i++) {
            combinedUsage += Math.min(usage.get(i),
calculateLimit(projects.get(i), x));
        }
        return combinedUsage <= capacity;
    }

    static int calculateLimit(Project project, int x) {
        int limit = x * project.importance();
        limit = Math.min(limit, project.maxLimit());
        limit = Math.max(limit, project.minLimit());
        return limit;
    }

    public static void main(String[] args) {
        List<Project> projects = List.of(
            new Project(100, 1000, 1),
            new Project(100, 1000, 1),
            new Project(100, 1000, 3),
            new Project(100, 1000, 5),
            new Project(0, 100, 5));
        List<Integer> usage = List.of(
            0,
            800,
            800,
            100,
            400);
        List<Integer> expectedLimits = List.of(
            200,
            200,
            600,
            1000,
            100);
        List<Integer> actualLimits = distribute(projects, usage, 1000);
        System.out.println("Wanted: " + expectedLimits);
        System.out.println("Got: " + actualLimits);
    }
}
```