Descrição do Problema

No desenvolvimento de teclados inteligentes e sistemas de processamento de linguagem natural, um desafio comum é a **Segmentação de Fluxo (Stream Segmentation)**. Quando um usuário utiliza funcionalidades como *Gesture Typing*, o sistema pode gerar uma string contínua de caracteres sem espaços.

O Desafio: Dada uma string contínua S e um dicionário de palavras válidas D acompanhado de suas frequências de uso, implemente um algoritmo que reconstrua a frase original inserindo os espaços nos locais mais prováveis.

- **Exemplo de Entrada:** "umacasaeumcarro"
- **Saída Esperada:** "uma casa e um carro"

2. Requisitos Técnicos

1. **Eficiência:** O algoritmo deve ser capaz de processar frases longas sem sofrer com explosão combinatória.
2. **Ambiguidade:** Deve-se resolver casos de palavras sobrepostas (ex: "casamento" vs "casa" + "mento").
3. **Heurística:** Utilizar a probabilidade das palavras (Unigramas) para decidir entre múltiplas segmentações válidas.

Arquitetura da Solução Proposta

Estrutura de Dados (Trie)

Utilizamos uma **Trie (Árvore de Prefixo)** para armazenar o dicionário. Isso permite que, ao percorrer a string, saibamos instantaneamente se um prefixo é uma palavra válida ou o início de uma.

Algoritmo (Programação Dinâmica)

A solução utiliza **Programação Dinâmica (DP)** para evitar o reprocessamento de sub-strings. O array dp[i] armazena a melhor segmentação encontrada até o índice i da string.

Esboço da Implementação (Python)

```
import math

def solve_segmentation(text, dictionary, frequencies):
    n = len(text)
    # dp[i] armazena (log_probabilidade_acumulada, frase_segmentada)
    dp = [(-float('inf'), "")] * (n + 1)
    dp[0] = (0, "")

    for i in range(n):
        if dp[i][0] == -float('inf'):
            continue

        # Busca por palavras que começam no índice i
        for j in range(i + 1, n + 1):
            word = text[i:j]
            if word in dictionary:
                # Usamos log para evitar underflow de números decimais pequenos
                prob = math.log(frequencies.get(word, 1e-6))
                new_score = dp[i][0] + prob

                if new_score > dp[j][0]:
                    dp[j] = (new_score, (dp[i][1] + " " + word).strip())

    return dp[n][1]
```