Distância de Edição

Quando se deseja comparar duas palavras[Strings] para saber o custo necessário para transformar uma na outra, normalmente é usado o algoritmo de distância de edição, que calcula a melhor forma de resolver o problema utilizando programação dinâmica.

Distância de Edição

- Dadas duas palavras *t* e *p*, definimos a **distância de edição** *D[t, p]* entre elas como o custo total *mínimo necessário para transformar t em p ou vice-versa*.
- Para calcular tal custo, define-se custos de operações de edição de uma palavra, que podem ser:
  - Substituição [*Replacement*]
  - Inserção [*Insertion*]
  - Remoção [*Deletion*]
  - Pareamento [*Match*]
- A operação de pareamento não é contada na distância de edição[ou seja, custo 0].
- Sendo assim, valores menores indicam menor distância de edição

Distância de Edição

- Para usar Programação Dinâmica:
  - Um primeiro passo (usualmente) é pensar/fazer uma versão recursiva. Assim, dadas duas Strings S e T:

```
ED(S, T, i, j): int
// S: String inicial, T: String final, i: [1..m], j:[1..n]
//     retorna o número mínimo de edições quando comparando
//     S[i] com T[j]. m é o tamanho de S, n o tamanho de T
//
Caso Base:
   Quando ficamos sem caracteres para comparer em S ou em T. Se em ambas, o
   resultado é 0. Se uma das duas, retorna o restante dos caracteres da que não
   está vazia;

Casos Recursivos
   Se S[i] == T[i], chame recursivamente ED(S, T, i-1, j-1) (foi match, não
   precisa fazer nada nesta posição, o custo é zero.
   Se não, três chamadas recursivas são necessárias:
   • Substituição: ED(S, T, i-1, j-1) + 1
   • Inserção:     ED(S, T, i, j-1) + 1
   • Remoção:      ED(S, T, i-1, j) + 1
   • Retorne a que resultar em menor custo
```