Edit Distance (Dasgupta et al., Seção 6.3)

- **Entrada:** Duas strings $x[1 \dots m]$ e $y[1 \dots n]$
- **Objetivo:** Transformar x em y com um mínimo de operações
- **Operações:** Inserir letra, Remover letra, **Substituir letra**.

Exemplo: ed(SNOWY,SUNNY)

S N O W Y	S _ N O W Y	S _ N O W Y
S U N N Y	S U N _ N Y	S U N N _ Y

- Edit Distance = 3

1. Propriedade da Subestrutura Ótima

- Em linguagem popular, pergunta-se se "*pedaços da solução ótima são soluções ótimas de pedaços do problema*".
- ed(SNOWY,SUNNY) = ed(SNOW,SUNN) = 1+ed(SNO,SUNN) = 2+ed(SN,SUN) = 2+ed(S,SU) = 3
- ed(SNOWY,SUNNY) = ed(SNOW,SUNN) = 1+ed(SNO,SUN) = 2+ed(SN,SU) = 3

2. Equação de Recorrência - Algoritmo recursivo simples

- Seja $ed(i,j) = ed(x[1 \ldots i], y[1 \ldots j])$
- $ed(0,j) = j$; $\qquad ed(i,0) = i$
- Testar as 3 operações possíveis com relação às últimas letras
- Seja $dif_{i,j} = 1$ se $x_i \neq y_j$ e seja 0, caso contrário.
-

$$ed(i,j) = \min\left\{\underbrace{ed(i-1,j-1)+dif_{i,j}}_{\text{Substituição}},\ \underbrace{ed(i,j-1)+1}_{\text{Inserção}},\ \underbrace{ed(i-1,j)+1}_{\text{Remoção}}\right\},$$

Edit-REC(x,y,i,j)

se $(i = 0)$ **então retorne** j
se $(j = 0)$ **então retorne** i
$a \leftarrow$ *Edit-REC*$(x,y,i-1,j-1) + dif_{i,j}$
$b \leftarrow$ *Edit-REC*$(x,y,i,j-1) + 1$
$c \leftarrow$ *Edit-REC*$(x,y,i-1,j) + 1$
retorne $\min\{a,b,c\}$

2. Equação de Recorrência - Algoritmo recursivo simples

Sobreposição de Subproblemas

- **Chamada inicial**: $Edit - REC(x, y, m, n)$
- **Tempo**: Exponencial $\Omega(3^{\min\{m,n\}}) \;\Rightarrow\; \Omega(3^n)$ para $m = n$
- **Indução**: $T(m,n) \geq T(m-1,n) + T(m,n-1) + T(m-1,n-1) \geq 3 \cdot T(m-1,n-1) \;\geq\; 3 \cdot 3^{\min\{m,n\}-1} \;=\; 3^{\min\{m,n\}}$
- **Superposição**: A instância $(m-1, n-1)$ é chamada por (m,n) e $(m-1,n)$ e $(m,n-1)$.

Edit-REC(x, y, i, j)

se $(i = 0)$ **então retorne** j
se $(j = 0)$ **então retorne** i
$a \leftarrow$ *Edit-REC*$(x, y, i-1, j-1) + dif_{i,j}$
$b \leftarrow$ *Edit-REC*$(x, y, i, j-1) + 1$
$c \leftarrow$ *Edit-REC*$(x, y, i-1, j) + 1$
retorne $\min\{a, b, c\}$

2b. Memoização (Alg. Recursivo + memória) - Top Down

Edit-memo(x, y, m, n, ed)

para $i \leftarrow 0$ **até** m: $ed[i, 0] \leftarrow i$
para $j \leftarrow 0$ **até** n: $ed[0, j] \leftarrow j$
para $i \leftarrow 1$ **até** m:
    para $j \leftarrow 1$ **até** n:
        $ed[i, j] \leftarrow -1$
retorne *Edit-REC-memo*(x, y, m, n, ed)

$Edit - REC - memo(x, y, i, j, ed)$

se $(ed[i, j] \geq 0)$ **então retorne** $ed[i, j]$
$a \leftarrow$ *Edit-REC-memo*$(x, y, i-1, j-1, ed) + dif_{i,j}$
$b \leftarrow$ *Edit-REC-memo*$(x, y, i, j-1, ed) + 1$
$c \leftarrow$ *Edit-REC-memo*$(x, y, i-1, j, ed) + 1$
$ed[i, j] \leftarrow \min\{a, b, c\}$
retorne $ed[i, j]$

3. Algoritmo p/ Valor Ótimo (Bottom-up, não recursivo)

.	-	S	U	N	N	Y
-	0	1	2	3	4	5
S	1	0	1	2	3	4
N	2	1	1	1	2	3
O	3	2	2	2	2	3
W	4	3	3	3	3	3
Y	5	4	4	4	4	3

Edit-Distance-PD(x, y, m, n)

```
Criar matriz ed[0 ... m,  0 ... n]
para i ← 0 até m:     ed[i, 0]  ←  i
para j ← 0 até n:     ed[0, j]  ←  j
para i ← 1 até m:
      para j ← 1 até n:
            ed[i, j] ← min{ed[i-1,j-1]+dif_{i,j}, ed[i, j-1]+1, ed[i-1,j]+1}
retorne ed[m, n]
```

3. Algoritmo p/ Valor Ótimo (Bottom-up, não recursivo)

.	-	S	U	N	N	Y
-	0	1	2	3	4	5
S	1	0	1	2	3	4
N	2	1	1	1	2	3
O	3	2	2	2	2	3
W	4	3	3	3	3	3
Y	5	4	4	4	4	3

Todas as soluções ótimas possíveis:

- S N O W Y   S _ N O W Y   S _ N O W Y
- S U N N Y   S U N _ N Y   S U N N _ Y

3. Algoritmo p/ Valor Ótimo (Bottom-up, não recursivo)

.	-	S	U	N	N	Y
-	0	1	2	3	4	5
S	1	0	1	2	3	4
N	2	1	1	1	2	3
O	3	2	2	2	2	3
W	4	3	3	3	3	3
Y	5	4	4	4	4	3

Todas as soluções ótimas possíveis:

- S N O W Y   S _ N O W Y   S _ N O W Y
- S U N N Y   S U N _ N Y   S U N N _ Y

3. Algoritmo p/ Valor Ótimo (Bottom-up, não recursivo)

.	-	S	U	N	N	Y
-	0	1	2	3	4	5
S	1	0	1	2	3	4
N	2	1	1	1	2	3
O	3	2	2	2	2	3
W	4	3	3	3	3	3
Y	5	4	4	4	4	3

Todas as soluções ótimas possíveis:

- S N O W Y   S _ N O W Y   S _ N O W Y
- S U N N Y   S U N _ N Y   S U N N _ Y

3. Algoritmo p/ Valor Ótimo (Bottom-up, não recursivo)

.	-	S	U	N	N	Y
-	0	1	2	3	4	5
S	1	↖0	←1	←2	←3	←4
N	2	↑1	↖1	↖1	←↖2	←3
O	3	↑2	↑↖2	↑↖2	↖2	←↖3
W	4	↑3	↑↖3	↑↖3	↑↖3	↖3
Y	5	↑4	↑↖4	↑↖4	↑↖4	↖3

Todas as soluções ótimas possíveis:

- S N O W Y   S _ N O W Y   S _ N O W Y
- S U N N   Y   S U N _ N   Y   S U N   N _ Y

3. Algoritmo p/ Valor Ótimo (Bottom-up, não recursivo)

.	-	S	U	N	N	Y
-	0	1	2	3	4	5
S	1	$\nwarrow 0$	$\leftarrow 1$	$\leftarrow 2$	$\leftarrow 3$	$\leftarrow 4$
N	2	$\uparrow 1$	$\nwarrow 1$	$\nwarrow 1$	$\leftarrow\nwarrow 2$	$\leftarrow 3$
O	3	$\uparrow 2$	$\uparrow\nwarrow 2$	$\uparrow\nwarrow 2$	$\nwarrow 2$	$\leftarrow\nwarrow 3$
W	4	$\uparrow 3$	$\uparrow\nwarrow 3$	$\uparrow\nwarrow 3$	$\uparrow\nwarrow 3$	$\nwarrow 3$
Y	5	$\uparrow 4$	$\uparrow\nwarrow 4$	$\uparrow\nwarrow 4$	$\uparrow\nwarrow 4$	$\nwarrow 3$

Todas as soluções ótimas possíveis:

- S N O W Y   S _ N O W Y   S _ N O W Y
- S U N N   Y   S U N _ N   Y   S U N   N _ Y

3. Algoritmo p/ Valor Ótimo (Bottom-up, não recursivo)

.	-	S	U	N	N	Y
-	0	1	2	3	4	5
S	1	↖0	←1	←2	←3	←4
N	2	↑1	↖1	↖1	←↖2	←3
O	3	↑2	↑↖2	↑↖2	↖2	←↖3
W	4	↑3	↑↖3	↑↖3	↑↖3	↖3
Y	5	↑4	↑↖4	↑↖4	↑↖4	↖3

Todas as soluções ótimas possíveis:

- S N O W Y   S _ N O W Y   S _ N O W Y
- S U N N Y   S U N _ N Y   S U N N _ Y

4. Algoritmo p/ obter uma Solução Ótima (Bottom-up)

Edit-Distance-PD(x, y, m, n)

Criar matriz $ed[0 \ldots m,\ 0 \ldots n]$
para $i \leftarrow 0$ **até** m: $\quad ed[i, 0] \leftarrow i; \quad R[i, 0] \leftarrow$ "$\uparrow$"
para $j \leftarrow 0$ **até** n: $\quad ed[0, j] \leftarrow j; \quad R[0, j] \leftarrow$ "$\leftarrow$"
para $i \leftarrow 1$ **até** m:
    para $j \leftarrow 1$ **até** n:
        se $(ed[i\text{-}1, j\text{-}1] + dif_{i,j} \leq 1 + \min\{ed[i, j\text{-}1], ed[i\text{-}1, j]\})$ **então**
            $ed[i, j] \leftarrow ed[i-1, j-1] + dif_{i,j}; \quad R[i, j] \leftarrow$ "$\nwarrow$"
        senão se $(ed[i, j-1] \leq ed[i-1, j])$ **então**
            $ed[i, j] \leftarrow ed[i, j-1] + 1; \quad R[i, j] \leftarrow$ "$\leftarrow$"
        senão
            $ed[i, j] \leftarrow ed[i-1, j] + 1; \quad R[i, j] \leftarrow$ "$\uparrow$"
retorne $ed[m, n]$ **e** R

Tempo $\Theta(m \cdot n)$

4b. Algoritmo p/ escrever a Solução Ótima (recursivo)

Print-Opt(x, y, i, j, R)

se $i = 0$ **e** $j = 0$ **então retorne**
se $R[i,j] =$ "$\nwarrow$" **então**
    Print-Opt$(x, y, i-1, j-1, R)$; **print** $\begin{bmatrix} x_i \\ y_j \end{bmatrix}$
se $R[i,j] =$ "$\leftarrow$" **então**
    Print-Opt$(x, y, i, j-1, R)$; **print** $\begin{bmatrix} - \\ y_j \end{bmatrix}$
se $R[i,j] =$ "$\uparrow$" **então**
    Print-Opt$(x, y, i-1, j, R)$; **print** $\begin{bmatrix} x_i \\ - \end{bmatrix}$

Tempo $\Theta(m+n)$

4b. Algoritmo p/ escrever a Solução Ótima (não-recursivo)

Print-OPT(x, y, m, n, R)

$Sol_1 \leftarrow \emptyset$; $Sol_2 \leftarrow \emptyset$; $k \leftarrow m+n$; $i \leftarrow m$; $j \leftarrow n$
enquanto $(i > 0)$ **ou** $(j > 0)$ **faça**
  se $R[i,j] =$ "$\nwarrow$" **então**
    $Sol_1[k] \leftarrow x_i$; $Sol_2[k] \leftarrow y_j$
    $k \leftarrow k-1$; $i \leftarrow i-1$; $j \leftarrow j-1$
  senão se $R[i,j] =$ "$\leftarrow$" **então**
    $Sol_1[k] \leftarrow$ "$\_$"; $Sol_2[k] \leftarrow y_j$
    $k \leftarrow k-1$; $j \leftarrow j-1$
  senão
    $Sol_1[k] \leftarrow x_i$; $Sol_2[k] \leftarrow$ "$\_$"
    $k \leftarrow k-1$; $i \leftarrow i-1$
print $Sol_1[k+1 \ldots m+n]$; **print** $Sol_2[k+1 \ldots m+n]$

Tempo $\Theta(m+n)$