```python
def solucion_dinamica(palabra1, palabra2, costos, acc = 0):
    n = len(palabra1)
    m = len(palabra2)

    # Inicialización de las tablas de costos y acciones
    dp = [[0 for _ in range(m + 1)] for _ in range(n + 1)]
    acciones = [["" for _ in range(m + 1)] for _ in range(n + 1)]

    # Inicializar primera fila y primera columna
    for i in range(n + 1):
        dp[i][0] = i * costos['borrar']
        acciones[i][0] = f"Borrar -> {costos['borrar']}" if i > 0 else ""

    for j in range(m + 1):
        dp[0][j] = j * costos['insertar']
        acciones[0][j] = f"Insertar -> {costos['insertar']}" if j > 0 else ""

    # Rellenar la tabla de dp y acciones
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            costo_borrar = dp[i - 1][j] + costos['borrar']
            costo_insertar = dp[i][j - 1] + costos['insertar']

            if palabra1[i - 1] == palabra2[j - 1]:
                costo_avanzar = dp[i - 1][j - 1] + costos['avanzar']
                accion_avanzar = f"Avanzar ( {palabra1[i - 1]} == {palabra2[j - 1]} ) -> {costos['avanzar']}"
            else:
                costo_avanzar = float('inf')  # No es una opción válida si los caracteres no coinciden
                accion_avanzar = ""

            costo_reemplazar = dp[i - 1][j - 1] + costos['reemplazar']
            accion_reemplazar = f"Reemplazar ( {palabra1[i - 1]} -> {palabra2[j - 1]} ) -> {costos['reemplazar']}"

            # Seleccionar la acción con el costo mínimo
            min_costo = min(costo_avanzar, costo_borrar, costo_insertar, costo_reemplazar)

            if min_costo == costo_avanzar:
                dp[i][j] = costo_avanzar
                acciones[i][j] = accion_avanzar
            elif min_costo == costo_borrar:
                dp[i][j] = costo_borrar
                acciones[i][j] = f"Borrar ( {palabra1[i - 1]} ) -> {costos['borrar']}"
            elif min_costo == costo_insertar:
                dp[i][j] = costo_insertar
                acciones[i][j] = f"Insertar ( {palabra2[j - 1]} ) -> {costos['insertar']}"
            else:
                dp[i][j] = costo_reemplazar
                acciones[i][j] = accion_reemplazar

    # Reconstrucción de la secuencia de acciones
    secuencia_acciones = []
    i, j = n, m
    while i > 0 or j > 0:
        accion = acciones[i][j]
        secuencia_acciones.append(accion)
        if "Borrar" in accion:
            i -= 1
        elif "Insertar" in accion:
            j -= 1
        else:
            i -= 1
            j -= 1

    secuencia_acciones.reverse()

    # Devolver resultados según la opción elegida (solo costo, solo acciones o ambos)
    if acc == 1:
        return secuencia_acciones
    elif acc == 2:
        return dp[n][m], secuencia_acciones
    return dp[n][m]
```

O(n+1) * O(m+1) { O(n x m)

O(m+1) O(n+1)

O(n + 1)

O(m+1)

O(n)

n x m1 → O(n x m)

O(n+m)

T(n,m) = O(n x m)

```python
def solucion_voraz(palabra1, palabra2, costos, acc=0):
    n = len(palabra1)
    m = len(palabra2)
    i, j = 0, 0
    costo_total = 0
    acciones = []

    # Mientras no hayamos recorrido completamente las palabras
    while i < n and j < m:
        if palabra1[i] == palabra2[j]:
            # Si los caracteres coinciden, elegimos entre avanzar o la opción más barata
            costo_avanzar = costos['avanzar']
            costo_reemplazar = costos['reemplazar']
            costo_borrar = costos['borrar']
            costo_insertar = costos['insertar']

            min_costo = min(costo_avanzar, costo_reemplazar, costo_borrar, costo_insertar)

            if min_costo == costo_avanzar:
                acciones.append(f"Avanzar ( {palabra1[i]} == {palabra2[j]} )")
                costo_total += costo_avanzar
                i += 1
                j += 1
            elif min_costo == costo_reemplazar:
                acciones.append(f"Reemplazar ( {palabra1[i]} -> {palabra2[j]} )")
                costo_total += costo_reemplazar
                i += 1
                j += 1
            elif min_costo == costo_borrar:
                acciones.append(f"Borrar ( {palabra1[i]} )")
                costo_total += costo_borrar
                i += 1
            else:
                acciones.append(f"Insertar ( {palabra2[j]} )")
                costo_total += costo_insertar
                j += 1
        else:
            # Si los caracteres no coinciden, consideramos reemplazar, borrar o insertar
            costo_reemplazar = costos['reemplazar']
            costo_borrar = costos['borrar']
            costo_insertar = costos['insertar']

            min_costo = min(costo_reemplazar, costo_borrar, costo_insertar)

            if min_costo == costo_reemplazar:
                acciones.append(f"Reemplazar ( {palabra1[i]} -> {palabra2[j]} )")
                costo_total += costo_reemplazar
                i += 1
                j += 1
            elif min_costo == costo_borrar:
                acciones.append(f"Borrar ( {palabra1[i]} )")
                costo_total += costo_borrar
                i += 1
            else:
                acciones.append(f"Insertar ( {palabra2[j]} )")
                costo_total += costo_insertar
                j += 1

    # Si quedan caracteres en palabra1, los borramos o destruimos
    while i < n:
        acciones.append(f"Borrar ( {palabra1[i]} )")
        costo_total += costos['borrar']
        i += 1

    # Si quedan caracteres en palabra2, los insertamos
    while j < m:
        acciones.append(f"Insertar ( {palabra2[j]} )")
        costo_total += costos['insertar']
        j += 1

    # Devolver resultados según acc
    if acc == 1:
        return acciones
    elif acc == 2:
        return costo_total, acciones
    return costo_total
```

$O(\min(n,m))$

$O(n)$

$O(m)$

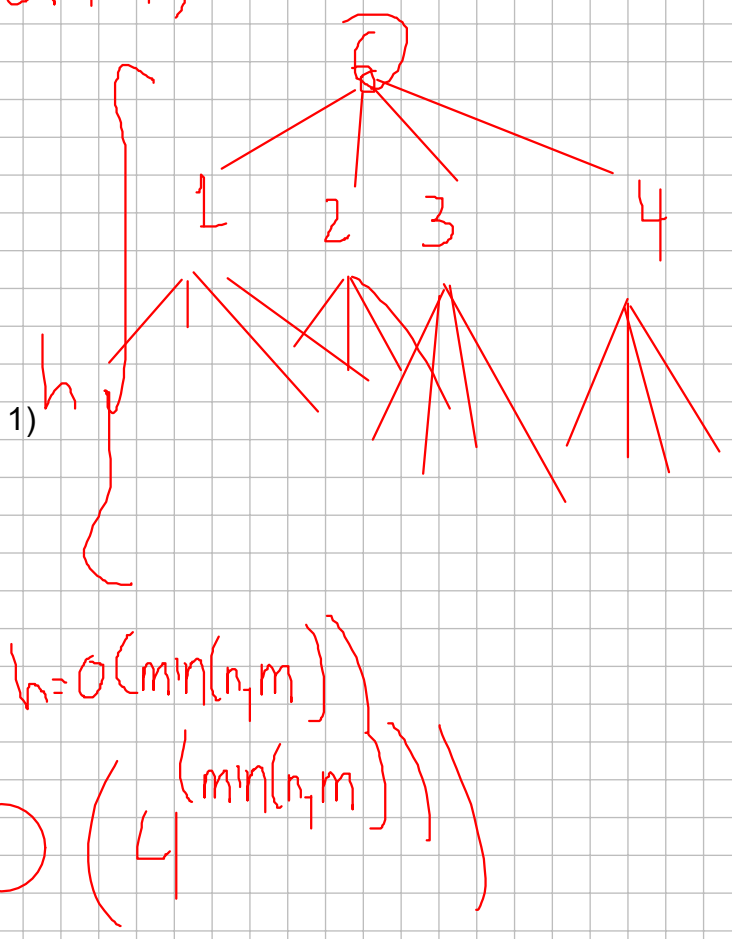
$O(n+m)$

$T(n,m) = O(n+m)$

```python
def fuerza_bruta(palabra1, palabra2, costos, i, j):
    if i == len(palabra1):
        return (len(palabra2) - j) * costos['insertar'], [
            "Insertar ({}) -> {}".format(palabra2[j], costos['insertar']) for j in range(j, len(palabra2))
        ]

    elif j == len(palabra2):
        return (len(palabra1) - i) * costos['borrar'], [
            "Borrar ({}) -> {}".format(palabra1[i], costos['borrar']) for i in range(i, len(palabra1))
        ]

    # Si los caracteres son iguales, no avanzamos automáticamente
    # sino que comparamos el costo de avanzar con el costo de otras operaciones
    costo_avanzar, acciones_avanzar = float('inf'), []
    if palabra1[i] == palabra2[j]:
        costo_avanzar, acciones_avanzar = fuerza_bruta(palabra1, palabra2, costos, i + 1, j + 1)
        costo_avanzar += costos['avanzar']

    # Opción de reemplazar
    costo_reemplazar, acciones_reemplazar = fuerza_bruta(palabra1, palabra2, costos, i + 1, j + 1)
    costo_reemplazar += costos['reemplazar']

    # Opción de insertar
    costo_insertar, acciones_insertar = fuerza_bruta(palabra1, palabra2, costos, i, j + 1)
    costo_insertar += costos['insertar']

    # Opción de borrar
    costo_borrar, acciones_borrar = fuerza_bruta(palabra1, palabra2, costos, i + 1, j)
    costo_borrar += costos['borrar']

    # Elegimos el menor costo entre avanzar, reemplazar, insertar y borrar
    min_costo = min(costo_avanzar, costo_reemplazar, costo_insertar, costo_borrar)

    if min_costo == costo_avanzar:
        return costo_avanzar, ["Avanzar ({} == {}) -> {}".format(palabra1[i], palabra2[j], costos['avanzar'])] + acciones_avanzar
    elif min_costo == costo_reemplazar:
        return costo_reemplazar, ["Reemplazar ({} -> {}) -> {}".format(palabra1[i], palabra2[j], costos['reemplazar'])] + acciones_reemplazar
    elif min_costo == costo_insertar:
        return costo_insertar, ["Insertar ({}) -> {}".format(palabra2[j], costos['insertar'])] + acciones_insertar
    else:
        return costo_borrar, ["Borrar ({}) -> {}".format(palabra1[i], costos['borrar'])] + acciones_borrar
```

O(m-j)

O(n-i)



$$T(n,m) = 4^{\min(n,m)}$$