```
import numpy as np

def edit_distance(s1, s2, ins_cost=1, del_cost=1, sub_cost=1):
    m, n = len(s1), len(s2)
    dp = np.zeros((m + 1, n + 1))

    # Initialize base cases
    for i in range(m + 1):
        dp[i][0] = i * del_cost
    for j in range(n + 1):
        dp[0][j] = j * ins_cost

    # Compute the DP table
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            cost_sub = 0 if s1[i - 1] == s2[j - 1] else sub_cost
            dp[i][j] = min(
                dp[i - 1][j] + del_cost,        # Deletion
                dp[i][j - 1] + ins_cost,        # Insertion
                dp[i - 1][j - 1] + cost_sub     # Substitution
            )

    # Backtrace to find the alignment
    alignment = []
    i, j = m, n
    while i > 0 or j > 0:
        if i > 0 and dp[i][j] == dp[i - 1][j] + del_cost:
            alignment.append((s1[i - 1], '-', 'Deletion'))
            i -= 1
        elif j > 0 and dp[i][j] == dp[i][j - 1] + ins_cost:
            alignment.append(('-', s2[j - 1], 'Insertion'))
            j -= 1
        else:
            alignment.append((s1[i - 1], s2[j - 1], 'Substitution' if s1[i - 1] != s2[j - 1] else 'Match'))
            i -= 1
            j -= 1

    alignment.reverse()
    return dp[m][n], alignment

# Example usage
s1 = "kitten"
s2 = "sitting"
distance, alignment = edit_distance(s1, s2, ins_cost=1, del_cost=1, sub_cost=2)
print(f"Minimum Edit Distance: {distance}")
print("Alignment:")
for step in alignment:
    print(step)
```

```
Minimum Edit Distance: 5.0
Alignment:
('-', 's', 'Insertion')
('k', '-', 'Deletion')
('i', 'i', 'Match')
('t', 't', 'Match')
('t', 't', 'Match')
('-', 'i', 'Insertion')
('e', '-', 'Deletion')
('n', 'n', 'Match')
('-', 'g', 'Insertion')
```