Algorithm 1 Backtrack a single alignment in a recursive way

```
1: S_1: Array(m), S_2: Array(n), M: Array(m + 1, n + 1),
2: function BacktrackRecurse(i, j)
3:     if i > 0 and j > 0 then
4:         substitute = M[i − 1][j − 1]
5:         delete = M[i − 1][j]
6:         insert = M[i][j − 1]
7:         min = min{substitute, delete, insert}
8:         if substitute = min then
9:             z = BacktrackRecurse(S_1, S_2, M, i − 1, j − 1)
10:            z = (S_1[i − 1]; S_2[j − 1]) ∘ z
11:        else if delete = min then
12:            z = BacktrackRecurse(S_1, S_2, M, i − 1, j)
13:            z = (S_1[i − 1]; ε) ∘ z
14:        else
15:            z = BacktrackRecurse(S_1, S_2, M, i, j − 1)
16:            z = (ε; S_2[j − 1]) ∘ z
17:        end if
18:    else if i > 0 then
19:        z = BacktrackRecurse(S_1, S_2, M, i − 1, j)
20:        z = (S_1[i − 1]; ε) ∘ z
21:    else if j > 0 then
22:        z = BacktrackRecurse(S_1, S_2, M, i, j − 1)
23:        z = (S_1[i − 1]; ε) ∘ z
24:    else
25:        return []
26:    end if
27:    return z
28: end function
29: function Backtrack(S_1: Array(m), S_2: Array(n), M: Array(m + 1, n + 1))
30:    return BacktrackRecurse(S_1, S_2, M, m, n)
31: end function
```

Algorithm 2 Backtrack all the optimum alignments in a recursive way

```
1: procedure RECURSE(S_1: Array(m), S_2: Array(n), M: Array(m+1, n+1), i, j)
2:     if i > 0 and j > 0 then
3:         substitute = M[i-1][j-1]
4:         delete = M[i-1][j]
5:         insert = M[i][j-1]
6:         min = min{substitute, delete, insert}
7:         if substitute = min then
8:             value = (S_1[i-1] / S_2[j-1])
9:             z' = value ∘ z
10:            BACKTRACKRECURSE(S_1, S_2, M, i-1, j-1, z')
11:        end if
12:        if delete = min then
13:            value = (S_1[i-1] / ε)
14:            z' = value ∘ z
15:            BACKTRACKRECURSE(S_1, S_2, M, i-1, j, z')
16:        end if
17:        if insert = min then
18:            value = (ε / S_2[j-1])
19:            z' = value ∘ z
20:            BACKTRACKRECURSE(S_1, S_2, M, i, j-1, z')
21:        end if
22:    else if i > 0 then
23:        value = (S_1[i-1] / ε)
24:        z' = value ∘ z
25:        BACKTRACKRECURSE(S_1, S_2, M, i-1, j, z')
26:    else if j > 0 then
27:        value = (ε / S_2[j-1])
28:        z' = value ∘ z
29:        BACKTRACKRECURSE(S_1, S_2, M, i, j-1, z')
30:    end if
31:    PRINT(z)
32: end procedure
33: procedure BACKTRACK(S_1: Array(m), S_2: Array(n), M: Array(m+1, n+1))
34:    return BACKTRACKRECURSE(S_1, S_2, M, m, n, [])
35: end procedure
```