Top19

a remplacer

Function Minimum(ByVal a As Integer, ByVal b As Integer, ByVal c As Integer) As Integer
If a <= b And a <= c Then
Minimum = a
ElseIf b <= a And b <= c Then
Minimum = b
Else
Minimum = c
End If
End Function

jacquard

```
Function JaccardDistance(ByVal s1 As String, ByVal s2 As String) As Double
    Dim set1 As Object
    Dim set2 As Object
    Dim common As Integer
    Dim total As Integer
    Dim char As String

    Set set1 = CreateObject("Scripting.Dictionary")
    Set set2 = CreateObject("Scripting.Dictionary")

    ' Ajouter chaque caractère de s1 dans set1
    For i = 1 To Len(s1)
        char = Mid(s1, i, 1)
        If Not set1.exists(char) Then
            set1.Add char, 1
        End If
    Next i

    ' Ajouter chaque caractère de s2 dans set2
    For i = 1 To Len(s2)
        char = Mid(s2, i, 1)
        If Not set2.exists(char) Then
            set2.Add char, 1
        End If
    Next i
```

```
    ' Calculer le nombre d'éléments communs
    For Each char In set1.keys
        If set2.exists(char) Then
            common = common + 1
        End If
    Next char

    ' Calculer le nombre total d'éléments
    total = set1.Count + set2.Count - common

    ' Calculer la distance de Jaccard
    If total = 0 Then
        JaccardDistance = 0
    Else
        JaccardDistance = common / total
    End If
End Function
```

Pour optimiser la rapidité du code, nous pouvons utiliser une technique appelée "mémoïsation". La mémoïsation consiste à stocker les résultats intermédiaires des calculs afin d'éviter de recalculer les mêmes valeurs à plusieurs reprises.

Dans le cas de l'algorithme de Levenshtein-Damerau, nous pouvons utiliser une matrice pour stocker les distances déjà calculées entre les préfixes des deux chaînes. Cela nous permettra de réduire considérablement le nombre de calculs répétés.

Voici comment nous pouvons mettre en œuvre la mémoïsation dans votre code :

```
Function Minimum(ByVal a As Integer, ByVal b As Integer, ByVal c As Integer)
As Integer
    If a <= b And a <= c Then
        Minimum = a
    ElseIf b <= a And b <= c Then
        Minimum = b
    Else
        Minimum = c
    End If
End Function

Function LevenshteinDamerauDistance(ByVal s1 As String, ByVal s2 As String)
As Integer
```

```
    Dim len1 As Integer
    Dim len2 As Integer
    Dim memo() As Integer
    Dim i As Integer
    Dim j As Integer
    Dim cost As Integer
    Dim costTransposition As Integer
    Dim char1 As String
    Dim char2 As String
    Dim bs1() As Byte
    Dim bs2() As Byte

    len1 = Len(s1)
    len2 = Len(s2)

    ReDim memo(0 To len1, 0 To len2)

    ' Convertir les chaînes en tableaux de bytes
    bs1 = StrConv(s1, vbFromUnicode)
    bs2 = StrConv(s2, vbFromUnicode)

    ' Initialise la mémoïsation
    For i = 0 To len1
        For j = 0 To len2
            memo(i, j) = -1 ' Marquer comme non calculé
        Next j
    Next i

    ' Appel récursif de la fonction de calcul avec mémoïsation
    LevenshteinDamerauDistance = ComputeDistanceWithMemo(len1, len2, bs1,
bs2, memo)
End Function

Function ComputeDistanceWithMemo(ByVal len1 As Integer, ByVal len2 As
Integer, ByVal bs1() As Byte, ByVal bs2() As Byte, ByRef memo() As Integer)
As Integer
    If len1 = 0 Then
        ComputeDistanceWithMemo = len2
    ElseIf len2 = 0 Then
        ComputeDistanceWithMemo = len1
    ElseIf memo(len1, len2) <> -1 Then
        ComputeDistanceWithMemo = memo(len1, len2)
    Else
```

```
        Dim cost As Integer
        Dim costTransposition As Integer
        Dim char1 As String
        Dim char2 As String

        char1 = Chr(bs1(len1 - 1))
        char2 = Chr(bs2(len2 - 1))

        If char1 = char2 Then
            cost = 0
        Else
            cost = 1
        End If

        costTransposition = 1 ' Coût de transposition par défaut

        ' Si transposition possible, calculer le coût
        If len1 > 1 And len2 > 1 Then
            If bs1(len1 - 1) = bs2(len2 - 2) And bs1(len1 - 2) = bs2(len2 -
1) Then
                costTransposition = 1 ' Coût de transposition
            End If
        End If

        ' Calculer le minimum des trois coûts
        memo(len1, len2) = Minimum(ComputeDistanceWithMemo(len1 - 1, len2,
bs1, bs2, memo) + 1, _
                                   ComputeDistanceWithMemo(len1, len2 - 1,
bs1, bs2, memo) + 1, _
                                   ComputeDistanceWithMemo(len1 - 1, len2 -
1, bs1, bs2, memo) + cost)

        ' Vérifier si la transposition est moins coûteuse
        If len1 > 1 And len2 > 1 Then
            memo(len1, len2) = Minimum(memo(len1, len2),
ComputeDistanceWithMemo(len1 - 2, len2 - 2, bs1, bs2, memo) +
costTransposition)
        End If

        ComputeDistanceWithMemo = memo(len1, len2)
    End If
End Function
```

Dans cette version, nous utilisons la matrice memo pour stocker les résultats intermédiaires des calculs. Avant de calculer une distance entre deux préfixes, nous vérifions d'abord si la distance a déjà été calculée et stockée dans memo. Si c'est le cas, nous utilisons la valeur stockée dans memo plutôt que de recalculer la distance. Cela réduit considérablement le nombre de calculs répétés et améliore les performances de la fonction.

Levenshtein

```
Function Minimum(ByVal a As Integer, ByVal b As Integer, ByVal c As Integer)
As Integer
    If a <= b And a <= c Then
        Minimum = a
    ElseIf b <= a And b <= c Then
        Minimum = b
    Else
        Minimum = c
    End If
End Function

Function LevenshteinDistance(ByVal s1 As String, ByVal s2 As String) As
Integer
    Dim len1 As Integer
    Dim len2 As Integer
    Dim memo() As Integer
    Dim i As Integer
    Dim j As Integer
    Dim cost As Integer
    Dim char1 As String
    Dim char2 As String

    len1 = Len(s1)
    len2 = Len(s2)

    ReDim memo(0 To len1, 0 To len2)

    ' Initialise la mémoïsation
    For i = 0 To len1
        For j = 0 To len2
            memo(i, j) = -1 ' Marquer comme non calculé
        Next j
    Next i

    ' Appel récursif de la fonction de calcul avec mémoïsation
```

```
    LevenshteinDistance = ComputeDistanceWithMemo(len1, len2, s1, s2, memo)
End Function

Function ComputeDistanceWithMemo(ByVal len1 As Integer, ByVal len2 As
Integer, ByVal s1 As String, ByVal s2 As String, ByRef memo() As Integer) As
Integer
    If len1 = 0 Then
        ComputeDistanceWithMemo = len2
    ElseIf len2 = 0 Then
        ComputeDistanceWithMemo = len1
    ElseIf memo(len1, len2) <> -1 Then
        ComputeDistanceWithMemo = memo(len1, len2)
    Else
        Dim cost As Integer
        Dim char1 As String
        Dim char2 As String

        char1 = Mid(s1, len1, 1)
        char2 = Mid(s2, len2, 1)

        If char1 = char2 Then
            cost = 0
        Else
            cost = 1
        End If

        ' Calculer le minimum des trois coûts
        memo(len1, len2) = Minimum(ComputeDistanceWithMemo(len1 - 1, len2,
s1, s2, memo) + 1, _
                                   ComputeDistanceWithMemo(len1, len2 - 1,
s1, s2, memo) + 1, _
                                   ComputeDistanceWithMemo(len1 - 1, len2 -
1, s1, s2, memo) + cost)

        ComputeDistanceWithMemo = memo(len1, len2)
    End If
End Function
```

JaroWinkler

Pour implémenter la mémoïsation dans la fonction Jaro-Winkler, nous devons prendre en compte la comparaison des caractères entre les deux chaînes. Nous pouvons stocker les résultats intermédiaires

des comparaisons de caractères dans une matrice de mémoïsation pour éviter de les recalculer à chaque fois. Voici comment nous pouvons le faire :

```
Function JaroWinklerDistance(ByVal s1 As String, ByVal s2 As String) As
Double
    Dim len1 As Integer
    Dim len2 As Integer
    Dim memo() As Double
    Dim i As Integer
    Dim j As Integer
    Dim matchWindow As Integer
    Dim char1 As String
    Dim char2 As String

    len1 = Len(s1)
    len2 = Len(s2)

    ReDim memo(0 To len1, 0 To len2)

    ' Initialise la mémoïsation
    For i = 0 To len1
        For j = 0 To len2
            memo(i, j) = -1 ' Marquer comme non calculé
        Next j
    Next i

    ' Appel récursif de la fonction de calcul avec mémoïsation
    JaroWinklerDistance = ComputeDistanceWithMemo(len1, len2, s1, s2, memo)
End Function

Function ComputeDistanceWithMemo(ByVal len1 As Integer, ByVal len2 As
Integer, ByVal s1 As String, ByVal s2 As String, ByRef memo() As Double) As
Double
    If len1 = 0 Or len2 = 0 Then
        ComputeDistanceWithMemo = 0
    ElseIf memo(len1, len2) <> -1 Then
        ComputeDistanceWithMemo = memo(len1, len2)
    Else
        Dim matchCount As Integer
        Dim transpositionCount As Integer
        Dim prefixLength As Integer
        Dim scalingFactor As Double
        Dim char1 As String
```

```
        Dim char2 As String

        matchWindow = WorksheetFunction.Max(len1, len2) \ 2 - 1

        For i = 1 To len1
            For j = WorksheetFunction.Max(1, i - matchWindow) To
WorksheetFunction.Min(len2, i + matchWindow)
                If memo(i, j) = -1 Then
                    char1 = Mid(s1, i, 1)
                    char2 = Mid(s2, j, 1)

                    If char1 = char2 Then
                        matchCount = matchCount + 1
                        memo(i, j) = 1
                    End If
                End If
            Next j
        Next i

        transpositionCount = ComputeTranspositionCount(s1, s2, memo)

        prefixLength = ComputePrefixLength(s1, s2)

        scalingFactor = 0.1

        ComputeDistanceWithMemo = JaroDistance(len1, len2, matchCount,
transpositionCount) + (prefixLength * scalingFactor * (1 -
JaroDistance(len1, len2, matchCount, transpositionCount)))

        memo(len1, len2) = ComputeDistanceWithMemo
    End If
End Function

Function ComputeTranspositionCount(ByVal s1 As String, ByVal s2 As String,
ByRef memo() As Double) As Integer
    Dim len1 As Integer
    Dim len2 As Integer
    Dim transpositionCount As Integer
    Dim i As Integer
    Dim j As Integer

    len1 = Len(s1)
    len2 = Len(s2)
```

```
    For i = 1 To len1
        For j = 1 To len2
            If memo(i, j) = 1 Then
                If Mid(s1, i, 1) <> Mid(s2, j, 1) Then
                    transpositionCount = transpositionCount + 1
                End If
            End If
        Next j
    Next i

    ComputeTranspositionCount = transpositionCount \ 2
End Function

Function ComputePrefixLength(ByVal s1 As String, ByVal s2 As String) As
Integer
    Dim prefixLength As Integer
    Dim maxLength As Integer
    Dim i As Integer

    maxLength = WorksheetFunction.Min(Len(s1), Len(s2))

    For i = 1 To maxLength
        If Mid(s1, i, 1) = Mid(s2, i, 1) Then
            prefixLength = prefixLength + 1
        Else
            Exit For
        End If
    Next i

    ComputePrefixLength = prefixLength
End Function

Function JaroDistance(ByVal len1 As Integer, ByVal len2 As Integer, ByVal
matchCount As Integer, ByVal transpositionCount As Integer) As Double
    Dim m As Integer
    Dim t As Double

    m = matchCount
    t = transpositionCount

    If m = 0 Then
        JaroDistance = 0
```

```
    Else
        JaroDistance = (m / len1 + m / len2 + (m - t) / m) / 3
    End If
```

ALL

Function ComputeDistances(ByVal s1 As String, ByVal s2 As String) As Variant
Dim len1 As Integer
Dim len2 As Integer
Dim memoLevenshtein() As Integer
Dim memoLevenshteinDamerau() As Integer
Dim scoreJaroWinkler As Double
Dim i As Integer
Dim j As Integer
Dim cost As Integer
Dim costTransposition As Integer
Dim char1 As String
Dim char2 As String
Dim bs1() As Byte
Dim bs2() As Byte

```
len1 = Len(s1)
len2 = Len(s2)

ReDim memoLevenshtein(0 To len1, 0 To len2)
ReDim memoLevenshteinDamerau(0 To len1, 0 To len2)

' Convertir les chaînes en tableaux de bytes
bs1 = StrConv(s1, vbFromUnicode)
bs2 = StrConv(s2, vbFromUnicode)

' Calculer les distances de Levenshtein et Levenshtein-Damerau avec
mémoïsation
ComputeLevenshteinAndDamerauDistances len1, len2, bs1, bs2, memoLevenshtein,
memoLevenshteinDamerau

' Calculer le score de Jaro-Winkler
scoreJaroWinkler = JaroWinklerDistance(s1, s2)

' Retourner les résultats des trois distances
ComputeDistances = Array(memoLevenshtein(len1, len2),
memoLevenshteinDamerau(len1, len2), scoreJaroWinkler)
```

End Function

```
Sub ComputeLevenshteinAndDamerauDistances(ByVal len1 As Integer, ByVal len2 As Integer, ByVal bs1() As Byte, ByVal bs2() As Byte, ByRef memoLevenshtein() As Integer, ByRef memoLevenshteinDamerau() As Integer)
Dim i As Integer
Dim j As Integer
Dim cost As Integer
Dim costTransposition As Integer
Dim char1 As String
Dim char2 As String

' Initialise la mémoïsation pour Levenshtein et Levenshtein-Damerau
For i = 0 To len1
    memoLevenshtein(i, 0) = i
    memoLevenshteinDamerau(i, 0) = i
Next i
For j = 0 To len2
    memoLevenshtein(0, j) = j
    memoLevenshteinDamerau(0, j) = j
Next j

For i = 1 To len1
    For j = 1 To len2
        char1 = Chr(bs1(i - 1))
        char2 = Chr(bs2(j - 1))

        If char1 = char2 Then
            cost = 0
        Else
            cost = 1
        End If

        costTransposition = 1 ' Coût de transposition par défaut

        ' Si transposition possible, calculer le coût
        If i > 1 And j > 1 Then
            If bs1(i - 1) = bs2(j - 2) And bs1(i - 2) = bs2(j - 1) Then
                costTransposition = 1 ' Coût de transposition
            End If
        End If

        ' Calculer les distances de Levenshtein et Levenshtein-Damerau avec mémoïsation
        memoLevenshtein(i, j) =
```

```
WorksheetFunction.Min(WorksheetFunction.Min(memoLevenshtein(i - 1, j) + 1,
memoLevenshtein(i, j - 1) + 1), memoLevenshtein(i - 1, j - 1) + cost)

        memoLevenshteinDamerau(i, j) =
WorksheetFunction.Min(WorksheetFunction.Min(memoLevenshteinDamerau(i - 1, j)
+ 1, memoLevenshteinDamerau(i, j - 1) + 1), memoLevenshteinDamerau(i - 1, j
- 1) + cost)

        ' Vérifier si la transposition est moins coûteuse pour Levenshtein-
Damerau
        If i > 1 And j > 1 Then
            memoLevenshteinDamerau(i, j) =
WorksheetFunction.Min(memoLevenshteinDamerau(i, j), memoLevenshteinDamerau(i
- 2, j - 2) + costTransposition)
        End If
    Next j
Next i

End Sub
```