Minimum ASCII Delete Sum for Two Strings

Given two strings `s1, s2`, find the lowest ASCII sum of deleted characters to make two strings equal.

Example 1:

```
Input: s1 = "sea", s2 = "eat"
Output: 231
Explanation: Deleting "s" from "sea" adds the ASCII value of "s" (115) to the sum.
Deleting "t" from "eat" adds 116 to the sum.
At the end, both strings are equal, and 115 + 116 = 231 is the minimum sum possible t
o achieve this.
```

Example 2:

```
Input: s1 = "delete", s2 = "leet"
Output: 403
Explanation: Deleting "dee" from "delete" to turn the string into "let",
adds 100[d]+101[e]+101[e] to the sum.  Deleting "e" from "leet" adds 101[e] to the su
m.
At the end, both strings are equal to "let", and the answer is 100+101+101+101 = 403.
If instead we turned both strings into "lee" or "eet", we would get answers of 433 or
 417, which are higher.
```

Note:

- `0 .`
- All elements of each string will have an ASCII value in `[97, 122]`.

Solution 1

The same idea as edit distance. Straightforward 19 lines.

```
class Solution {
    public int minimumDeleteSum(String s1, String s2) {
        int[][] count = new int[s1.length() + 1][s2.length() + 1];
        for(int i = 1; i < count.length; i++){
            count[i][0] = count[i-1][0] + s1.charAt(i-1);
        }
        for(int i = 1; i < count[0].length; i++){
            count[0][i] = count[0][i-1] + s2.charAt(i-1);
        }
        for(int i = 1; i < count.length; i++){
            for(int j = 1; j < count[0].length; j++){
                int cost = (s1.charAt(i-1) == s2.charAt(j-1))? 0 : s1.charAt(i-1) +
s2.charAt(j-1);
                count[i][j] = Math.min(count[i-1][j] + s1.charAt(i-1), count[i][j-1
] + s2.charAt(j-1));
                count[i][j] = Math.min(count[i][j], count[i-1][j-1] + cost);
            }
        }
        return count[s1.length()][s2.length()];
    }
}
``
```

written by DonaldTrump original link here

Solution 2

This is clearly a DP problem.

```
dp[i][j] is the cost for s1.substr(0,i) and s2.substr(0, j). Note s1[i], s2[j] not i
ncluded in the substring.

Base case: dp[0][0] = 0
target: dp[m][n]

if s1[i-1] = s2[j-1]   // no deletion
    dp[i][j] = dp[i-1][j-1];
else   // delete either s1[i-1] or s2[j-1]
    dp[i][j] = min(dp[i-1][j]+s1[i-1], dp[i][j-1]+s2[j-1]);
```

We can use a 2D vector, or an optimized O(n) extra space. See below. The run time is O(mn).

```cpp
class Solution {
public:
    int minimumDeleteSum(string s1, string s2) {
        int m = s1.size(), n = s2.size();
        vector<vector<int>> dp(m+1, vector<int>(n+1, 0));
        for (int j = 1; j <= n; j++)
            dp[0][j] = dp[0][j-1]+s2[j-1];
        for (int i = 1; i <= m; i++) {
            dp[i][0] = dp[i-1][0]+s1[i-1];
            for (int j = 1; j <= n; j++) {
                if (s1[i-1] == s2[j-1])
                    dp[i][j] = dp[i-1][j-1];
                else
                    dp[i][j] = min(dp[i-1][j]+s1[i-1], dp[i][j-1]+s2[j-1]);
            }
        }
        return dp[m][n];
    }
};
```

Optimized O(n) extra space

```
class Solution {
public:
    int minimumDeleteSum(string s1, string s2) {
        int m = s1.size(), n = s2.size();
        vector<int> dp(n+1, 0);
        for (int j = 1; j <= n; j++)
            dp[j] = dp[j-1]+s2[j-1];
        for (int i = 1; i <= m; i++) {
            int t1 = dp[0];
            dp[0] += s1[i-1];
            for (int j = 1; j <= n; j++) {
                int t2 = dp[j];
                dp[j] = s1[i-1] == s2[j-1]? t1:min(dp[j]+s1[i-1], dp[j-1]+s2[j-1]);
                t1 = t2;
            }
        }
        return dp[n];
    }
};
```

written by zestypanda original link here

Solution 3

DP Formula

```
/**
 * dp[i][j] = a[i] == b[j] ? dp[i + 1][j + 1] :
 *            min(a[i] + dp[i + 1][j],  // delete a[i] + minimumDeleteSum(a.substr(
i+1), b.substr(j))
 *                b[j] + dp[i][j + 1])  // delete b[j] + minimumDeleteSum(a.substr(
i), b.substr(j+1))
 */
```

Java

```java
class Solution {
    public int minimumDeleteSum(String s1, String s2) {
        int m = s1.length(), n = s2.length(), MAX = Integer.MAX_VALUE;
        char[] a = s1.toCharArray(), b = s2.toCharArray();
        int[][] dp = new int[m + 1][n + 1];
        for (int i = m; i >= 0; i--) {
            for (int j = n; j >= 0; j--) {
                if (i < m || j < n)
                    dp[i][j] = i < m && j < n && a[i] == b[j] ?
                        dp[i + 1][j + 1] : Math.min((i < m ? a[i] + dp[i + 1][j] :
MAX), (j < n ? b[j] + dp[i][j + 1] : MAX));
            }
        }
        return dp[0][0];
    }
}
```

C++

```cpp
class Solution {
public:
    int minimumDeleteSum(string a, string b) {
        int m = a.size(), n = b.size();
        vector<vector<int>> dp(m + 1, vector<int>(n + 1, 0));
        for (int i = m; i >= 0; i--) {
            for (int j = n; j >= 0; j--) {
                if (i < m || j < n)
                    dp[i][j] = i < m && j < n && a[i] == b[j] ?
                        dp[i + 1][j + 1] : min((i < m ? a[i] + dp[i + 1][j] : INT_M
AX), (j < n ? b[j] + dp[i][j + 1] : INT_MAX));
            }
        }
        return dp[0][0];
    }
};
```

written by alexander original link here