Minimum Window Subsequence

Given strings `S` and `T`, find the minimum (contiguous) **substring** `W` of `S`, so that `T` is a **subsequence** of `W`.

If there is no such window in `S` that covers all characters in `T`, return the empty string `""`. If there are multiple such minimum-length windows, return the one with the left-most starting index.

Example 1:

```
Input: 
S = "abcdebdde", T = "bde"
Output: "bcde"
Explanation: 
"bcde" is the answer because it occurs before "bdde" which has the same length.
"deb" is not a smaller window because the elements of T in the window must occur in o
rder.
```

Note:

- All the strings in the input will only contain lowercase letters.
- The length of `S` will be in the range `[1, 20000]`.
- The length of `T` will be in the range `[1, 100]`.

Solution 1

```
For substring S[0, i] and T[0, j],
dp[i][j] is starting index k of the shortest postfix of S[0, i], such that T[0, j] i
s a subsequence of S[k, i].
Here T[0] = S[k], T[j] = S[i]. Otherwise, dp[i][j] = -1.

The goal is the substring with length of min(i-dp[i][n-1]) for all i < m,  where m i
s S.size() and n is T.size()
Initial condition: dp[i][0] = i if S[i] = T[0], else -1
Equations: If S[i] = T[j], dp[i][j] = max(dp[k][j-1]) for all k < i; else dp[i][j]
= -1;
```

O(mn) space 82 ms

```cpp
class Solution {
public:
    string minWindow(string S, string T) {
        int m = S.size(), n = T.size();
        vector<vector<int>> dp(n, vector<int>(m, -1));
        for (int i = 0; i < m; i++)
            if (S[i] == T[0]) dp[0][i] = i;
        for (int j = 1; j < n; j++) {
            int k = -1;
            for (int i = 0; i < m; i++) {
                if (k != -1 && S[i] == T[j]) dp[j][i] = k;
                if (dp[j-1][i] != -1) k = dp[j-1][i];
            }
        }
        int st = -1, len = INT_MAX;
        for (int i = 0; i < m; i++) {
            if (dp[n-1][i] != -1 && i-dp[n-1][i]+1 < len) {
                st = dp[n-1][i];
                len = i-dp[n-1][i]+1;
            }
        }
        return st == -1? "":S.substr(st, len);
    }
};
```

O(m) space 53 ms

```
class Solution {
public:
    string minWindow(string S, string T) {
        int m = S.size(), n = T.size();
        vector<int> dp(m, -1);
        for (int i = 0; i < m; i++)
            if (S[i] == T[0]) dp[i] = i;
        for (int j = 1; j < n; j++) {
            int k = -1;
            vector<int> tmp(m, -1);
            for (int i = 0; i < m; i++) {
                if (k != -1 && S[i] == T[j]) tmp[i] = k;
                if (dp[i] != -1) k = dp[i];
            }
            swap(dp, tmp);
        }
        int st = -1, len = INT_MAX;
        for (int i = 0; i < m; i++) {
            if (dp[i] != -1 && i-dp[i]+1 < len) {
                st = dp[i];
                len = i-dp[i]+1;
            }
        }
        return st == -1? "":S.substr(st, len);
    }
};
```

written by zestypanda original link here

Solution 2

```
class Solution {
public:
    string minWindow(string s, string t) {
        int ns = s.size(), nt= t.size();
        int dp[ns+1][nt+1] = {};
        const int mxx = ns + 1;
        for (int i = 0 ; i <= ns; ++i) {
            for (int j = 1; j <= nt; ++j) {
                dp[i][j] = mxx;
                if (i) {
                    dp[i][j] = min(dp[i][j], 1 + dp[i-1][j]);
                    if (s[i-1] == t[j-1]) dp[i][j]  = min(dp[i][j], 1 + dp[i-1][j-1
]);
                }
            }
        }

        int ans = ns + 1, x = -1;
        for (int i = 0; i <=ns; ++i) if (dp[i][nt] < ans) {
            x = i;
            ans = dp[i][nt];
        }

        if (x < 0) return "";
        return s.substr(x-ans,ans);
    }
};
```

written by elastico original link here

Solution 3

```
class Solution {
    public String minWindow(String S, String T) {
        String output = "";
        int minLen = 20001;
        for (int i = 0; i <= S.length() - T.length(); i++) {
         while (i < S.length() && S.charAt(i) != T.charAt(0)) {
          i++;
         }
         int l = find(S.substring(i, Math.min(i + minLen, S.length())), T);
         if (l != -1 && l < minLen) {
          minLen = l;
          output = S.substring(i, i + l);
         }
        }
        return output;
    }

 private int find(String S, String T) {
  for (int i = 0, j = 0; i < S.length() && j < T.length();) {
   if (S.charAt(i) == T.charAt(j)) {
    i++;
    j++;
    if (j == T.length()) {
     return i;
    }
   } else {
    i++;
   }
  }
  return -1;
 }
}
```

written by 2499370956 original link here