727. Minimum Window Subsequence <hard>

class Solution {

public:

string minWindow(string S, string T) {

/\*

int N = S.length();

vector<int> last(26, -1);

vector<vector<int>> nxt(N, vector<int>(26));

for (int i = N - 1; i >= 0; --i) {

last[S[i] - 'a'] = i;

for (int k = 0; k < 26; ++k) {

nxt[i][k] = last[k];

}

}

vector<vector<int>> windows;

for (int i = 0; i < N; ++i) {

if (S[i] == T[0])

windows.push\_back({i, i});

}

for (int j = 1; j < T.length(); ++j) {

int letterIndex = T[j] - 'a';

for (vector<int>& window: windows) {

if (window[1] < N-1 && nxt[window[1]+1][letterIndex] >= 0) {

window[1] = nxt[window[1]+1][letterIndex];

}

else {

window[0] = window[1] = -1;

break;

}

}

}

vector<int> ans = {-1, (int) S.length()};

for (vector<int>& window: windows) {

if (window[0] == -1) break;

if (window[1] - window[0] < ans[1] - ans[0]) {

ans = window;

}

}

return ans[0] >= 0 ? S.substr(ans[0], ans[1] + 1 - ans[0]) : "";

\*/

// two pointer

int N = S.size();

int M = T.size();

int start = -1;

int minLen = INT\_MAX;

int i = 0, j = 0;

// int startpos = 0;

while (i < N) {

if (S[i] == T[j]) {

// if(j == 0)

// startpos = i;

j++;

if (j == M) {

int end = i;

#if 1

while (j > 0) {

j--;

while (S[i] != T[j]) {

i--;

}

i--;

}

//++i;

i++;

#else

i = startpos;

j = 0;

#endif

// cout << i << ", " << j << endl;

if (end - i + 1 < minLen) {

minLen = end - i + 1;

start = i;

}

}

}

++i;

}

return (start != -1) ? S.substr(start, minLen) : "";

}

};

/\*

// https://leetcode.com/problems/minimum-window-subsequence/discuss/109356/JAVA-two-pointer-solution-(12ms-beat-100)-with-explaination

// Comment by caraxin

// 1. Normal order, find each of the first position of T[i] in S. i.

// 2. From there, in reverse order, find the start position in S. j.

// 3. If the new length i-j+1 is smaller, then update it.

class Solution {

public:

string minWindow(string S, string T) {

string res;

int i=-1, j=0, n=S.size();

while(i<n){

for(const auto& c:T){

i = S.find\_first\_of(c,i+1); // Search for positions at or after i+1.

if(i==string::npos)

return res;

}

j=i+1;

for(int k=T.size()-1; k>=0; --k)

j = S.find\_last\_of(T[k],j-1); // Search for positions at or before j-1;

if(res.empty() || res.size()>i-j+1)

res=S.substr(j,i-j+1);

if(res.size()==T.size())

return res;

i=j; // Start from the next element i S.

}

return res;

}

};

\*/