/// <summary>

/// Leet code #9900. Count number of substrings with exactly k distinct

/// characters

///

/// Given a string of lowercase alphabets, count all possible substrings

/// (not necessarily distinct) that has exactly k distinct characters.

///

/// Examples:

///

/// Input: abc, k = 2

/// Output: 2

/// Possible substrings are {"ab", "bc"}

///

/// Input: aba, k = 2

/// Output: 3

/// Possible substrings are {"ab", "ba", "aba"}

///

/// Input: aa, k = 1

/// Output: 3

/// Possible substrings are {"a", "a", "aa"}

/// </summary>

int LeetCode::countkDist(string str, int k)

{

// start of the string with k characters

int first = 0;

// end of the string with k characters

int last = 0;

// end position where the substring first seen with k characters

int end = -1;

int result = 0;

// protect k == 0;

if (k == 0) return result;

// remember all position of specific character

unordered\_map<char, deque<int>> char\_map;

while (last <= (int)str.size())

{

if (char\_map.size() == k)

{

if (last == str.size() || char\_map.count(str[last]) == 0)

{

// shrink and count

while (char\_map.size() == k)

{

result += last - end;

char\_map[str[first]].pop\_front();

if (char\_map[str[first]].empty())

{

char\_map.erase(str[first]);

}

else

{

end = max(char\_map[str[first]].front(), end);

}

first++;

}

end = -1;

}

}

if (last < (int)str.size())

{

char\_map[str[last]].push\_back(last);

// first time hit k distinct

if (char\_map.size() == k && end == -1) end = last;

}

last++;

}

return result;

}

/// <summary>

/// Leet code #907. Sum of Subarray Minimums

///

/// Given an array of integers A, find the sum of min(B), where B ranges over

/// every (contiguous) subarray of A.

///

/// Since the answer may be large, return the answer modulo 10^9 + 7.

///

/// Example 1:

///

/// Input: [3,1,2,4]

/// Output: 17

/// Explanation: Subarrays are [3], [1], [2], [4], [3,1], [1,2], [2,4],

/// [3,1,2], [1,2,4], [3,1,2,4].

/// Minimums are 3, 1, 2, 4, 1, 1, 2, 1, 1, 1. Sum is 17.

///

/// Note:

///

/// 1. 1 <= A.length <= 30000

/// 2. 1 <= A[i] <= 30000

/// </summary>

int LeetCode::sumSubarrayMins(vector<int>& A)

{

// A array of pair, dp[i][0] is the position in A, dp[i][1] is the value of

// and dp[i][2] is the partial sum

size\_t M = 1000000007;

vector<vector<int>> dp;

dp.push\_back({ -1, 0, 0 });

int result = 0;

for (int i = 0; i < (int)A.size(); i++)

{

// kick out all the values greater than current

while (dp.back()[1] > A[i]) dp.pop\_back();

// starting from one position after previous smaller element to current

// all subarry choose current value

int sum = A[i] \* (i - dp.back()[0]);

sum %= M;

// for any subarry ending current but including previous smaller number

// and the more smaller number before previous...

sum += dp.back()[2];

sum %= M;

result += sum;

result %= M;

dp.push\_back({ i, A[i], sum });

}

return result;

}