```cpp
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

#include <set>
#include <unordered_map>
#include <algorithm>
#include <climits>
#include <vector>
using namespace std;

int getMinSpaces(string pi, set<string> numbersTable,
                 unordered_map<int, int> *cache, int idx);

// O(n^3 + m) time | O(n + m) space - where n is the number of digits in Pi and
// m is the number of favorite numbers
int numbersInPi(string pi, vector<string> numbers) {
  set<string> numbersTable;
  for (string number : numbers) {
    numbersTable.insert(number);
  }
  unordered_map<int, int> cache;
  for (int i = pi.length() - 1; i >= 0; i--) {
    getMinSpaces(pi, numbersTable, &cache, i);
  }
  return cache.at(0) == INT_MAX ? -1 : cache.at(0);
}

int getMinSpaces(string pi, set<string> numbersTable,
                 unordered_map<int, int> *cache, int idx) {
  if (idx == pi.length())
    return -1;
  if (cache->find(idx) != cache->end())
    return cache->at(idx);
  int minSpaces = INT_MAX;
  for (int i = idx; i < pi.length(); i++) {
    string prefix = pi.substr(idx, i + 1 - idx);
    if (numbersTable.find(prefix) != numbersTable.end()) {
      int minSpacesInSuffix = getMinSpaces(pi, numbersTable, cache, i + 1);
      // Handle int overflow.
      if (minSpacesInSuffix == INT_MAX) {
        minSpaces = min(minSpaces, minSpacesInSuffix);
      } else {
        minSpaces = min(minSpaces, minSpacesInSuffix + 1);
      }
    }
  }
  cache->insert({idx, minSpaces});
  return cache->at(idx);
}
```