Given an input string and a dictionary of words, find out if the input string can be segmented into a space-separated sequence of dictionary words. See following examples for more details.
This is a famous Google interview question, also being asked by many other companies now a days.

```
Consider the following dictionary
{ i, like, sam, sung, samsung, mobile, ice,
  cream, icecream, man, go, mango}

Input:  ilike
Output: Yes
The string can be segmented as "i like".

Input:  ilikesamsung
Output: Yes
The string can be segmented as "i like samsung" or "i like sam sung".
```

Recursive implementation:
The idea is simple, we consider each prefix and search it in dictionary. If the prefix is present in dictionary, we recur for rest of the string (or suffix). If the recursive call for suffix returns true, we return true, otherwise we try next prefix. If we have tried all prefixes and none of them resulted in a solution, we return false.

We strongly recommend to see **substr** function which is used extensively in following implementations.

```
// A recursive program to test whether a given string can be segmented into
// space separated words in dictionary
#include <iostream>
using namespace std;

/* A utility function to check whether a word is present in dictionary or not.
  An array of strings is used for dictionary.  Using array of strings for
  dictionary is definitely not a good idea. We have used for simplicity of
  the program*/
int dictionaryContains(string word)
{
    string dictionary[] = {"mobile","samsung","sam","sung","man","mango",
                           "icecream","and","go","i","like","ice","cream"};
    int size = sizeof(dictionary)/sizeof(dictionary[0]);
    for (int i = 0; i < size; i++)
        if (dictionary[i].compare(word) == 0)
           return true;
    return false;
}

// returns true if string can be segmented into space separated
// words, otherwise returns false
bool wordBreak(string str)
{
    int size = str.size();

    // Base case
    if (size == 0)  return true;

    // Try all prefixes of lengths from 1 to size
    for (int i=1; i<=size; i++)
    {
        // The parameter for dictionaryContains is str.substr(0, i)
        // str.substr(0, i) which is prefix (of input string) of
        // length 'i'. We first check whether current prefix is in
        // dictionary. Then we recursively check for remaining string
        // str.substr(i, size-i) which is suffix of length size-i
        if (dictionaryContains( str.substr(0, i) ) &&
            wordBreak( str.substr(i, size-i) ))
```

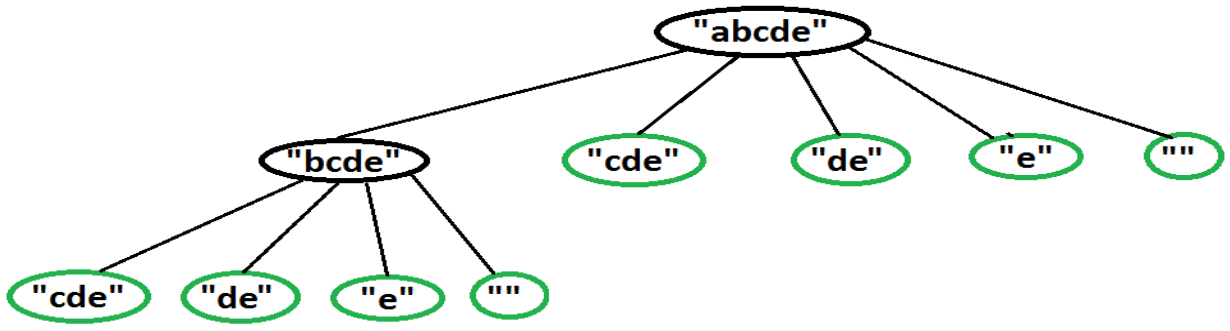
```
            return true;
    }

    // If we have tried all prefixes and none of them worked
    return false;
}

// Driver program to test above functions
int main()
{
    wordBreak("ilikesamsung")? cout <<"Yes\n": cout << "No\n";
    wordBreak("iiiiiiii")? cout <<"Yes\n": cout << "No\n";
    wordBreak("")? cout <<"Yes\n": cout << "No\n";
    wordBreak("ilikelikeimangoiii")? cout <<"Yes\n": cout << "No\n";
    wordBreak("samsungandmango")? cout <<"Yes\n": cout << "No\n";
    wordBreak("samsungandmangok")? cout <<"Yes\n": cout << "No\n";
    return 0;
}
```

Output:

```
Yes
Yes
Yes
Yes
Yes
No
```

Dynamic Programming

Why Dynamic Programming? The above problem exhibits overlapping sub-problems. For example, see the following partial recursion tree for string “abcde” in worst case.



Partial recursion tree for input string "abcde". The subproblems encircled with green color are overlapping subproblems

```
// A Dynamic Programming based program to test whether a given string can
// be segmented into space separated words in dictionary
#include <iostream>
#include <string.h>
using namespace std;

/* A utility function to check whether a word is present in dictionary or not.
  An array of strings is used for dictionary.  Using array of strings for
  dictionary is definitely not a good idea. We have used for simplicity of
  the program*/
int dictionaryContains(string word)
{
    string dictionary[] = {"mobile","samsung","sam","sung","man","mango",
                           "icecream","and","go","i","like","ice","cream"};
    int size = sizeof(dictionary)/sizeof(dictionary[0]);
    for (int i = 0; i < size; i++)
        if (dictionary[i].compare(word) == 0)
```

```
            return true;
    return false;
}

// Returns true if string can be segmented into space separated
// words, otherwise returns false
bool wordBreak(string str)
{
    int size = str.size();
    if (size == 0)   return true;

    // Create the DP table to store results of subroblems. The value wb[i]
    // will be true if str[0..i-1] can be segmented into dictionary words,
    // otherwise false.
    bool wb[size+1];
    memset(wb, 0, sizeof(wb)); // Initialize all values as false.

    for (int i=1; i<=size; i++)
    {
        // if wb[i] is false, then check if current prefix can make it true.
        // Current prefix is "str.substr(0, i)"
        if (wb[i] == false && dictionaryContains( str.substr(0, i) ))
            wb[i] = true;

        // wb[i] is true, then check for all substrings starting from
        // (i+1)th character and store their results.
        if (wb[i] == true)
        {
            // If we reached the last prefix
            if (i == size)
                return true;

            for (int j = i+1; j <= size; j++)
            {
                // Update wb[j] if it is false and can be updated
                // Note the parameter passed to dictionaryContains() is
                // substring starting from index 'i' and length 'j-i'
                if (wb[j] == false && dictionaryContains( str.substr(i, j-i) ))
                    wb[j] = true;

                // If we reached the last character
                if (j == size && wb[j] == true)
                    return true;
            }
        }
    }

    /* Uncomment these lines to print DP table "wb[]"
     for (int i = 1; i <= size; i++)
        cout << " " << wb[i]; */

    // If we have tried all prefixes and none of them worked
    return false;
}

// Driver program to test above functions
int main()
{
    wordBreak("ilikesamsung")? cout <<"Yes\n": cout << "No\n";
    wordBreak("iiiiiiii")? cout <<"Yes\n": cout << "No\n";
    wordBreak("")? cout <<"Yes\n": cout << "No\n";
    wordBreak("ilikelikeimangoiii")? cout <<"Yes\n": cout << "No\n";
    wordBreak("samsungandmango")? cout <<"Yes\n": cout << "No\n";
    wordBreak("samsungandmangok")? cout <<"Yes\n": cout << "No\n";
    return 0;
}
```

Output:

```
Yes
Yes
Yes
Yes
Yes
No
```

Exercise:
The above solutions only finds out whether a given string can be segmented or not. Extend the above Dynamic Programming solution to print all possible partitions of input string. See extended recursive solution for reference.

Examples:

```
Input: ilikeicecreamandmango
Output:
i like ice cream and man go
i like ice cream and mango
i like icecream and man go
i like icecream and mango

Input: ilikesamsungmobile
Output:
i like sam sung mobile
i like samsung mobile
```