Dynamic Programming | Set 32 (Word Break Problem)

GeeksforGeeks GATE CS Corner
GeeksforGeeks Practice

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 su
ng".
```

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 s
// space separated words in dictionary
#include <iostream>
using namespace std;

/* A utility function to check whether a word is present in di
  An array of strings is used for dictionary.  Using array of
  dictionary is definitely not a good idea. We have used for s
  the program*/
int dictionaryContains(string word)
{
    string dictionary[] = { "mobile","samsung","sam","sung","r
                           "icecream","and","go","i","like","
    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 separate
// 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(
        // str.substr(0, i) which is prefix (of input string)
        // length 'i'. We first check whether current prefix i
        // dictionary. Then we recursively check for remaining
        // str.substr(i, size-i) which is suffix of length siz
        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
```

Videos by GeeksforGeeks

Placement Course

Like us on Facebook

Follow us on Twitter

```
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 <<
    wordBreak("samsungandmango")? cout <<"Yes\n": cout << "No
    wordBreak("samsungandmangok")? cout <<"Yes\n": cout << "N
    return 0;
}
```

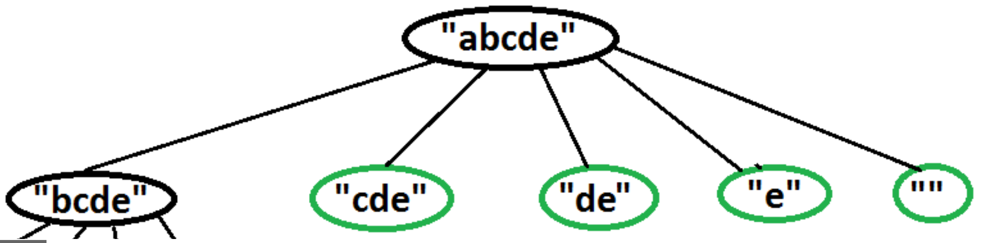
Run on IDE

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.

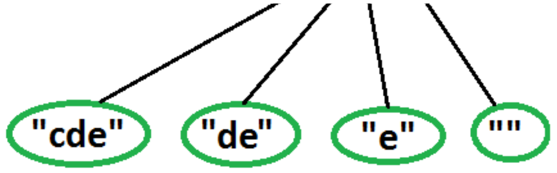


Recent Comments

Jumpstart Raspberry Pi &
Arduino Projects with Cayenne
Connect your hardware in minutes and hook up sensors, actuators, extensions and more.
Free Download

Popular Posts

- Top 10 Algorithms and Data Structures for Competitive Programming
- Top 10 algorithms in Interview Questions
- How to begin with Competitive Programming?
- Step by Step Guide for Placement Preparation
- Reflection in Java
- Memory Layout of C Programs
- Heavy Light Decomposition



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
// 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 di
  An array of strings is used for dictionary.  Using array of
  dictionary is definitely not a good idea. We have used for s
  the program*/
int dictionaryContains(string word)
{
    string dictionary[] = {"mobile","samsung","sam","sung","r
                           "icecream","and","go","i","like","
    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 separate
// 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
    // will be true if str[0..i-1] can be segmented into dicti
```

- Sorted Linked List to Balanced BST
- Generics in Java
- Aho-Corasick Algorithm for Pattern Searching
- Insertion Sort , Binary Search , QuickSort , MergeSort , HeapSort

- Common Interview Puzzles
- Interview Experiences
- Advanced Data Structures
- Design Patterns
- Dynamic Programming
- Greedy Algorithms
- Backtracking
- Pattern Searching
- Divide & Conquer
- Geometric Algorithms
- Searching
- Sorting
- Analysis of Algorithms
- Mathematical Algorithms
- Randomized Algorithms
- Recursion
- Game Theory

```
    // otherwise false.
    bool wb[size+1];
    memset(wb, 0, sizeof(wb)); // Initialize all values as fa

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

        // wb[i] is true, then check for all substrings starti
        // (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 upda
                // Note the parameter passed to dictionaryCont
                // substring starting from index 'i' and lengt
                if (wb[j] == false && dictionaryContains( str
                    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;
```

Tags

Adobe Advanced Data Structure Amazon Arrays Articles Binary Search Tree Bit Magic C/C++ Puzzles Competitive Programming cpp-library Divide and Conquer Dynamic Programming Experienced CS-DS-&-Algo GBlog Graph Graph Greedy Hash Internship Interview Experiences Java-Collections Linked Lists Mathematical Matrix MCQ Microsoft Misc number-digits Oracle Output Pattern Searching Project Python Searching Sorting Stack STL Strings subsequence Technical Scripter Trees XOR

```
}

// 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 <<
    wordBreak("samsungandmango")? cout <<"Yes\n": cout << "Nc
    wordBreak("samsungandmangok")? cout <<"Yes\n": cout << "N
    return 0;
}
```

Run on IDE

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
```

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Dynamic Programming Dynamic Programming

Related Posts:

- Maximum decimal value path in a binary matrix
- Longest subsequence such that difference between adjacents is one
- Sum of average of all subsets
- Count All Palindrome Sub-Strings in a String
- Maximum subsequence sum such that no three are consecutive
- Maximum sum alternating subsequence
- Maximum sum of pairs with specific difference
- Path with maximum average value

<< Previous Post Next Post >>

(Login to Rate and Mark)

3.5 Average Difficulty : **3.5/5.0**
Based on **38** vote(s)

Add to TODO List

Mark as DONE

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.

Load Comments