<https://algorithmstuff.wordpress.com/tag/memoization/>

Given a String and a dictionary of words, write a program that returns true if the given string can be formed by concatenating one or more of the words in the dictionary, eg.

Given the string S = “algorithmstuff”

and the dictionary dic = {“algorithm”, “stuff”}

Our function must return true since S can be formed by concatenating “algorithm” + “stuff”

but given S = “algorithmstuffisgood”

and dic = {“algorithms”, “good”, “stuff”} must return false since there’s no way to build S from the strings in the dictionary.

Solution

This is a very good problem for a programming interview since it has one “lazy solution” which is very inefficient but it can be converted to a very fast algorithm with just a simple modification, so you could think about the most obvious solution even if it’s slow and once you got a working solution you can think about how to improve it.

This problem is clearly a good candidate for a recursive solution, for each recursive call the idea is to start checking if the first character is in our dictionary, if so, we call the function recursively but now with string S starting with the next character, otherwise we check if the first two characters form a valid word and so on, for instance:

S = “hithere”

dic = { “there”, “hi”}

We start with the string “hithere” at index 0, so we check if the substring “h” is a valid word, since it’s not we continue now with the next character which is “i”, now we check if the substring “hi” is a valid word in the dictionary, it is so we know that “hi” can be formed by concatenating one or more words of the dictionary so we now call the function recursively but starting with the next index which is 2, so now we are gonna have the same function with the substring “there”, we are gonna stop when we reach the end of the string S.

This is basically a DFS algorithm (<http://en.wikipedia.org/wiki/Depth-first_search>) , here the code:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | public boolean stringBreak(String s, Set<String> dict) {      return dfs(s, 0, dict);  }  boolean dfs(String s, int i, Set<String> dict)  {      if(dict.contains(s.substring(i)))          return true;        for(int j = i; j < s.length(); ++j)          if(dict.contains(s.substring(i, j + 1)))              if(dfs(s, j + 1, dict, set)) return true;        return false;  } |

The solution above works! but it’s too slow since there are many overlapping calls (it does the same thing many times), let me show you with an example:

let’s say we have the string S = “helloworld”

and dict = {“he”, “e”, “h”}

Our recursive function is gonna build a tree like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | helloworld            /                \    h-elloworld           he-lloworld         |    e-lloworld |

As you can see the substring “lloworld” is gonna be computed two times and the result is gonna be the same, the problem here is that we could end up with too many overlapping calls which would be a waste of time. So, how can we avoid overlapping calls?

The solution is to have a record of which substrings we have already computed so that when our function is called again for the same substring we already know the answer, this technique is called MEMOIZATION (<http://en.wikipedia.org/wiki/Memoization>), Here is the code that shows this optimization.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | public boolean stringBreak(String s, Set<String> dict) {      HashSet<Integer> memo = new HashSet<Integer>(); // We just need to keep track of those indexes that we have already computed and the result is false      return dfs(s, 0, dict, memo);   }   boolean dfs(String s, int i, Set<String> dict, HashSet<Integer> memo)   {       if(dict.contains(s.substring(i)))           return true;       if(memo.contains(i)) // if we have already computed the result for this substring we just return the answer           return false;       //otherwise, compute the answer for this substring       for(int j = i; j < s.length(); ++j)           if(dict.contains(s.substring(i, j + 1)))               if(dfs(s, j + 1, dict, set)) return true;         //we just store the results for substrings which result is false       memo.add(i);       return false;   } |