Word Break

Given a string *s* and a dictionary of words *dict*, determine if *s* can be segmented into a space-separated sequence of one or more dictionary words.

For example, given
s = `"leetcode"`,
dict = `["leet", "code"]`.

Return true because `"leetcode"` can be segmented as `"leet code"`.

Solution 1

```
public class Solution {
    public boolean wordBreak(String s, Set<String> dict) {

        boolean[] f = new boolean[s.length() + 1];

        f[0] = true;


        /* First DP
        for(int i = 1; i <= s.length(); i++){
            for(String str: dict){
                if(str.length() <= i){
                    if(f[i - str.length()]){
                        if(s.substring(i-str.length(), i).equals(str)){
                            f[i] = true;
                            break;
                        }
                    }
                }
            }
        }*/

        //Second DP
        for(int i=1; i <= s.length(); i++){
            for(int j=0; j < i; j++){
                if(f[j] && dict.contains(s.substring(j, i))){
                    f[i] = true;
                    break;
                }
            }
        }

        return f[s.length()];
    }
}
```

written by segfault original link here

Solution 2

We use a boolean vector dp[]. dp[***i***] is set to true if a valid word (word sequence) ends there. The optimization is to look from current position ***i*** back and only substring and do dictionary look up in case the preceding position ***j*** with *dp[**j**] == true* is found.

```
bool wordBreak(string s, unordered_set<string> &dict) {
        if(dict.size()==0) return false;

        vector<bool> dp(s.size()+1,false);
        dp[0]=true;

        for(int i=1;i<=s.size();i++)
        {
            for(int j=i-1;j>=0;j--)
            {
                if(dp[j])
                {
                    string word = s.substr(j,i-j);
                    if(dict.find(word)!= dict.end())
                    {
                        dp[i]=true;
                        break; //next i
                    }
                }
            }
        }

        return dp[s.size()];
    }
```

written by paul7 original link here

Solution 3

People have posted elegant solutions using DP. The solution I post below using BFS is no better than those. Just to share some new thoughts.

We can use a graph to represent the possible solutions. The vertices of the graph are simply the positions of the first characters of the words and each edge actually represents a word. For example, the input string is "nightmare", there are two ways to break it, "night mare" and "nightmare". The graph would be

0-->5-->9

|__ __ _^

The question is simply to check if there is a path from 0 to 9. The most efficient way is traversing the graph using BFS with the help of a queue and a hash set. The hash set is used to keep track of the visited nodes to avoid repeating the same work.

For this problem, the time complexity is O(n^2) and space complexity is O(n), the same with DP. This idea can be used to solve the problem word break II. We can simple construct the graph using BFS, save it into a map and then find all the paths using DFS.

```
bool wordBreak(string s, unordered_set<string> &dict) {
    // BFS
    queue<int> BFS;
    unordered_set<int> visited;

    BFS.push(0);
    while(BFS.size() > 0)
    {
        int start = BFS.front();
        BFS.pop();
        if(visited.find(start) == visited.end())
        {
            visited.insert(start);
            for(int j=start; j<s.size(); j++)
            {
                string word(s, start, j-start+1);
                if(dict.find(word) != dict.end())
                {
                    BFS.push(j+1);
                    if(j+1 == s.size())
                        return true;
                }
            }
        }
    }

    return false;
}
```

written by GuaGua original link here