1. A message containing letters from `A-Z` is being encoded to numbers using the following mapping:

```
'A' -> 1
'B' -> 2
...
'Z' -> 26
```

Given an encoded message containing digits, determine the total number of ways to decode it.

For example,

Given encoded message `"12"`, it could be decoded as `"AB"` (1 2) or `"L"` (12).

The number of ways decoding `"12"` is 2.

Subscribe to see which companies asked this question

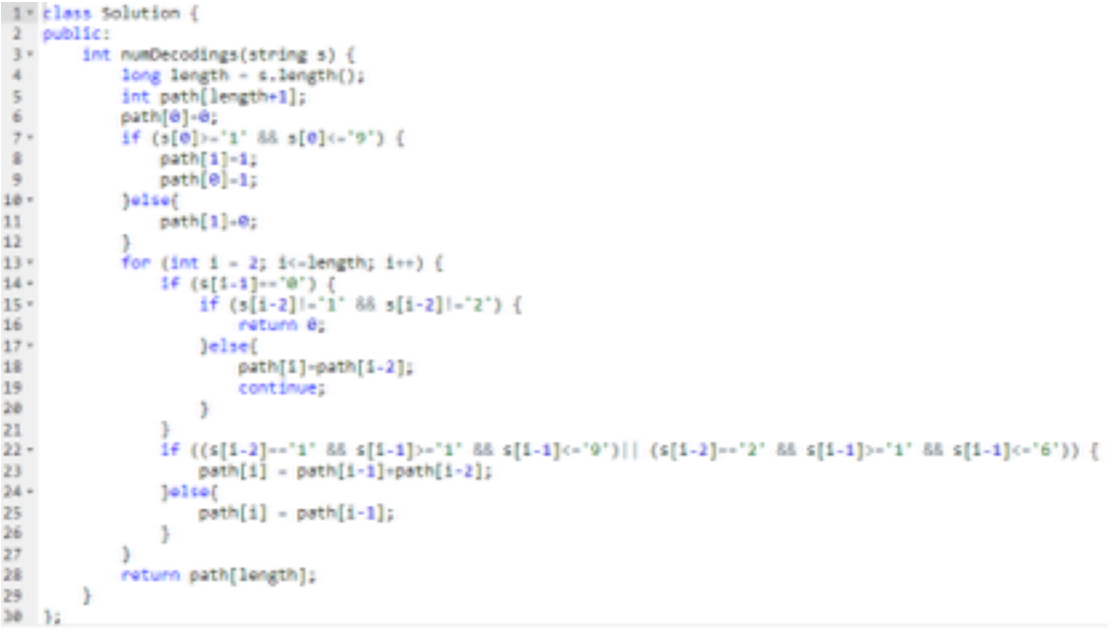
根据题意分析可得：

(1) 当出现数字0的时候必须和前面的数字构成10或者20，如果0前面没有数字或者出现的不是1或者2，那么说明错误。

(2) S[i]表示到达数字i的方法数，有递推关系式：

S[i] = s[i-2]+s[i-1] 其中i>=2 &&(

(s[i-1]∈[1,9] && s[i-2]==1)或(s[i-1]∈[1,6] && s[i-2]==2))



```cpp
class Solution {
public:
    int numDecodings(string s) {
        long length = s.length();
        int path[length+1];
        path[0]=0;
        if (s[0]>='1' && s[0]<='9') {
            path[1]=1;
            path[0]=1;
        }else{
            path[1]=0;
        }
        for (int i = 2; i<=length; i++) {
            if (s[i-1]=='0') {
                if (s[i-2]!='1' && s[i-2]!='2') {
                    return 0;
                }else{
                    path[i]=path[i-2];
                    continue;
                }
            }
            if ((s[i-2]=='1' && s[i-1]>='1' && s[i-1]<='9')|| (s[i-2]=='2' && s[i-1]>='1' && s[i-1]<='6')) {
                path[i] = path[i-1]+path[i-2];
            }else{
                path[i] = path[i-1];
            }
        }
        return path[length];
    }
};
```

2. Given a binary tree, return the *inorder* traversal of its nodes' values.

For example:

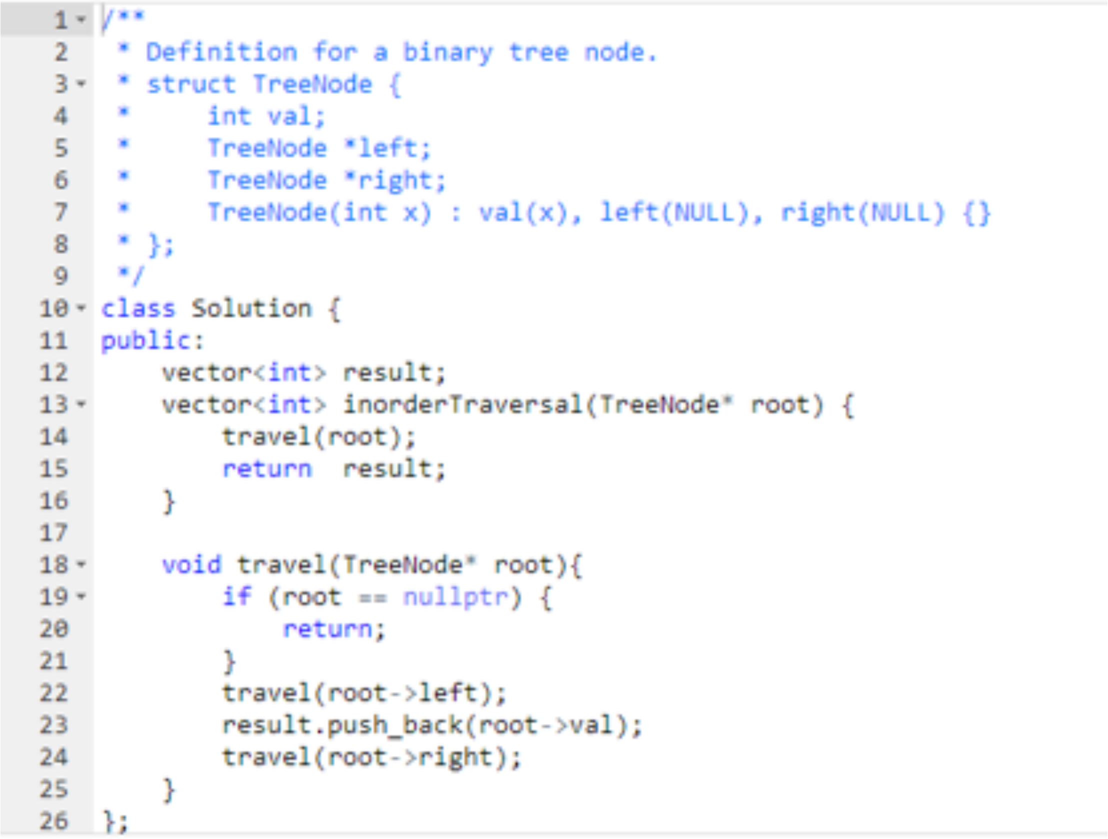
Given binary tree `{1,#,2,3}`,

```
   1

    \

     2

    /

   3
```

return `[1,3,2]`.

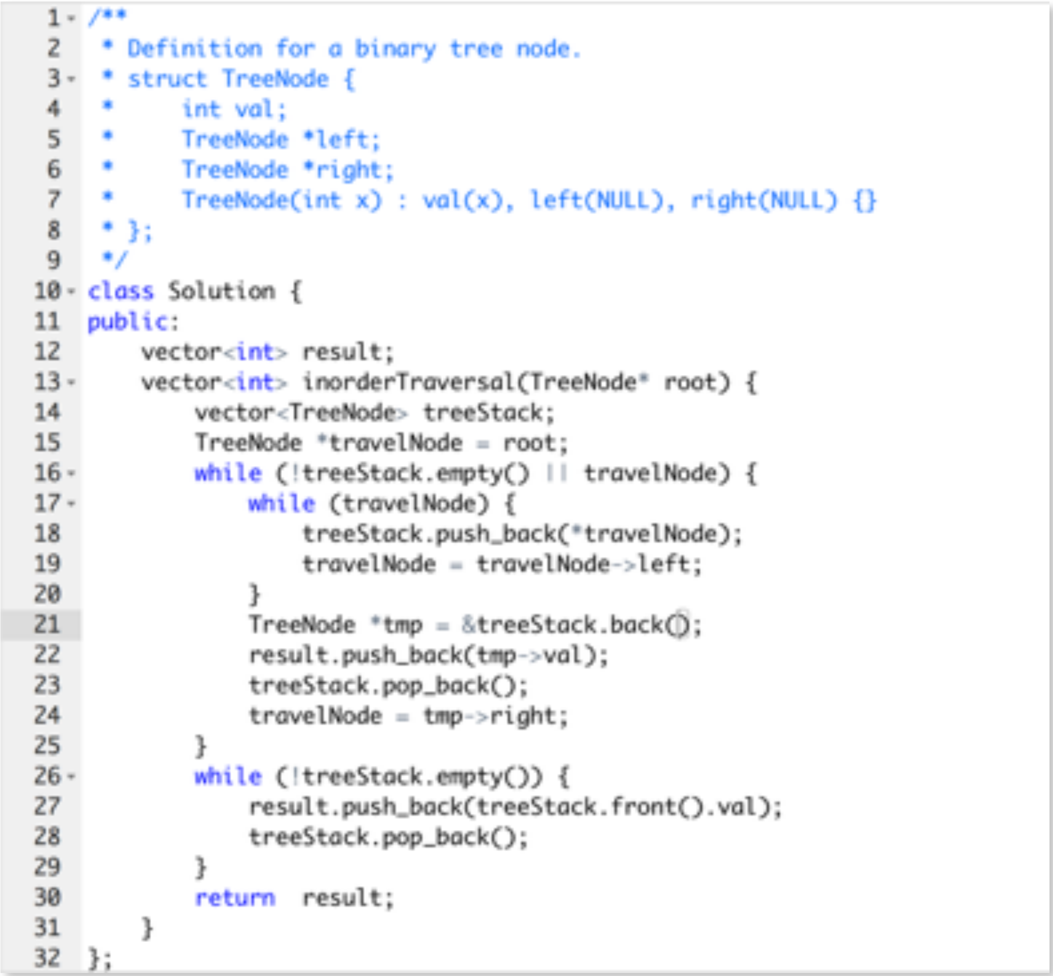
Note: Recursive solution is trivial, could you do it iteratively?

思路：中序遍历树，首先想到了用递归的思路，按照左子树-根节点-右子树的顺序遍历，代码如下：



```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<int> result;
    vector<int> inorderTraversal(TreeNode* root) {
        travel(root);
        return  result;
    }

    void travel(TreeNode* root){
        if (root == nullptr) {
            return;
        }
        travel(root->left);
        result.push_back(root->val);
        travel(root->right);
    }
};
```

但是发现递归会造成用时过长，使用栈：



```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<int> result;
    vector<int> inorderTraversal(TreeNode* root) {
        vector<TreeNode> treeStack;
        TreeNode *travelNode = root;
        while (!treeStack.empty() || travelNode) {
            while (travelNode) {
                treeStack.push_back(*travelNode);
                travelNode = travelNode->left;
            }
            TreeNode *tmp = &treeStack.back();
            result.push_back(tmp->val);
            treeStack.pop_back();
            travelNode = tmp->right;
        }
        while (!treeStack.empty()) {
            result.push_back(treeStack.front().val);
            treeStack.pop_back();
        }
        return  result;
    }
};
```

3. Given a string *s* and a dictionary of words *dict*, add spaces in *s* to construct a sentence where each word is a valid dictionary word.

Return all such possible sentences.

For example, given

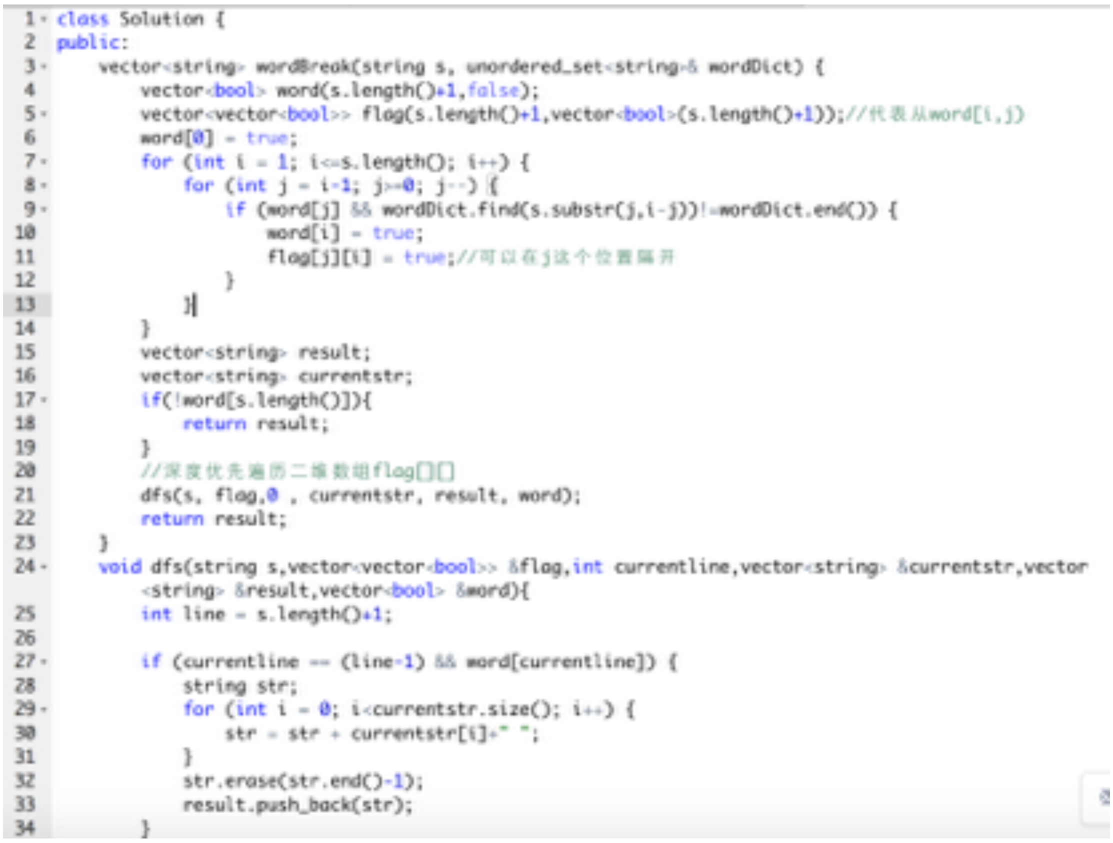
s = `"catsanddog"`,

dict = `["cat", "cats", "and", "sand", "dog"]`.

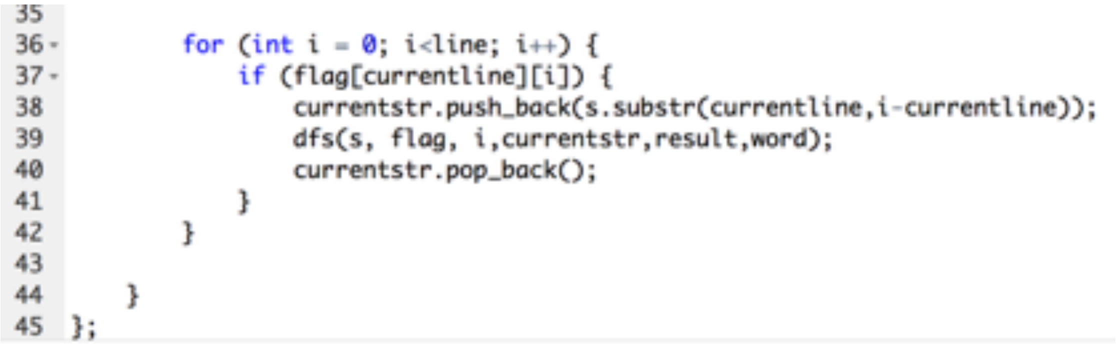
A solution is `["cats and dog", "cat sand dog"]`.

Subscribe to see which companies asked this question

类比word Break的问题，work Break解决的是是否可分的问题，而这道题不仅要解决是否可分，还要将所有的可能性列举出来，自然而然想到利用回溯的思想，如果可分就要记录下当前可分的位置。



```cpp
class Solution {
public:
    vector<string> wordBreak(string s, unordered_set<string>& wordDict) {
        vector<bool> word(s.length()+1,false);
        vector<vector<bool>> flag(s.length()+1,vector<bool>(s.length()+1));//代表从word[i,j)
        word[0] = true;
        for (int i = 1; i<=s.length(); i++) {
            for (int j = i-1; j>=0; j--) {
                if (word[j] && wordDict.find(s.substr(j,i-j))!=wordDict.end()) {
                    word[i] = true;
                    flag[j][i] = true;//可以在j这个位置隔开
                }
            }
        }
        vector<string> result;
        vector<string> currentstr;
        if(!word[s.length()]){
            return result;
        }
        //深度优先遍历二维数组flag[][]
        dfs(s, flag,0 , currentstr, result, word);
        return result;
    }
    void dfs(string s,vector<vector<bool>> &flag,int currentline,vector<string> &currentstr,vector<string> &result,vector<bool> &word){
        int line = s.length()+1;

        if (currentline == (line-1) && word[currentline]) {
            string str;
            for (int i = 0; i<currentstr.size(); i++) {
                str = str + currentstr[i]+" ";
            }
            str.erase(str.end()-1);
            result.push_back(str);
        }
```



```cpp

        for (int i = 0; i<line; i++) {
            if (flag[currentline][i]) {
                currentstr.push_back(s.substr(currentline,i-currentline));
                dfs(s, flag, i,currentstr,result,word);
                currentstr.pop_back();
            }
        }

    }
};
```