1. Add and Search Word - Data structure design

Design a data structure that supports the following two operations:

void addWord(word)  
bool search(word)

search(word) can search a literal word or a regular expression string containing only letters a-z or .. A . means it can represent any one letter.

**Example:**

addWord("bad")  
addWord("dad")  
addWord("mad")  
search("pad") -> false  
search("bad") -> true  
search(".ad") -> true  
search("b..") -> true

**Note:** You may assume that all words are consist of lowercase letters a-z.

**解法1** hashmap。将单词按照长度映射

class WordDictionary {  
public:  
 unordered\_map<int, set<string>>mp;  
 /\*\* Initialize your data structure here. \*/  
 WordDictionary() { }  
 /\*\* Adds a word into the data structure. \*/  
 void addWord(string word) {  
 int m = word.size();  
 mp[m].insert(word);  
 }  
 /\*\* Returns if the word is in the data structure. A word could contain the dot character '.' to represent any one letter. \*/  
 bool search(string word) {  
 int m = word.size();  
 if(mp.count(m) == 0)return false;  
 for(auto s : mp[m]){  
 int i;  
 for(i = 0; i < m; ++i){  
 if(s[i] == word[i] || word[i] == '.')continue;  
 else break;  
 }  
 if(i == m)return true;  
 }  
 return false;  
 }  
};

**解法2** trie-tree。查找时，对于包含了.的部分，递归往后查询

class trie\_node{  
public:  
 bool isWord;  
 vector<trie\_node\*>child;  
 trie\_node() : isWord(false), child(26, NULL){}  
 ~trie\_node(){  
 for(auto &c : child)delete c;  
 }  
};  
class WordDictionary {  
public:  
 trie\_node\* root;  
 /\*\* Initialize your data structure here. \*/  
 WordDictionary() {  
 root = new trie\_node;  
 }  
 /\*\* Adds a word into the data structure. \*/  
 void addWord(string s) {  
 trie\_node \*cur = root;  
 for(int i = 0; i < s.size(); ++i){  
 int idx = s[i] - 'a';  
 if(cur->child[idx] == NULL){  
 cur->child[idx] = new trie\_node();  
 }  
 cur = cur->child[idx];  
 }  
 cur->isWord = true;  
 }  
 /\*\* Returns if the word is in the data structure. A word could contain the dot character '.' to represent any one letter. \*/  
 bool search(string word) {  
 return \_search(word, root);  
 }  
 bool \_search(string s, trie\_node\* root){  
 if(s == "")return root->isWord;  
 if(s[0] == '.'){  
 for(int j = 0; j < 26; ++j){  
 if(root->child[j] && \_search(s.substr(1), root->child[j]))return true;  
 }  
 return false;  
 }else{  
 if(root->child[s[0]-'a'] == NULL)return false;  
 return \_search(s.substr(1), root->child[s[0]-'a']);  
 }  
 }  
};