Lecture – 18 Backtracking

1. <https://leetcode.com/problems/palindrome-partitioning/>

class Solution {

bool validPalindrome(string s, int start, int end)

{

while(start <= end)

{

if(s[start]!=s[end])

{

return false;

}

start++;

end--;

}

return true;

}

void validPartition(int index, string s, vector<string>&curr, vector<vector<string>>&res)

{

// base case

if(index==s.size())

{

res.push\_back(curr);

return;

}

for(int i=index; i<s.size(); i++)

{

if(validPalindrome(s, index, i))

{

curr.push\_back(s.substr(index, i-index+1));

validPartition(i+1, s, curr, res);

curr.pop\_back();

}

}

}

public:

vector<vector<string>> partition(string s) {

vector<vector<string>> res;

vector<string> curr;

validPartition(0, s, curr, res);

return res;

}

};

1. <https://leetcode.com/problems/word-search/>

class Solution {

bool check(vector<vector<char>>& board, string word, int index, int i, int j)

{

if(i<0 || i>=board.size() || j<0 || j>=board[0].size() || board[i][j]!=word[index])

{

return false;

}

if(index==word.size()-1)

{

return true;

}

bool found = false;

char temp = board[i][j];

board[i][j] = '#';

found = check(board, word, index+1, i-1, j) ||

check(board, word, index+1, i, j-1) ||

check(board, word, index+1, i+1, j) ||

check(board, word, index+1, i, j+1);

board[i][j] = temp;

return found;

}

public:

bool exist(vector<vector<char>>& board, string word) {

for(int i=0; i<board.size(); i++)

{

for(int j=0; j<board[0].size(); j++)

{

if(board[i][j] == word[0] && check(board, word, 0, i, j))

{

return true;

}

}

}

return false;

}

};