Search with DFS and BFS

2019年10月13日　　上午 12:16

Commonly connected with Combination and Enumeration: Find all possilbe scenarios like finding pathes

DFS 更适于快速查询以及 限制条件 Leetcode 51 N-Queens and 79 Word Search

BFS 最短路径 Leetcode 675，127 word ladder

★ Leetcode 51 N-Queens 79 Word-search :: **DFS前标记visited, 后标记unvisited**

```
// Try every column
    for (int x = 0; x < n; ++x) {
      if (!available(x, y, n)) continue;
      updateBoard(x, y, n, true);
      nqueens(n, y + 1);
      updateBoard(x, y, n, false);
    }
```

? Private 成员变量可以直接被成员函数调用（public and private），不用传入

Classical : 17 Letter Combinations of a Phone Number ----> templates

❤ ➢ **DFS: 找寻可能的答案，**vector<int>& cur; Leetcode 39, 40 combination sum 77，78**对应 HuaHua Search 1**

- **Leetcode 51 nqueens ---- good example to learn DFS !!!! How to DFS and How to decide whether it is valid or not**
- Cur.push_back()
  ....
  Cur.pop_back()**// 每次修改完后都要恢复所有的改动，这样才能backtracking**

- 防止答案重复，可以先用set<vector<int>> 后面利用 vector<vector<int>> (ans.begin(),ans.end());
- Sort(ans.begin(),ans.end()); **先sort再计算结果**可以避免很多不想要的结果！！！i > index && nums[i] == nums[i-1]
- if (i > index && nums[i] == nums[i-1]) continue; 防止重复结果
- **Permutations:** vector<int> used 来判断 第i个元素有没有用过 Leetcode 46 47 注意相同结果怎么消除的条件！！！

```
for (int i = 0; i < nums.size(); ++i) { // Permutation|| leetcode 47// For represents all possible scenarios for this position
        if (used[i]) continue;
        if (i > 0 && nums[i] == nums[i - 1] && !used[i - 1]) continue;
        used[i] = 1;
        cur.push_back(nums[i]);
        dfs(nums, cur, used, ans);
        cur.pop_back();// trackbacking
        used[i] = 0;
      }
```

- **Combinations:** leetcode 22 generate parentheses recursion当 剩余 left > right 的时候，一定会产生)(，这样就是不合法的

SP4 Time/Space Complexity of Recursion

```
combination(d, s):
  if d == n: return func()
  for i in range(s, n):
    combination(d + 1, i + 1)
```

```
permutation(d, used):
  if d == n: return func()
  for i in range(0, n):
    if i in used: continue
    used.add(i)
    permutation(d + 1, used)
    used.remove(i)
```

```
Time complexity:
T(n) = T(n-1) + T(n-2) + ... + T(1)
     = O(2^n)

Space complexity:
O(n)
```

```
Time complexity:
T(n) = n * T(n-1)
     = O(n!)

Space complexity:
O(n)
```

Template

```
void DFS(arguments)
{
    if (achieve total number of items)
    {
        all_ans.push_back(cur);
    }
    // i = 0, permutations or sequences matter; i = index, combinations;
    // Possible for no iterations, just two possible situations, like uppercase or lowercase alphabetes
    for (int i = index or 0; i < N; ++i)
    {
        if (doesn't meet the requirement) return;
        cur.add(item);
        DFS(updated arguments);
        cur.pop(item);
    }
}
```

- 当判断正误的时候，把不合格的条件都放在开头
- **Leetcode 79 Word Search**枚举四种可能情况，将判断条件全部移到开头一起，这样简洁高效

小技巧：对于字母的遍历，visited可以在 board里面修改为‘0’ 而不需要 新建visited vector

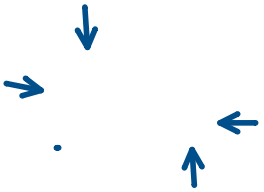
```
class Solution {
public:
    bool exist(vector<vector<char>>& board, string word) {
       if (board.size() == 0) return false;
        h = board.size();
        w = board[0].size();
        for (int i = 0; i < h; ++i)
        {
            for (int j = 0; j < w; ++j)
            {
                if (search(board,word,0,i,j)) return true;
            }
        }
        return false;
    }

    bool search(vector<vector<char>>& board, const string& word,int d, int r, int c)
    {
        if (r < 0 || c < 0 || r == h || c == w || word[d] != board[r][c])
        {
            return false;
        }

        if (d == word.length()-1)
        {
            return true;
        }
        char cur = board[r][c];
        board[r][c] = '0';
        bool found = search(board,word,d+1,r,c+1)
            || search(board,word,d+1,r,c-1)
            || search(board,word,d+1,r+1,c)
            || search(board,word,d+1,r-1,c);
        board[r][c] = cur;
        return found;
    }

private:
    int h,w;
};
```

下面是我刚开始的实现方法，十分低级，代码也不够简洁，要思考如何整合信息

```
private:
    void search(vector<vector<char>>& board, vector<vector<int>> visited, string& word, int r, int c, int index)
    {
        if (index == word.size())
        {
            find = true;
            return;
        }
        char s = word[index];
        if ((r+1 < r_size) && (board[r+1][c] == s) && (visited[r+1][c] == 0))
        {
            visited[r+1][c] = 1;
            search(board,visited,word,r+1,c,index+1);
            visited[r+1][c] = 0;
        }
        if ((r-1 >= 0) && board[r-1][c] == s && visited[r-1][c] == 0)
        {
            visited[r-1][c] = 1;
            search(board,visited,word,r-1,c,index+1);
            visited[r-1][c] = 0;
        }
        if ((c-1 >= 0) && board[r][c-1] == s && visited[r][c-1] == 0)
        {
            visited[r][c-1] = 1;
            search(board,visited,word,r,c-1,index+1);
            visited[r][c-1] = 0;
        }
        if ((c+1 < c_size) && board[r][c+1] == s && visited[r][c+1] == 0)
        {
            visited[r][c+1] = 1;
            search(board,visited,word,r,c+1,index+1);
            visited[r][c+1] = 0;
        }
```

❤️⭐➢ **DFS找联通分量** Leetcode 200 以及 934 ，**联通小岛**

```
// Author: Huahua
// Time complexity: O(mn)
// Running time: 6 ms
class Solution {
public:
    int numIslands(vector<vector<char>>& grid) {
        if (grid.empty()) return 0;
        int m = grid.size();
        int n = grid[0].size();
        int ans = 0;
        for (int y = 0; y < m; ++y)
            for (int x = 0; x < n; ++x) {
                ans += grid[y][x] - '0';
                dfs(grid, x, y, m, n);
            }
        return ans;
    }
private:
    void dfs(vector<vector<char>>& grid, int x, int y, int m, int n) {
        if (x < 0 || y < 0 || x >= n || y >= m || grid[y][x] == '0')
            return;
        grid[y][x] = '0';
        dfs(grid, x + 1, y, m, n);
        dfs(grid, x - 1, y, m, n);
        dfs(grid, x, y + 1, m, n);
        dfs(grid, x, y - 1, m, n);
```



```
    }
};
```

来自 <https://zxi.mytechroad.com/blog/searching/leetcode-200-number-of-islands/>

❤️ DFS 解决均分集合问题：

Leetcode 698:

Given an array of integers nums and a positive integer k, find whether it's possible to divide this array into k non-empty subsets whose sums are all equal.

用visited 记录, backtracking

然后start记录开始点，当=target, 计算后面的！

还是用模板

```
class Solution {
public:
    bool canPartitionKSubsets(vector<int>& nums, int k) {
        int sum = accumulate(nums.begin(), nums.end(), 0);
        if (sum % k != 0) return false;
        sort(nums.begin(), nums.end(), greater<int>());
        vector<bool> visited(nums.size(), false);
        return helper(nums, k, sum / k, 0, 0, visited);
    }
    bool helper(vector<int>& nums, int k, int target, int start, int curSum, vector<bool>& visited) {
        if (k == 1) return true;
        if (curSum > target) return false;
        if (curSum == target) return helper(nums,k-1,target,0,0,visited);
        for (int i = start; i < nums.size(); ++i)
        {
            if (visited[i]) continue;
            visited[i] = true;
            if (helper(nums,k,target,i+1,curSum+nums[i],visited)) return true;
            visited[i] = false;
        }
        return false;
    }
};
```

🟢 **DFS partition** **分三种情况分割 string, using substr(I,j)**

For 循环其实是各种情况的可能性，因为该题必须包括头部，也就是start，所以从1 到 3 进行分割成三种情况；如果不考虑顺序，就可以随机选择，那么每次就是从0 开始，看哪些没有用过；

```
class Solution {
public:
  vector<string> restoreIpAddresses(string s) {
    vector<string> ans;
    string ip;
    dfs(0, s, ip, ans);
    return ans;
  }

private:
  void dfs(int d, string s, string ip, vector<string> &ans) {
    int l = s.length();
    if (d == 4)
    {
        if (l == 0) ans.push_back(ip);
        return;
    }
    for (int i = 1; i <= min(3,s[0] =='0'? 1:l); ++i)
    {
        string ss = s.substr(0,i);
        if (i == 3 && stoi(ss) > 255) return;
        dfs(d+1,s.substr(i),ip + (d==0? "":".") + ss, ans);
    }
  }
};
```

★ DFS遍历图复制

```
class Solution {
public:
    Node* cloneGraph(Node* node) {
        unordered_map<Node*, Node*> m;
        return helper(node, m);
    }
    Node* helper(Node* node, unordered_map<Node*, Node*>& m) {
       if (!node) return NULL;
       if (m.count(node)) return m[node];
       Node* clone = new Node(node -> val);
        m[node] = clone;
       for (auto& neighbor: node->neighbors)
       {
           clone -> neighbors.push_back(helper(neighbor,m));
       }
        return clone;
    }
};
```

➢ 当提及 shortest path 以及间距是 1 的时候就要用BFS 更快，而不是用DFS

BFS

转化 vector to set or map， 查找存不存在更快

花花酱 LeetCode 127. Word Ladder

```
q.push(start)
step = 0

while not q.empty():
  ++step
  size = q.size()
  while size-- > 0:
    node = q.pop()
    new_nodes = expand(node)
    if goal in new_nodes: return step + 1
    q.append(new_nodes)

return NOT_FOUND
```

BFS

```
s1.insert(start)
s2.insert(end)
step = 0

while not (s1.empty() || s2.empty()):
  ++step
  swap*(s1, s2)
  s = {}
  for node in s1:
    new_nodes = expand(node)
    if any(new_nodes) in s2: return step + 1
    s.append(new_nodes)
  s1 = s

return NOT_FOUND
```

Bidirectional BFS

Single BFS using **queue** to front, pop, push for all items

Bidirectional BFS using **two sets** to store every level's items after one BFS, not all of items;

Tips: while (size--){.... }遍历vector 里面元素，很简洁的方式

Dirs vector to present four directions, more concise

```
vector<int> dirs{0, -1, 0, 1, 0};// visit four negibors, more concise
for (int k = 0; k < 4; ++k) {
    int x = j + dirs[k];
    int y = i + dirs[k + 1];
}
```

BFS 转换思维：找**所有点到**最近0的距离，可以**对所有0做BFS,**更新每个点的steps from 0

求最短路径往往可以逆向思考，从少的到多的

求所有点到某点的最短路径，**一般也可以用 DP**，周围点联系起来

Leetcode 542 01 Matrix

DP form 先根据左边和上边持续更新，从左上角到右下角；

再根据右边和下面更新距离，从右下角到左上角，这样保证距离最短