图和搜索的陨落篇

该篇主要介绍图思想的应用。图的题目相对较少，我们要掌握的就是图的 BFS 和 DFS 思想以及如何将该思想应用到可能的题目中。该篇首先根据 Clone Graph 这道题讲解如何使用 BFS 和 DFS 在对图进行遍历的同时克隆该图，然后简单介绍拓扑排序，最后介绍和讲解几道使用 DFS 搜索来解决的题目。注意，Word Ladder 两题是如何使用 BFS 思想。

Clone Graph，该题中的 map 有 2 个作用，其一保存了原图结点和新结点的映射关系；其二可以检查某结点是否已经克隆过了。

```
    // Solution 1 - BFS
    public UndirectedGraphNode cloneGraph(UndirectedGraphNode node) {
        if(node == null){
            return null;
        }
        HashMap<UndirectedGraphNode, UndirectedGraphNode> map = new HashMap<UndirectedGraphNode, UndirectedGraphNode>();
        UndirectedGraphNode copy = new UndirectedGraphNode(node.label);
        map.put(node, copy);
        LinkedList<UndirectedGraphNode> queue = new LinkedList<UndirectedGraphNode>();
        queue.offer(node);
        while(!queue.isEmpty()){
            UndirectedGraphNode cur = queue.poll();
            for(int i = 0; i < cur.neighbors.size(); i++){
                if(!map.containsKey(cur.neighbors.get(i))){
                    copy = new UndirectedGraphNode(cur.neighbors.get(i).label);
                    map.put(cur.neighbors.get(i), copy);
                    queue.offer(cur.neighbors.get(i));
                }
                map.get(cur).neighbors.add(map.get(cur.neighbors.get(i)));
            }

        }
        return map.get(node);
    }
    // Solution 2 - DFS
    public UndirectedGraphNode cloneGraph(UndirectedGraphNode node) {
        if(node == null){
            return null;
        }
        HashMap<UndirectedGraphNode, UndirectedGraphNode> map = new
HashMap<UndirectedGraphNode, UndirectedGraphNode>();
        UndirectedGraphNode copy = new UndirectedGraphNode(node.label);
        map.put(node, copy);
        LinkedList<UndirectedGraphNode> stack = new
LinkedList<UndirectedGraphNode>();
        stack.push(node);
        while(!stack.isEmpty()){
            UndirectedGraphNode cur = stack.pop();
            for(int i = 0; i < cur.neighbors.size(); i++){
                if(!map.containsKey(cur.neighbors.get(i))){
                    copy = new UndirectedGraphNode(cur.neighbors.get(i).label);
                    map.put(cur.neighbors.get(i), copy);
                    stack.push(cur.neighbors.get(i));
                }
                map.get(cur).neighbors.add(map.get(cur.neighbors.get(i)));
            }
        }
        return map.get(node);
    }

    // Solution 3 – DFS (Recursion Version)
    public UndirectedGraphNode cloneGraph(UndirectedGraphNode node) {
        if(node == null){
            return null;
        }
        HashMap<UndirectedGraphNode, UndirectedGraphNode> map = new
HashMap<UndirectedGraphNode, UndirectedGraphNode>();
        UndirectedGraphNode copy = new UndirectedGraphNode(node.label);
        map.put(node, copy);
        helper(node, map);
        return copy;
    }
    private void helper(UndirectedGraphNode node,
HashMap<UndirectedGraphNode, UndirectedGraphNode> map){
        for(int i = 0; i < node.neighbors.size(); i++){
            UndirectedGraphNode cur = node.neighbors.get(i);
            if(!map.containsKey(cur)){
                UndirectedGraphNode copy = new
UndirectedGraphNode(cur.label);
                map.put(cur, copy);
                helper(cur, map);
            }
            map.get(node).neighbors.add(map.get(cur));
        }
    }
```

Topological Sorting简介：

http://www.geeksforgeeks.org/topological-sorting/

Subsets，Permutations，Combination Sum，N-Queens，Palindrome Partitioning，这几道题目在回溯篇都有详细的讲解，这里就不再赘述。

Word Ladder，将每个单词可能的变换想象成一个图，图的每层之间的单词只有一个字母不相同。我们模仿 BFS 遍历，从 start 单词开始第一次找到 end 单词时返回当时层数即可。

```
    public int ladderLength(String start, String end, Set<String> dict) {
        if(start.length() != end.length() || dict == null || dict.size() == 0){
            return 0;
        }
        LinkedList<String> queue = new LinkedList<String>();
        queue.offer(start);
        HashSet<String> visited = new HashSet<String>();
        visited.add(start);
        int step = 1;
        int curNum = 1;
        int nextNum = 0;
        while(!queue.isEmpty()){
            String cur = queue.poll();
            curNum--;
            for(int i = 0; i < cur.length(); i++){
                char[] charArr = cur.toCharArray();
                for(char c = 'a'; c <= 'z'; c++){
                    charArr[i] = c;
                    String word = new String(charArr);
                    if(word.equals(end)){
                        return step + 1;
                    } else if(dict.contains(word) && !visited.contains(word)){
                        queue.offer(word);
                        visited.add(word);
                        nextNum++;
                    }
                }
            }
            if(curNum == 0){
                curNum = nextNum;
                nextNum = 0;
                step++;
            }
        }
        return 0;
    }
```

Word Ladder II，该题目过难，放弃治疗，提供 2 种解法仅供参考。

```
    class StringWithLevel{
        String str;
        int level;
        public StringWithLevel(String str, int level){
            this.str = str;
            this.level = level;
        }
    }

    public ArrayList<ArrayList<String>> findLadders(String start, String end,
Set<String> dict) {
        ArrayList<ArrayList<String>> res = new ArrayList<ArrayList<String>>();
        HashSet<String> unvisitedSet = new HashSet<String>();
        unvisitedSet.addAll(dict);
        unvisitedSet.add(start);
        unvisitedSet.remove(end);
        Map<String, List<String>> nextMap = new HashMap<String, List<String>>();
        for(String s : unvisitedSet){
            nextMap.put(s, new ArrayList<String>());
        }
        LinkedList<StringWithLevel> queue = new LinkedList<StringWithLevel>();
        queue.offer(new StringWithLevel(end, 0));
        int preLevel = 0;
        int curLevel = 0;
        int finalLevel = Integer.MAX_VALUE;
        boolean found = false;
        HashSet<String> curLevelVisited = new HashSet<String>();
        while(!queue.isEmpty()){
            StringWithLevel cur = queue.poll();
            String curStr = cur.str;
            curLevel = cur.level;
            if(found && curLevel > finalLevel){
                break;
            }
            if(curLevel > preLevel){
                unvisitedSet.removeAll(curLevelVisited);
            }
            preLevel = curLevel;
            char[] charArr = curStr.toCharArray();
            for(int i = 0; i < curStr.length(); i++){
                char originalChar = charArr[i];
                boolean foundCurCycle = false;
                for(char c = 'a'; c <= 'z'; c++){
                    charArr[i] = c;
                    String newStr = new String(charArr);
                    if(originalChar != c && unvisitedSet.contains(newStr)){
                        nextMap.get(newStr).add(curStr);
                        if(newStr.equals(start)){
                            found = true;
                            foundCurCycle = true;
                            finalLevel = curLevel;
                            break;
                        } else if(curLevelVisited.add(newStr)){
                            queue.offer(new StringWithLevel(newStr, curLevel +
1));
                        }
                    }
                }
                charArr[i] = originalChar;
                if(foundCurCycle){
                    break;
                }
            }
        }
        if(found){
            List<String> list = new ArrayList<String>();
            list.add(start);
            helper(start, end, finalLevel + 1, list, res, nextMap);
        }
        return res;
    }

    private void helper(String cur, String end, int level, List<String> list,
ArrayList<ArrayList<String>> res, Map<String, List<String>> nextMap){
        if(cur.equals(end)){
            res.add(new ArrayList<String>(list));
            return;
        } else if(level > 0){
            List<String> parents = nextMap.get(cur);
            for(String s : parents){
                list.add(s);
                helper(s, end, level - 1, list, res, nextMap);
                list.remove(list.size() - 1);
            }
        }
    }

    public List<List<String>> findLadders(String start, String end, Set<String> dict) {
        List<List<String>> ladders = new ArrayList<List<String>>();
        Map<String, List<String>> map = new HashMap<String, List<String>>();
        Map<String, Integer> distance = new HashMap<String, Integer>();
        dict.add(start);
        dict.add(end);
        bfs(map, distance, start, end, dict);
        List<String> path = new ArrayList<String>();
        dfs(ladders, path, end, start, distance, map);
        return ladders;
    }

    void dfs(List<List<String>> ladders, List<String> path, String crt, String start,
Map<String, Integer> distance, Map<String, List<String>> map) {
        path.add(crt);
        if (crt.equals(start)) {
            Collections.reverse(path);
            ladders.add(new ArrayList<String>(path));
            Collections.reverse(path);
        } else {
            for (String next : map.get(crt)) {
       if (distance.containsKey(next) && distance.get(crt) == distance.get(next) + 1) {
                    dfs(ladders, path, next, start, distance, map);
                }
            }
        }
        path.remove(path.size() - 1);
    }

    void bfs(Map<String, List<String>> map, Map<String, Integer> distance, String
start, String end, Set<String> dict) {
        Queue<String> q = new LinkedList<String>();
        q.offer(start);
        distance.put(start, 0);
        for (String s : dict) {
            map.put(s, new ArrayList<String>());
        }
        while (!q.isEmpty()) {
            String crt = q.poll();
            List<String> nextList = expand(crt, dict);
            for (String next : nextList) {
                map.get(next).add(crt);
                if (!distance.containsKey(next)) {
                    distance.put(next, distance.get(crt) + 1);
                    q.offer(next);
                }
            }
        }
    }
```

```
List<String> expand(String crt, Set<String> dict) {
    List<String> expansion = new ArrayList<String>();
    for (int i = 0; i < crt.length(); i++) {
        for (char ch = 'a'; ch <= 'z'; ch++) {
            if (ch != crt.charAt(i)) {
                String expanded = crt.substring(0, i) + ch
                        + crt.substring(i + 1);
                if (dict.contains(expanded)) {
                    expansion.add(expanded);
                }
            }
        }
    }
    return expansion;
}
```

时间复杂度总结:

DFS:

1. Find all possible solutions - O(2 ^ n)
2. Permutations / Subsets - O(n!)

BFS:

1. Graph traversal - O(m)
2. Find shortest path in a sample graph - O(n)