Depth First Search && Breadth First Search

Building Blocks

- Vertices; Edges
- Indegree; Outdegree
- Directed Graph; Undirected Graph; Simple Graph; Connected Graph
- Subgraph
- Dense Graph; Spares Graph
- Walk, Path, Cycle, Loop
- Tree; Forest
- Adjacency List; Adjacency Matrix
- Number of edges for a directed/undirected graph?
- Relationship between edges and vertices in a tree? What about a connected undirected graph and a forest?
- Bipartite

Please refer to:
https://www.cs.cmu.edu/~clo/www/CMU/DataStructures/Lessons/lesson5_1.htm
or google if you are not familiar with some of them.

Depth First Search

- A traversal/ searching algorithm for tree or graph data structures
- Recall pre-order, in-order, post-order traversal on a tree from week 2
- 一路扎到底

Breadth First Search

- A traversal/ searching algorithm for tree or graph data structures
- Recall level-order traversal on a tree from week 2
- 由点到面

对所有点的遍历：每个点放进/弹出container一次

Depth First Search and Breadth First Search on Explicit Graph

Q1 Graph Valid Tree

https://leetcode.com/problems/graph-valid-tree/

Recall: V = E + 1 for a tree

method 1:
V = E + 1 && all vertices are connected in a graph => The graph is a tree.

How do we know if all vertices are connected?
Given one random node in an undirected graph, all other nodes are reachable from this node.
In other words, we want to **traverse all nodes reachable from a random node selected in the graph,** and count the number of nodes that have traversed**.**

method2:
connected && no cycle => The graph is a tree.

Solution:

```
    public boolean validTree(int n, int[][] edges) {
        if (n - 1 != edges.length) {
            return false;
        }
        List<List<Integer>> graph = getGraph(n, edges);
        return isConnected(graph);
    }
    private List<List<Integer>> getGraph(int n, int[][] edges) {
        List<List<Integer>> list = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            list.add(new ArrayList());
        }
        for (int[] edge : edges) {
            int n1 = edge[0];
            int n2 = edge[1];
            list.get(n1).add(n2);
            list.get(n2).add(n1);
        }
        return list;
    }
    private boolean isConnected(List<List<Integer>> graph) {
        Set<Integer> set = new HashSet<>();
        dfs(graph, 0, set); // for dfs traversal
        //bfs(graph, 0, set); // for bfs traversal
        return set.size() == graph.size();
    }
    private void dfs(List<List<Integer>> graph, int node, Set<Integer> set)
{
        set.add(node);
        for (Integer nei : graph.get(node)){
            if (!set.contains(nei)){
                dfs(graph, nei, set);
            }
        }
    }
    private void bfs(List<List<Integer>> graph, int node, Set<Integer>
set){
        Queue<Integer> queue = new ArrayDeque<>();
        queue.offer(node);
        set.add(node);
        while (!queue.isEmpty()){
            Integer temp = queue.poll();
            for (Integer nei : graph.get(temp)){
                if (set.add(nei)){
                    queue.offer(nei);
                }
            }
        }
    }
```

```
    private boolean isTree(List<List<Integer>> graph) {
        Set<Integer> checked = new HashSet<>();
        Set<Integer> path = new HashSet<>();
        boolean cyclic = hasCycle(graph, 0, -1, checked, path);
        return  !cyclic && checked.size() == graph.size();
    }
    private boolean hasCycle(List<List<Integer>> graph, int cur, int prev,
Set<Integer> checked, Set<Integer> path){
        if (checked.contains(cur)){
            return false;
        }
        if (!path.add(cur)){
            return true;
        }
        for (Integer nei : graph.get(cur)){
            if (nei == prev){
                continue;
            }
            if (hasCycle(graph, nei, cur, checked, path)){
                return true;
            }
        }
        path.remove(cur);
        checked.add(cur);
        return false;
    }
    private boolean hasCycle2(List<List<Integer>> graph, int cur, int prev,
int[] color){
        if (color[cur] == 1){
            return false;
        }
        if (color[cur] == 2){
            return true;
        }
        for (Integer nei : graph.get(cur)){
            if (nei == prev){
                continue;
            }
            if (hasCycle(graph, nei, cur, color)){
                return true;
            }
        }
        color[cur] = 1;
        return false;
    }
```

Discussion One : Adjacency List vs Adjacency Matrix

Which one is better?

- Time Complexity
  - Adjacency Matrix: O(V^2)
  - Adjacency List: O(V + E)
- Space Complexity
  - Adjacency Matrix: O(V^2)
  - Adjacency List: O(V + E)

For a sparse graph, an adjacency list saves a lot of space and time.
For a dense graph, pretty much the same in terms of space and time.
Adjacency matrix has better cache locality, so R/W performance is usually better.

Q2 Clone Graph

https://leetcode.com/problems/clone-graph/

Deep copy vs Shallow copy

Key: construct a **one-one relationship** between each node in the original graph and each node in the deep copied graph, each node is copied precisely once.

Solution:

```
    public Node cloneGraph(Node node) {
        Map<Node, Node> map = new HashMap<>();
        return bfs(node, map);
    }
    private Node dfs(Node node, Map<Node, Node> map){
        if (node == null) {
            return null;
        }
        Node copiedNode = map.get(node);
        if (copiedNode != null){
```

```
            return copiedNode;
        }
        copiedNode = new Node(node.val);
        map.put(node, copiedNode);
        for (Node nei : node.neighbors){
            copiedNode.neighbors.add(dfs(nei, map));
        }
        return copiedNode;
    }
    private Node bfs(Node node, Map<Node, Node> map){
        if (node == null){
            return null;
        }
        Queue<Node> queue = new ArrayDeque<>();
        queue.add(node);
        map.put(node, new Node(node.val));
        while (!queue.isEmpty()){
            Node temp = queue.poll();
            Node copiedTemp = map.get(temp);
            for (Node nei : temp.neighbors){
                if (!map.containsKey(nei)){
                    queue.offer(nei);
                }
                map.putIfAbsent(nei, new Node(nei.val));
                copiedTemp.neighbors.add(map.get(nei));
            }
        }
        return map.get(node);
    }
```

Q3 Is Graph Bipartite

https://leetcode.com/problems/is-graph-bipartite/

For each node we traversed, it must be in a different set than its neighbors.

唯一的难点： find a contradiction: a node and some of its neighbors are in the same set.

```
    public boolean isBipartite(int[][] graph) {
        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < graph.length; i++) {
            if (map.containsKey(i)){
```

```
                continue;
            }
            map.put(i, 0);
            if (!bfs(graph, i, map)) {
                return false;
            }
        }
        return true;
    }
    private boolean dfs(int[][] graph, int i, Map<Integer, Integer> map) {
        int color = map.get(i);
        int[] neighbors = graph[i];
        for (int nei : neighbors) {
            Integer value = map.get(nei);
            if(value != null && value == color) {
                return false;
            }
            if (value == null) {
                map.put(nei, color ^ 1);
                if (!dfs(graph, nei, map)){
                    return false;
                };
            }
        }
        return true;
    }
    private boolean bfs(int[][] graph, int i, Map<Integer, Integer> map){
        Queue<Integer> queue = new ArrayDeque<>();
        queue.add(i);
        while (!queue.isEmpty()){
            Integer temp = queue.poll();
            int color = map.get(temp);
            for (int nei : graph[temp]){
                Integer neiColor = map.get(nei);
                if (neiColor == null){
                    map.put(nei, color ^ 1);
                    queue.offer(nei);
                } else if (neiColor == color) {
                    return false;
                }
            }
        }
        return true;
    }
```

Discussion Two: DFS && BFS: Which one is more efficient?

Time & Space complexity: both are the same
But which one is better? Totally depends on the topological structure of a graph.

Depth First Search and Breadth First Search on Implicit Graph

Implicit Graph: vertices and edges are **not explicitly** presented in the computer's memory.

难点：建图，什么是点，每个点的邻居是谁？

Q4 Number of Islands

https://leetcode.com/problems/number-of-islands/

Each island is a connected component. Can be deemed as a connected undirected graph.
Given a random vertex in a connected undirected graph, we are able to traverse all vertices in this graph.

vertex：each position whose value is 1

edge: its adjacent position: up, down, left, right

Solution:

```
    public int numIslands(char[][] grid) {
        int count = 0;
        for (int i = 0; i < grid.length; i++){
            for (int j = 0; j < grid[0].length; j++){
                if (grid[i][j] =='0'){
                    continue;
                }
                dfs(grid, i, j);
                count++;
            }
        }
        return count;
    }
    private static final int[][] DIRS = {{0, 1},{1, 0},{0, -1},{-1, 0}};
    private void dfs(char[][] grid, int i, int j){
        grid[i][j] = '0';
        for (int[] dir : DIRS){
            int row = i + dir[0];
            int col = j + dir[1];
            if (withinBound(row, col, grid) && grid[row][col] == '1'){
                dfs(grid, row, col);
            }
        }
    }
    private void bfs(char[][] grid, int i, int j){
        Queue<Cell> queue = new ArrayDeque<>();
        queue.add(new Cell(i, j));
        grid[i][j] = '0';
        while (!queue.isEmpty()){
            Cell temp = queue.poll();
            for (int[] dir : DIRS){
                int row = i + dir[0];
                int col = j + dir[1];
                if (withinBound(row, col, grid) && grid[row][col] == '1'){
                    grid[row][col] = '0';
                    queue.offer(new Cell(row, col));
                }
            }
        }
    }
    private boolean withinBound(int i, int j, char[][] grid){
        return i >= 0 && j >= 0 && i < grid.length && j < grid[0].length;
    }
    class Cell{
        int i;
        int j;
        Cell(int i, int j){
            this.i = i;
            this.j = j;
```

```
        }
    }
```

Q5. World Ladders

https://leetcode.com/problems/word-ladder/

vertex: each word
edge: if two words can be transformed from each other by changing precisely one letter, then they are adjacent.

如何知道k步可达？

- recall level order traversal bfs
- 开一个int: step
- 开一个HashMap存从origin到达我的最小step

如何efficiently的找到所有邻居？

- given word w, for each word v in dictionary, determine if they are adjacent by linearly scan both words from left to right O(size(dict) * (len(w) + len(v)))
- given word w, for each letter l in w, try replace l by each 26 alphabet and see if the changed one is in dictionary O(len(w) * 26 * len(w)) on average

Solution:

```
    public int ladderLength(String beginWord, String endWord, List<String>
wordList) {
        Set<String> set = getWordSet(wordList);
        Queue<String> queue = new ArrayDeque<>();
        Set<String> visited = new HashSet<>();
        queue.offer(beginWord);
```

```
        visited.add(beginWord);
        int seqNum = 1;
        while(!queue.isEmpty()){
            int size = queue.size();
            for (int i = 0; i < size; i++) {
                String temp = queue.poll();
                if (temp.equals(endWord)){
                    return seqNum;
                }
                List<String> neighbors = getNeighbors(temp, set);
                for (String nei : neighbors){
                    if (visited.add(nei)){
                        queue.offer(nei);
                    }
                }
            }
            seqNum++;
        }
        return 0;
    }
    private Set<String> getWordSet(List<String> wordList){
        Set<String> set = new HashSet<>();
        for (String s : wordList){
            set.add(s);
        }
        return set;
    }
    private List<String> getNeighbors(String temp, Set<String> set){
        List<String> result = new ArrayList<>();
        char[] word = temp.toCharArray();
        for (int i = 0; i < word.length; i++) {
            char original = word[i];
            for (char c = 'a'; c <= 'z'; c++){
                if (c == original){
                    continue;
                }
                word[i] = c;
                String cur = new String(word);
                if (set.contains(cur)){
                    result.add(cur);
                }
            }
            word[i] = original;
        }
        return result;
    }
```

Q6. Shortest Distance from all buildings

https://leetcode.com/problems/shortest-distance-from-all-buildings/

vertex: each reachable position in the given matrix edges: adjacent positions that are reachable: left, right, up, down

思路一：

for each empty land, find its shortest distance to each building, then sum up all distances

Is this efficient enough??

for each row O(m)

for each col O(n)

if zero : find shortest distance to each building; find sum

O(m* n)

Can we do better?
num of buildings normally <= num of empty lands

思路二：

for each building, find its shortest distance to each empty land and store them in a matrix.

大坑：

可能存在某个empty land不能到达所有building.

- 存一个visited, 一个unreachable matrix

二者时间复杂度一样，但具体哪个更优，需要在面试时和面试官讨论use case：

- num of buildings vs. num of empty lands??

Solution

```
    private static final int[][] DIRS = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
    public int shortestDistance(int[][] grid) {
        Queue<Position> queue = new ArrayDeque<>();
        int[][] distance = new int[grid.length][grid[0].length];
        boolean[][] unreachable = new boolean[grid.length][grid[0].length];
        for (int i = 0; i < grid.length; i++){
            for (int j = 0; j < grid[0].length; j++) {
                if (grid[i][j] == 1){
                    travel(queue, distance, grid, unreachable, i, j);
                }
            }
        }
        return getMinDistance(distance, unreachable);
    }
    private void travel(Queue<Position> queue, int[][] distance, int[][]
grid, boolean[][] unreachable,int k, int j){
        int step = 0;
        queue.offer(new Position(k, j));
        boolean[][] visited = new boolean[grid.length][grid[0].length];
        visited[k][j] = true;
        while(!queue.isEmpty()){
            int size = queue.size();
            for (int i = 0; i < size; i++){
                Position curPos = queue.poll();
                for (int[] dir : DIRS){
                    int row = curPos.row + dir[0];
                    int col = curPos.col + dir[1];
                    if (withinBound(grid, row, col) && canReach(visited,
grid, row, col)){
                        visited[row][col] = true;
                        distance[row][col] += step + 1;
                        queue.offer(new Position(row, col));
                    }
                }
            }
            step++;
        }
        findUnreachablePosition(visited, grid, unreachable);
```

```
    }
    private boolean withinBound(int[][] grid, int row, int col){
        return row >= 0 && col >= 0 && row < grid.length && col <
grid[0].length;
    }
    private boolean canReach(boolean[][] visited, int[][] grid, int row,
int col){
        return grid[row][col] != 2 && !visited[row][col] &&
grid[row][col] != 1;
    }
    private boolean findUnreachablePosition(boolean[][] visited, int[][]
grid, boolean[][] unreachable){
        for (int i = 0; i < grid.length; i++) {
            for (int j = 0; j < grid[0].length; j++){
                if (grid[i][j] == 0 && !visited[i][j]){
                    unreachable[i][j] = true;
                }
            }
        }
        return true;
    }
    private int getMinDistance(int[][] distance, boolean[][] unreachable){
        int globalMin = Integer.MAX_VALUE;
        for (int i = 0; i < distance.length; i++) {
            for (int j = 0; j < distance[0].length; j++) {
                if (unreachable[i][j]){
                    continue;
                }
                if (distance[i][j] != 0){
                    globalMin = Math.min(globalMin, distance[i][j]);
                }
            }
        }
        return globalMin == Integer.MAX_VALUE ? -1 : globalMin;
    }
    class Position{
        int row;
        int col;
        Position(int row, int col){
            this.row = row;
            this.col = col;
        }
    }
```

Discussion Three: Find shortest path: dfs or bfs?

How to find the shortest path from origin to destination using dfs?
Find all possible paths from origin to destination. Update globalMin.

所有点遍历 vs 所有路径的遍历

Why is dfs usually not as efficient as bfs in a lot of cases?

- To find all possible paths using dfs, each vertex in the graph is traversed at least once
- To find all possible paths using bfs, each vertex in the graph is traversed at most once

Discussion Four: Possible follow up question in an interview

- print the shortest path
- print all shortest paths

Summary

DFS

To traverse all nodes reachable from V:

- mark V as visited (hashSet, hashMap)
- for each neighbor nei:
    - if not visited: traverse all nodes reachable from nei
- pruning: find contradiction; find result

BFS

Data Structure: Queue
Initialize: put start node into the queue

For each step:
    expand a node
    generate all neighbors
        need deduplication: tree? graph?
termination condition: find target, find contradiction, queue is empty

Homework

- # of connected components in a graph
  https://leetcode.com/problems/number-of-connected-components-in-an-undirected-graph/

- number of closed islands
  https://leetcode.com/problems/number-of-closed-islands/

- walls and gate
  https://leetcode.com/problems/walls-and-gates/

- evaluate division
  https://leetcode.com/problems/evaluate-division/

- maze I

https://leetcode.com/problems/the-maze/

- optional: 815 bus route, 773 sliding puzzle