LeetCode Summary

deepwaterooo

January 26, 2015

Contents

Chapter 1

线性表

1.1 Array

1.1.1 Two Sum

1.1.2 Two Sum II - Input Array is Sorted

1.1.3 Two Sum III - Data Structure Design

1.1.4 Valid Palindrome

1.2 单链表：Single Linked List

Chapter 2

字符串：String

2.0.1 Implement strStr()

2.0.2 Reverse Words in a String

2.0.3 Reverse Words in a String II

2.0.4 String to Integer (atoi)

2.0.5 Valid Number

2.0.6 Longest Substring Without Repeating Characters

2.0.7 Longest Substring with At Most Two Distinct Characters

2.0.8 Longest Palindromic Substring

2.0.9 Missing Ranges

2.0.10 One Edit Distance

2.0.11 Read N Characters Given Read4

2.0.12 Read N Characters Given Read4 II - Call multiple times

Chapter 3

Linked List

3.1 Easier Maybe?

3.1.1 Merge Two Sorted List

3.1.2 Add Two Numbers

3.1.3 Swap Nodes in Pairs

3.1.4 Copy List With Random Pointers

Chapter 4

树: Binary Tree, Binary Search Tree

4.1 二叉树的遍历

4.1.1

4.2 二叉树的构建

4.2.1

4.3 二叉树查找

4.4 二叉树递归

Chapter 5

栈和队列

5.1 栈

5.1.1 Evaluate Reverse Polish Notation

5.1.2 Valid Parentheses

5.1.3

5.2 队列

5.2.1

5.3 Heap: Merge k sorted List

Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity. **Tags**: Divide and Conquer, Linked List, Heap

```
public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
    if (l1 == null) return l2;
    if (l2 == null) return l1;
    ListNode result;
    if (l1.val < l2.val) {
        result = l1;
        l1 = l1.next;
        result.next = null;
    } else {
        result = l2;
        l2 = l2.next;
        result.next = null;
    }
    ListNode curr = result;
    while (l1 != null && l2 != null) {
        if (l1.val < l2.val) {
            curr.next = l1;
            curr = curr.next;
            l1 = l1.next;
            curr.next = null;
        } else {
            curr.next = l2;
            curr = curr.next;
            l2 = l2.next;
            curr.next = null;
        }
    }
    if (l1 == null && l2 == null)
        return result;
    l1 = (l1 == null) ? l2 : l1;
    curr.next = l1;
    return result;
}
```

```java
public ListNode mergeKLists(List<ListNode> lists) {
    if (lists.size() == 0) return null;
    if (lists.size() == 1) return lists.get(0);
    if (lists.size() == 2) return mergeTwoLists(lists.get(0), lists.get(1));
    return mergeTwoLists((mergeKLists(lists.subList(0, lists.size() / 2))),
                         (mergeKLists(lists.subList(lists.size() / 2, lists.size()))));
}
```

Chapter 6

Backtracing and Recursion

6.1 排列：Permutation

6.1.1 Permutation

Given a collection of numbers, return all possible permutations.
For example,

```
[1,2,3] have the following permutations:
[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], and [3,2,1].
```

6.1.2 Permutation II

Given a collection of numbers that might contain duplicates, return all possible unique permutations.
For example,

```
[1,1,2] have the following unique permutations:
[1,1,2], [1,2,1], and [2,1,1].
```

6.1.3 Permutation Sequence

The set [1,2,3,⋯,n] contains a total of n! unique permutations.
By listing and labeling all of the permutations in order, We get the following sequence (ie, for n = 3):

```
"123"
"132"
"213"
"231"
"312"
"321"
```

Given n and k, return the kth permutation sequence.
Note: Given n will be between 1 and 9 inclusive.

6.2 组合：Combination

6.2.1 Combinationas

Given two integers n and k, return all possible combinations of k numbers out of 1 ⋯ n.
For example,
If n = 4 and k = 2, a solution is:

```
[
  [2,4],
  [3,4],
  [2,3],
  [1,2],
  [1,3],
  [1,4],
]
```

6.2.2 Combination Sum

Given a set of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

The same repeated number may be chosen from C unlimited number of times.

Note:

- All numbers (including target) will be positive integers.
- Elements in a combination (a1, a2, ⋯, ak) must be in non-descending order. (ie, a1 ⩽ a2 ⩽⋯⩽ ak).
- The solution set must not contain duplicate combinations.

For example, given candidate set 2,3,6,7 and target 7,

A solution set is:

```
[7]
[2, 2, 3]
```

6.2.3 Combination Sum II

Given a collection of candidate numbers (C) and a target number (T), find all unique combinations in C where the candidate numbers sums to T.

Each number in C may only be used once in the combination.

Note:

- All numbers (including target) will be positive integers.
- Elements in a combination (a1, a2, ⋯, ak) must be in non-descending order. (ie, a1 ⩽ a2 ⩽⋯⩽ ak).
- The solution set must not contain duplicate combinations.

For example, given candidate set 10,1,2,7,6,1,5 and target 8,

A solution set is:

```
[1, 7]
[1, 2, 5]
[2, 6]
[1, 1, 6]
```

6.3 Subsets

6.3.1 Subsets: Bit Manipulation

Given a set of distinct integers, S, return all possible subsets.

Note:

- Elements in a subset must be in non-descending order.
- The solution set must not contain duplicate subsets.

For example, If S = [1,2,3], a solution is:

```
[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]
```

Tags: Array Backtracking, Bit Manipulation

6.3.2 Subsets II

Given a collection of integers that might contain duplicates, S, return all possible subsets.
Note:

- Elements in a subset must be in non-descending order.
- The solution set must not contain duplicate subsets.

For example, If S = [1,2,2], a solution is:

```
[
  [2],
  [1],
  [1,2,2],
  [2,2],
  [1,2],
  []
]
```

6.4 with Recursion

6.4.1 Letter Combinationas of Phone Number

Given a digit string, return all possible letter combinations that the number could represent.
A mapping of digit to letters (just like on the telephone buttons) is given below.

```
Input:Digit string "23"
Output: ["ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"].
```

Note:
Although the above answer is in lexicographical order, your answer could be in any order you want.

6.4.2 Restore IP Addresses

Given a string containing only digits, restore it by returning all possible valid IP address combinations.
For example:

```
Given "25525511135",
return ["255.255.11.135", "255.255.111.35"]. (Order does not matter)
```

6.4.3 Generate Parentheses

Given n pairs of parentheses, write a function to generate all combinations of well-formed parentheses.
For example, given n = 3, a solution set is:

```
"((()))", "(()())", "(())()", "()(())", "()()()"
```

6.4.4 Gray Code

The gray code is a binary numeral system where two successive values differ in only one bit.
Given a non-negative integer n representing the total number of bits in the code, print the sequence of gray code. A gray code sequence must begin with 0.
For example, given n = 2, return [0,1,3,2]. Its gray code sequence is:

```
00 - 0
01 - 1
11 - 3
10 - 2
```

Note:

- For a given n, a gray code sequence is not uniquely defined.
- For example, [0,2,3,1] is also a valid gray code sequence according to the above definition.
- For now, the judge is able to judge based on one instance of gray code sequence. Sorry about that.

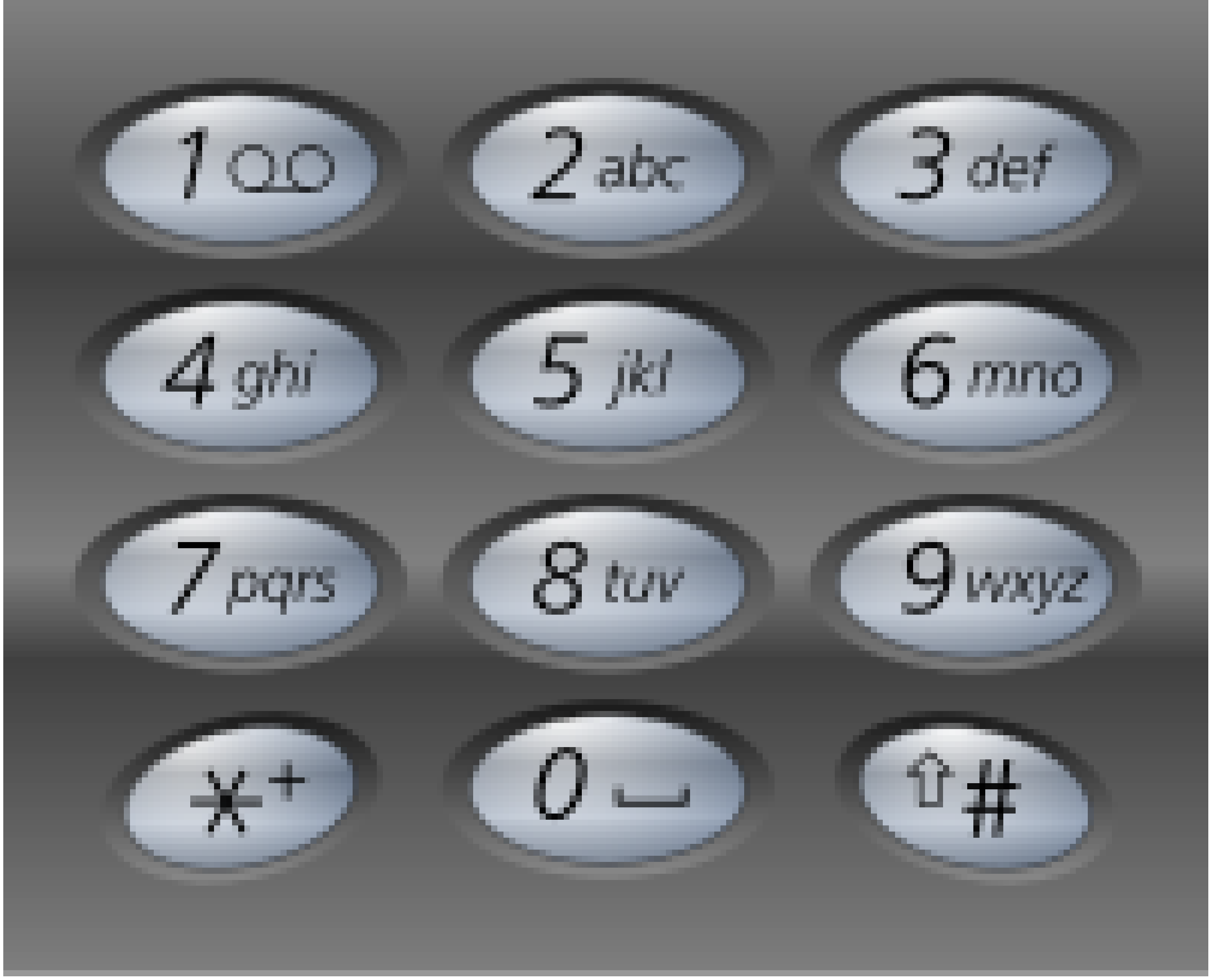


Figure 6.1: Letter Combinationas of Phone Number

6.4.5 Word Search

Given a 2D board and a word, find if the word exists in the grid.

The word can be constructed from letters of sequentially adjacent cell, where ”adjacent” cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once.

For example,

```
Given board =
[
  ["ABCE"],
  ["SFCS"],
  ["ADEE"]
]
word = "ABCCED", -> returns true,
word = "SEE", -> returns true,
word = "ABCB", -> returns false.
```

6.4.6 Palindrome Partitioning

Given a string s, partition s such that every substring of the partition is a palindrome.

Return all possible palindrome partitioning of s.

For example,

```
given s = "aab",
Return
```

```
[
  ["aa","b"],
  ["a","a","b"]
]
```

6.4.7 N-Queens

The n-queens puzzle is the problem of placing n queens on an n×n chessboard such that no two queens attack each other.

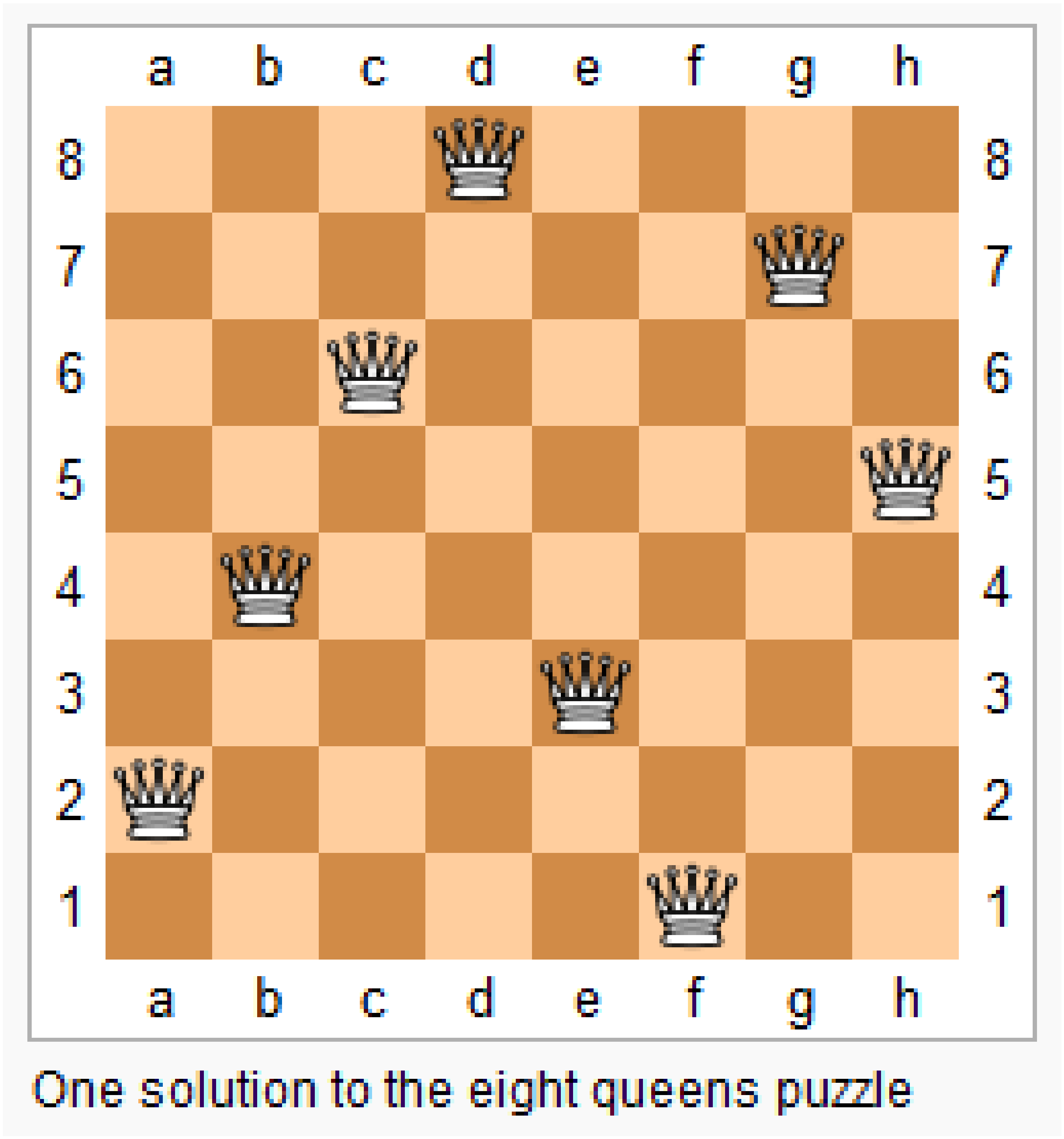


Figure 6.2: N-Queens

Given an integer n, return all distinct solutions to the n-queens puzzle.

Each solution contains a distinct board configuration of the n-queens' placement, where 'Q' and '.' both indicate a queen and an empty space respectively.

For example,

There exist two distinct solutions to the 4-queens puzzle:

```
[
 [".Q..",  // Solution 1
  "...Q",
  "Q...",
  "..Q."],

 ["..Q.",  // Solution 2
  "Q...",
  "...Q",
  ".Q.."]
]
```

6.4.8 N-Queens II

Follow up for N-Queens problem.
Now, instead outputting board configurations, return the total number of distinct solutions.

6.4.9 Sudoku Solver

Write a program to solve a Sudoku puzzle by filling the empty cells.
Empty cells are indicated by the character '.'.
You may assume that there will be only one unique solution.
A sudoku puzzle···
···and its solution numbers marked in red.
Tags: Backtracking, Hash Table

6.4.10 Regular Expression Matching

Implement regular expression matching with support for '.' and '*'.

- '.' Matches any single character.

- '*' Matches zero or more of the preceding element.

The matching should cover the entire input string (not partial).
The function prototype should be:
bool isMatch(const char *s, const char *p)
Some examples:

```
isMatch("aa","a") → false
isMatch("aa","aa") → true
isMatch("aaa","aa") → false
isMatch("aa", "a*") → true
isMatch("aa", ".*") → true
isMatch("ab", ".*") → true
isMatch("aab", "c*a*b") → true
```

Tags: Dynamic Programming, Backtracking, String

6.4.11 Wild Card Matching

Implement wildcard pattern matching with support for '?' and '*'.

- '?' Matches any single character.

- '*' Matches any sequence of characters (including the empty sequence).

The matching should cover the entire input string (not partial).
The function prototype should be:
bool isMatch(const char *s, const char *p)
Some examples:

```
isMatch("aa","a") → false
isMatch("aa","aa") → true
isMatch("aaa","aa") → false
isMatch("aa", "*") → true
isMatch("aa", "a*") → true
isMatch("ab", "?*") → true
isMatch("aab", "c*a*b") → false
```

Tags: Dynamic Programming, Backtracking, Greedy, String

6.4.12 Word Break II

Given a string s and a dictionary of words dict, add spaces in s to construct a sentence where each word is a valid dictionary word.

Return all such possible sentences.

For example, given

```
s = "catsanddog",
dict = ["cat", "cats", "and", "sand", "dog"].

A solution is ["cats and dog", "cat sand dog"].
```

Tags: Dynamic Programming Backtracking

6.4.13 Word Ladder II

Given two words (start and end), and a dictionary, find all shortest transformation sequence(s) from start to end, such that:

- Only one letter can be changed at a time
- Each intermediate word must exist in the dictionary

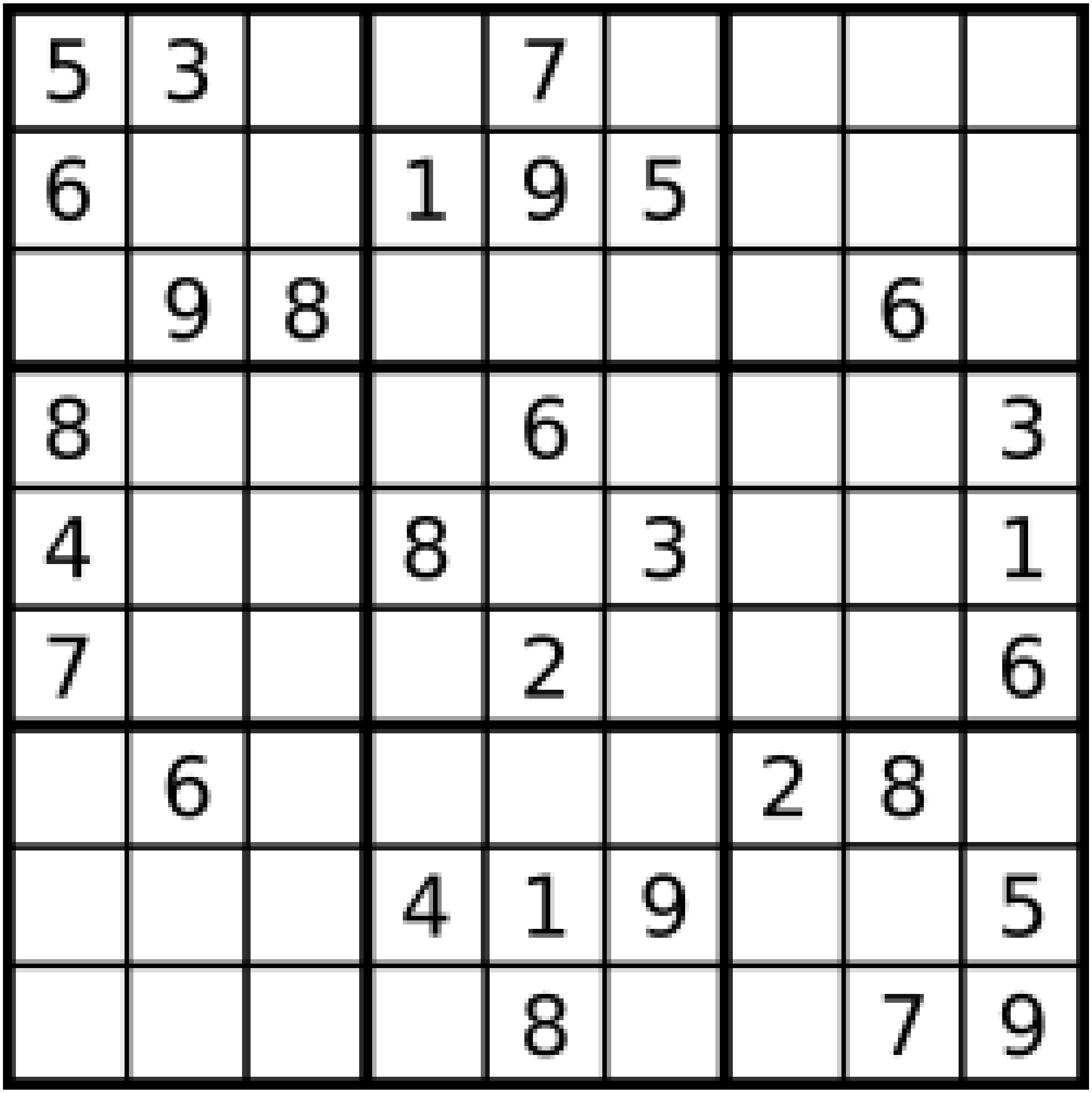
For example,

```
Given:
start = "hit"
end = "cog"
dict = ["hot","dot","dog","lot","log"]
Return
  [
    ["hit","hot","dot","dog","cog"],
    ["hit","hot","lot","log","cog"]
  ]
```

Note:

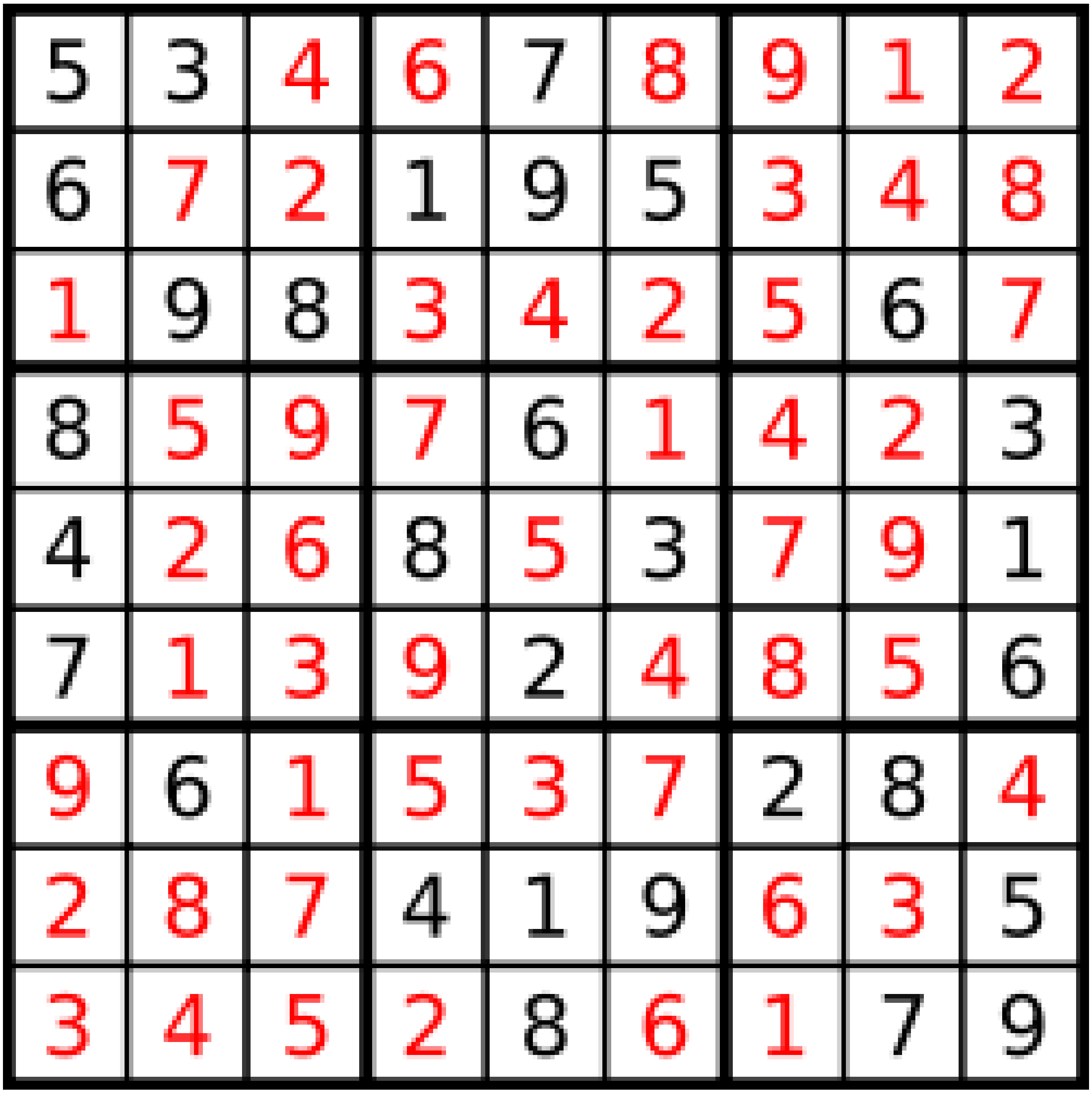
- All words have the same length.
- All words contain only lowercase alphabetic characters.

Tags: Array, Backtracking, Breadth-first Search, String



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Figure 6.3: Sudoku Solver 1



5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Figure 6.4: Sudoku Solver 2

Chapter 7

Bit Manipulation

7.0.14 Subsets: Bit Manipulation

Given a set of distinct integers, S, return all possible subsets.
Note:

- Elements in a subset must be in non-descending order.
- The solution set must not contain duplicate subsets.

For example, If S = [1,2,3], a solution is:

```
[
  [3],
  [1],
  [2],
  [1,2,3],
  [1,3],
  [2,3],
  [1,2],
  []
]
```

Tags: Array Backtracking, Bit Manipulation

7.0.15 Single Number

7.0.16 Single Number II

7.0.17 Majority Element

Chapter 8

排序

Chapter 9

查找: Binary Search

9.1 Easier Maybe?

9.1.1 Search Insert Position

9.1.2 Find Minimum in Rotated Sorted Array

9.1.3 Find Minimum in Rotated Sorted Array II - with Duplicates

9.1.4

Chapter 10

暴力枚举法

10.0.5

Chapter 11

广度优先搜索: Breadth First Search

11.0.6 Word Ladder

Given two words (start and end), and a dictionary, find the length of shortest transformation sequence from start to end, such that:

- Only one letter can be changed at a time
- Each intermediate word must exist in the dictionary

For example,
Given:

```
start = "hit"
end = "cog"
dict = ["hot","dot","dog","lot","log"]
As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog",
return its length 5.
```

Note:

- Return 0 if there is no such transformation sequence.
- All words have the same length.
- All words contain only lowercase alphabetic characters.

11.0.7

Chapter 12

深度优先搜索: Depth First Search

Chapter 13

分治法: Divide and Conqure

13.0.8 Maximum Subarray

13.0.9 Merge K sorted List

13.0.10 Median of Two Sorted Array

13.0.11 Majority Element

13.0.12

13.0.13

13.0.14

Chapter 14

贪心法: Greedy Search

14.0.15 Gas Station

14.0.16 Best Time to Buy and Sell Stock II

14.0.17 Jump Game

14.0.18 Jump Game II

14.0.19 Candy

14.0.20 Wildcard Matching

14.0.21

Chapter 15

动态归划：Dynamic Programming

15.0.22 Climbing Stairs

15.0.23 Unique Path

15.0.24 Unique Path II

15.0.25 Maximum Sum Subarray

15.0.26 Maximum Product Subarray

15.0.27 Coins in a Line

15.0.28 Word Break

Given a string s and a dictionary of words dict, determine if s can be segmented into a space-separated sequence of one or more dictionary words.

For example, given

```
s = "leetcode",
dict = ["leet", "code"].
Return true because "leetcode" can be segmented as "leet code".
```

15.0.29 Palindrome Partitioning II

Given a string s, partition s such that every substring of the partition is a palindrome.

Return the minimum cuts needed for a palindrome partitioning of s.

For example,

```
given s = "aab",
Return 1 since the palindrome partitioning ["aa","b"] could be produced using 1 cut.
```

Chapter 16

Two Pointers and Sliding Window

16.0.30

Chapter 17

图: Graphics

17.1 Clone Graph

Clone an undirected graph. Each node in the graph contains a label and a list of its neighbors.

OJ's undirected graph serialization:

Nodes are labeled uniquely.

We use # as a separator for each node, and , as a separator for node label and each neighbor of the node.

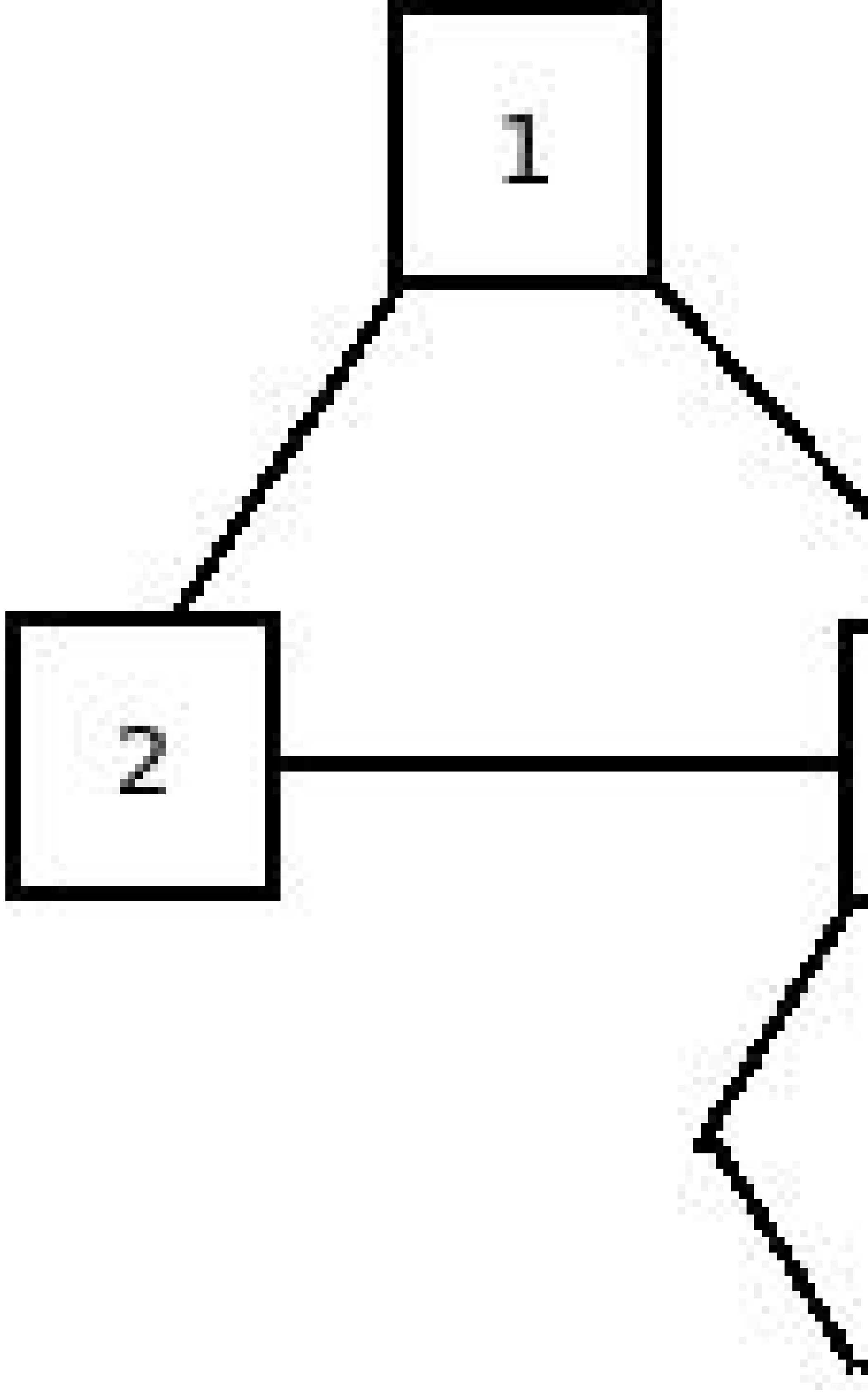
As an example, consider the serialized graph {0,1,2#1,2#2,2}.

The graph has a total of three nodes, and therefore contains three parts as separated by #.

1. First node is labeled as 0. Connect node 0 to both nodes 1 and 2.

2. Second node is labeled as 1. Connect node 1 to node 2.

3. Third node is labeled as 2. Connect node 2 to node 2 (itself), thus forming a self-cycle.



Visually, the graph looks like the following:

17.1.1 分析：广度优先遍历或深度优先遍历都可以

17.1.2 DFS：时间复杂度 O(n), 空间复杂度 O(n)

```
/**
 * Definition for undirected graph.
 * class UndirectedGraphNode {
 *     int label;
 *     List<UndirectedGraphNode> neighbors;
 *     UndirectedGraphNode(int x) { label = x; neighbors = new ArrayList<UndirectedGraphNode>(); }
 * };
 */
```

```
public UndirectedGraphNode cloneGraph(UndirectedGraphNode node) {
    if (node == null) return null;
    UndirectedGraphNode res = new UndirectedGraphNode(node.label);  // result head
    if (node.neighbors == null || node.neighbors.size() == 0) return res;

    Map<UndirectedGraphNode, UndirectedGraphNode> map = new HashMap<UndirectedGraphNode, UndirectedGrap
    Queue<UndirectedGraphNode> q = new LinkedList<UndirectedGraphNode>();
    q.add(node);                      // added first node, need add its all Neighbors as well
    map.put(node, res);

    List<UndirectedGraphNode> curNbr = new ArrayList<UndirectedGraphNode>();
    UndirectedGraphNode curr = null;
    while (!q.isEmpty()) {
        curr = q.poll();
        curNbr = curr.neighbors;                   // ori
        for (UndirectedGraphNode aNbr : curNbr) {  // for build connection among copies
            if (!map.containsKey(aNbr)) {
                UndirectedGraphNode acpNbr = new UndirectedGraphNode(aNbr.label);
                map.put(aNbr, acpNbr);
                map.get(curr).neighbors.add(acpNbr);
                q.add(aNbr);
            } else
                map.get(curr).neighbors.add(map.get(aNbr));
        }
    }
    return res;
}
```

17.1.3 BFS:

17.2 Word Ladder, Word Ladder II: Backtracing

17.2.1 Word Ladder

Given two words (start and end), and a dictionary, find the length of shortest transformation sequence from start to end, such that:

1. Only one letter can be changed at a time
2. Each intermediate word must exist in the dictionary

For example,
Given: start = "hit", end = "cog"
dict = ["hot","dot","dog","lot","log"]
As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog",
return its length 5.
Note:

- Return 0 if there is no such transformation sequence.
- All words have the same length.
- All words contain only lowercase alphabetic characters.

17.2.2 Word Ladder II

Given two words (start and end), and a dictionary, find all shortest transformation sequence(s) from start to end, such that:

1. Only one letter can be changed at a time
2. Each intermediate word must exist in the dictionary

For example,
Given:
start = "hit", end = "cog"
dict = ["hot","dot","dog","lot","log"]
Return

```
[
  ["hit","hot","dot","dog","cog"],
  ["hit","hot","lot","log","cog"]
]
```

Note:

- All words have the same length.
- All words contain only lowercase alphabetic characters.

17.3 Check whether the graph is bigraph

17.4 Topological Sort

Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge uv, vertex u comes before v in the ordering. Topological Sorting for a graph is not possible if the graph is not a DAG.

For example, a topological sorting of the following graph is "5 4 2 3 1 0 ". There can be more than one topological sorting for a graph. For example, another topological sorting of the following graph is "4 5 2 3 1 0 ". The first vertex in topological sorting is always a vertex with in-degree as 0 (a vertex with no in-coming edges).

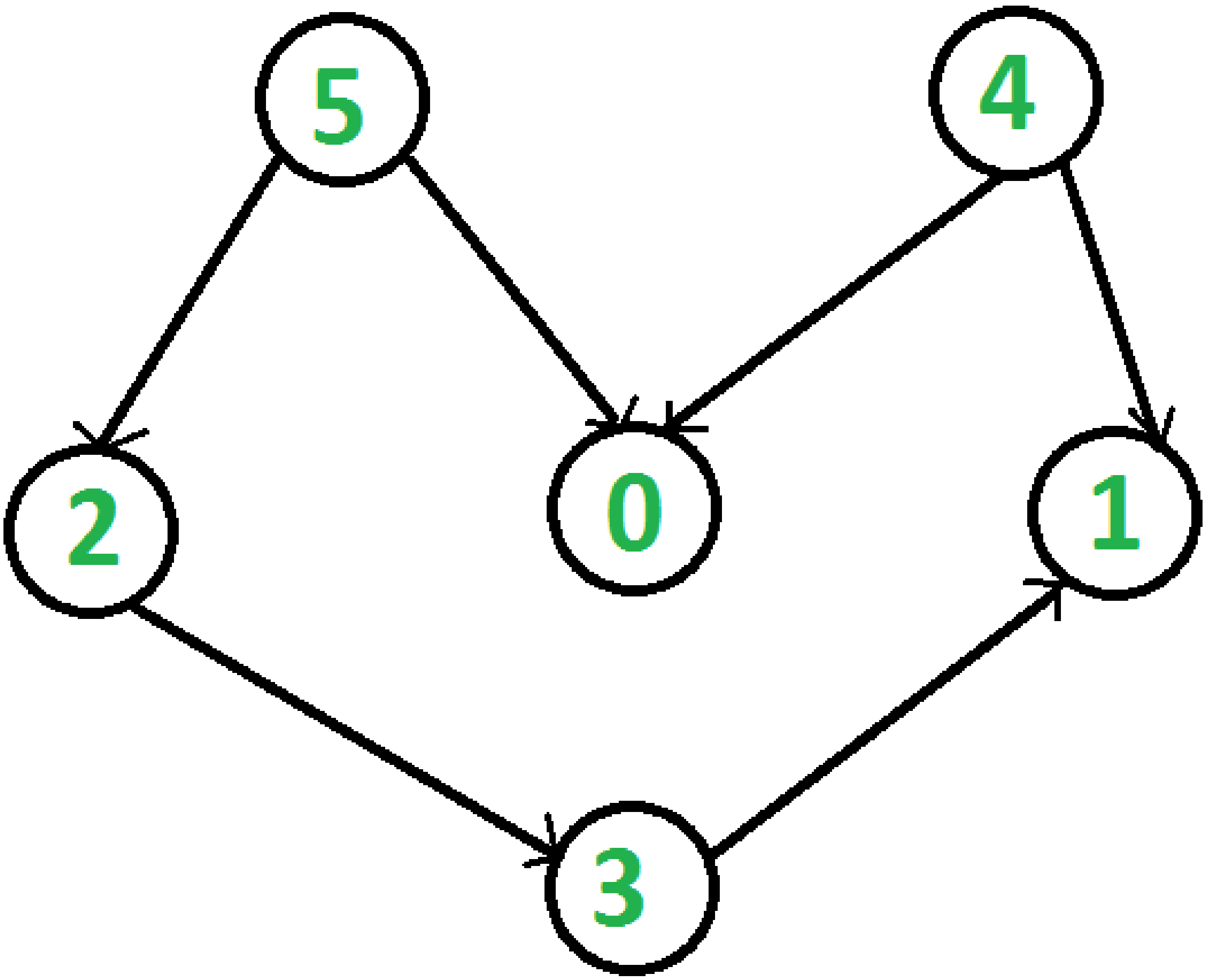


Figure 17.1: Topological Sorting

17.4.1 Topological Sorting vs Depth First Traversal (DFS):

In DFS, we print a vertex and then recursively call DFS for its adjacent vertices. In topological sorting, we need to print a vertex before its adjacent vertices. For example, in the given graph, the vertex ‘5’ should be printed before vertex ‘0’, but unlike DFS, the vertex ‘4’ should also be printed before vertex ‘0’. So Topological sorting is different from DFS. For example, a DFS of the above graph is “5 2 3 1 0 4 ″, but it is not a topological sorting.

17.4.2 Algorithm to find Topological Sorting:

We recommend to first see implementation of DFS here. We can modify DFS to find Topological Sorting of a graph. In DFS, we start from a vertex, we first print it and then recursively call DFS for its adjacent vertices. In topological sorting, we use a temporary stack. We don’t print the vertex immediately, we first recursively call topological sorting for all its adjacent vertices, then push it to a stack. Finally, print contents of stack. Note that a vertex is pushed to stack only when all of its adjacent vertices (and their adjacent vertices and so on) are already in stack.

Chapter 18

Data Structure

18.1 Min Stack

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

- push(x) – Push element x onto stack.
- pop() – Removes the element on top of the stack.
- top() – Get the top element.
- getMin() – Retrieve the minimum element in the stack. **Tags:** Stack Data Structure

```
public static class MinStack {
    Stack<Integer> stack = new Stack<Integer>();
    Stack<Integer> minStack = new Stack<Integer>();
    public void push(int x) {
        stack.push(x);
        if (minStack.isEmpty() || x <= minStack.peek()) {
            minStack.push(x);
        }
    }

    public void pop() { // java boxing & unboxing, container, object specific methods
        if (stack.peek().intValue() == minStack.peek().intValue())
            minStack.pop();
        stack.pop();
    }

    public int top() {
        return stack.peek();
    }

    public int getMin() {
        if (!minStack.isEmpty()) return minStack.peek();
        else return -1;
    }
}
```

18.2 LRU Cache

Design and implement a data structure for Least Recently Used (LRU) cache. It should support the following operations: get and set.

- get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.
- set(key, value) - Set or insert the value if the key is not already present. When the cache reached its capacity, it should invalidate the least recently used item before inserting a new item.

```java
public static class LRUCache {
    public class Node {
        int key;
        int value;
        Node prev;
        Node next;
        public Node(int x, int y){
            key = x;
            value = y;
        }
    }

    private HashMap<Integer, Node> hash;
    private int cap;
    private int number;
    Node head;
    Node tail;
    public LRUCache(int capacity) {
        cap = capacity;
        number = 0;
        head = new Node(-1, -1);
        head.prev = null;
        head.next = null;
        tail = head;
        hash = new HashMap<Integer, Node>(capacity); // so I can restrict a size !!
    }

    public int get(int key) {
        Node res = hash.get(new Integer(key)); // don't understand here
        if (res != null) {
            refresh(res);   // update usage frequence
            return res.value;
        } else return -1;
        /*
          if (hash.containsKey(key)) {
          //Node res = hash.get(new Integer(key)); // don't understand here
          Node res = hash.get(key);
          refresh(res);   // update usage frequence
          return res.value;
          } else {
          return -1;
          }
        */
    }

    // so still, must maintain a doubly-linked list to order usage frequency
    public  void refresh(Node tmp) {
        if (tmp == head.next) return; // it's head already

        Node temp = head.next;  // head node in the hash
        Node prevNode = tmp.prev;
        Node nextNode = tmp.next;
        // set to be most recently used~~ move the tmp node to be head.next, connections
        head.next = tmp;
        tmp.prev = head;
        tmp.next = temp;
        temp.prev = tmp;
        prevNode.next = nextNode;
        if (nextNode != null)
            nextNode.prev = prevNode;
        else tail = prevNode;   // remember tail as well
    }

    public void set(int key, int value) {
        Node res = hash.get(new Integer(key));
        if (res != null) {
            refresh(res);
            res.value = value;
        } else {
            //if (!hash.containsKey(key)) { // another way of detecting existing
            Node prevNode = new Node(key, value);
            Node temp;
            if (number == cap) { // remove tail;
                temp = tail.prev;
                hash.remove(tail.key);      // remember to remove from hash too !!!
                if (temp != null) {
                    temp.next = null;
                }
                tail.prev = null;
                tail = temp;
                number--;
            }
            // add to tail first
            tail.next = prevNode;
            prevNode.prev = tail;
            tail = prevNode;
            refresh(prevNode);
            hash.put(key, prevNode);
            number++;  // count node numbers
        }
    }
}
```

18.3 Two Sum III - data Structure Design

Design and implement a TwoSum class. It should support the following operations: add and find.

- add - Add the number to an internal data structure.
- find - Find if there exists any pair of numbers which sum is equal to the value.

For example,

```
add(1); add(3); add(5);
find(4) -> true
find(7) -> false
```

Tags: Hash Table, Data Structure

Chapter 19

细节实现题

Chapter 20

Math

20.1 Easy

20.1.1 Reverse Integer

20.1.2 Plus One

20.1.3 Palindrome Number

20.1.4 Next Permutation: Math

Implement next permutation, which rearranges numbers into the lexicographically next greater permutation of numbers.

If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

The replacement must be in-place, do not allocate extra memory.

Here are some examples. Inputs are in the left-hand column and its corresponding outputs are in the right-hand column.

```
1,2,3 → 1,3,2
3,2,1 → 1,2,3
1,1,5 → 1,5,1
```

20.1.5

20.2