Flexport Coding 面试

Board Game

一个棋盘，啥也不说。两边对手，没有block

上面对手只能往下走，往斜对角走。

下面对手只能往上走，斜对角走。

题解：

1.getAllNextStep(char turn, char[][] board)

2.canMove(Position current, Position target)

3.move(Position current, Position target).

接口文档

1、getAllNextStep(boolean isNextPlayer)

获取黑棋或者白棋的当前状态所有可能可以走的步骤。

输入：布尔类型 true表示白棋子在右下角 false表示黑棋左上角

输出：List<int[]> allSteps list结构中包括所有的可能走的步骤

2、canMove(int[] current, int[] target, boolean isNextPlayer)

判断当前位置是否能够到达目标地址位置

输入：当前位置和目标位置， 是黑棋还是白棋

输出：bolean 布尔值表示是否能够到达

3、move(int[] current, int[] target, boolean isNextPlayer)

移动到当前位置

输入：当前位置和目标位置， 是黑棋还是白棋

输出：无

例子：

board 当前状态

```json
1 0 0 0
0 1 0 0
0 0 2 0
0 0 0 2
```

1、getAllNextStep(false)

output: [[2,0]]

2、canMove(...)

intput: current [1,1] target[3,1]

output: true

3、move()

intput: current [1,1] target[3,1]

output:

board:

JSON 复制代码

```
1 0 0 0
0 1 0 0
1 0 2 0
0 1 0 2
```

Java 复制代码

```java
public class BoardGame {

    private char[][] board;
    private char[][] tmpBoard;

    public BoardGame(){

    }

    public void move(int[] current, int[] target, boolean isNextPlayer){
        if (canMove(current,target,isNextPlayer)){
            if (isNextPlayer){
                tmpBoard[target[0]][target[1]] = '2';
            }else {
                tmpBoard[target[0]][target[1]] = '1';
            }
            board = tmpBoard.clone();
        }else {
            System.out.println("Target position couldn't arrival!");
        }
    }

    public boolean canMove(int[] current, int[] target, boolean isNextPlayer){
          tmpBoard = board.clone();
          int x = current[0];
          int y = current[1];
          int targetX = target[0];
          int targetY = target[1];
        return moveNextStep(tmpBoard,x,y,targetX,targetY,isNextPlayer);
    }

    private boolean moveNextStep(char[][] newBoard,
                                 int x,int y,
                                 int targetX,int targetY,
                                 boolean isNextPlayer){
        // 走出范围
        // ||白棋：已经走到x坐标小于目标坐标x了说明无法到达
        // ||黑棋：已经走到x坐标大于目标坐标x了说明无法到达
        if (x==targetX&&y==targetY){
            return true;
        }
        if (!inAreas(x,y)||
                (isNextPlayer&&(targetX>x))||
                (!isNextPlayer&&(targetX<x))){
```

```java
            return false;
        }

        boolean ans = false;
        char tmp = newBoard[x][y];
        if (isNextPlayer){
            // 白旗走法
            if (newBoard[x][y]!='1'){
                newBoard[x][y] = '2';
                ans = moveNextStep(newBoard,x-1,y-1,targetX,targetY,isNextPlayer)||
                        moveNextStep(newBoard,x-1,y+1,targetX,targetY,isNextPlayer);
                if (!ans){
                    newBoard[x][y] = tmp;
                }
            }

        }else {
            // 黑旗走法
            if (newBoard[x][y]!='2'){
                newBoard[x][y] = '1';
                ans = moveNextStep(newBoard,x+1,y+1,targetX,targetY,isNextPlayer)||
                        moveNextStep(newBoard,x+1,y-1,targetX,targetY,isNextPlayer);
                if (!ans){
                    newBoard[x][y] = tmp;
                }
            }
        }
        return ans;
    }


    /**
     *
     * @param isNextPlayer 那个player的可能要走的steps
     * @return
     */
    public List<int[]> getAllNextStep(boolean isNextPlayer){
        List<int[]> allNextStep = new ArrayList<>();
        if (isNextPlayer){
            getNextStep(board.length-1, board.length-1, allNextStep,isNextPlayer);
        }else {
            getNextStep(0,0, allNextStep,isNextPlayer);
        }
        return allNextStep;
    }

    private void getNextStep(int i,int j,List<int[]> steps,boolean isNextPlayer){
        if (!inAreas(i,j)){
            return;
        }
        if (board[i][j]=='0'){
            steps.add(new int[]{i,j});
            return;
        }
        if (isNextPlayer){
            // 白旗走法
            if (board[i][j]=='2'){
                getNextStep(i-1,j-1,steps,isNextPlayer);
                getNextStep(i-1,j+1,steps,isNextPlayer);
            }

        }else {
            // 黑旗走法
            if (board[i][j]=='1'){
                getNextStep(i+1,j+1,steps,isNextPlayer);
                getNextStep(i+1,j-1,steps,isNextPlayer);
            }
        }
    }

    private boolean inAreas(int i,int j){
        return 0<=i&&i< board.length&&
                0<=j&&j<board.length;
    }

    public static void printArr(char[][] board){
        StringBuilder ans = new StringBuilder();
        ans.append("[");
        for (int i=0;i<board.length;i++){

            for (int j=0;j<board.length;j++){
                ans.append(board[i][j]);
                if (j==board.length-1){
                    ans.append(',');
                };
            }
        }
        ans.append("]");
        System.out.println(new String(ans));
    }

    public static void main(String[] args) {
        BoardGame bg = new BoardGame();
        int n = 4;
        bg.board = new char[][]{
                {'1','0','0','0'},
                {'0','1','0','0'},
                {'0','0','2','0'},
                {'0','0','0','2'}
        };
        List<int[]> black = bg.getAllNextStep(false);
        System.out.println(JSONObject.toJSONString(black));
        int[] current = {1,1};
        int[] target = {3,1};
        boolean ans = bg.canMove(current,target,false);
        System.out.println(ans);

        bg.move(current,target,false);
        printArr(bg.board);
    }
}
```

Token Card Games

Coding Card game, step by step. Money is represented in number of each color: black, blue, green, green, white. A card has properties of how much it requires by it. Implement a canPurchase() method to decide whether a certain amount of money can buy a card.Implement the purchase method, which should update the money and cards owned. Discount with card owned. Each card has a property of color. If you own cards of a color. Next time purchase of another card, the price of this color can be discounted by the number of card you owned. Update the canPurchase and purchase method.

Implement canPurchase and purchase

Java 复制代码

```java
/**
 * 1、使用map定义key是颜色，value是所需要的数量，然后对比card和player的对应颜色的数量是否一致问题
 * 2、定义Tokens结构，从结构里面获取各个颜色的数量对比是否一致
 * @Author: Twiss
 * @Date: 2022/8/20 12:14 下午
 */
public class TokensCardGames {

    private Map<String,Integer> card;
    private Map<String,Integer> player;
    private Map<String,Integer> playerCards;

    public TokensCardGames(){
        this.card = new HashMap<>();
        this.player = new HashMap<>();
        this.playerCards = new HashMap<>();
    }

    public boolean canPurchase() {
        Map<String,Integer> resultCard = new HashMap<>();
        for (String tokenCard : player.keySet()) {
            if (!card.containsKey(tokenCard)) {
                return false;
            } else {
                resultCard.put(tokenCard, card.get(tokenCard) - player.get(tokenCard));
            }
        }
        for (String tokenCard : resultCard.keySet()) {
            if (resultCard.get(tokenCard) > 0) {
                return false;
            }
        }
        return true;
    }

    public void purchase(Map<String,Map<String,Integer>> cards, Map<String,Integer> player) {
        for (String cardName:cards.keySet()){
            card = cards.get(cardName);
            if (canPurchase()){
                for (String tokenCard:player.keySet()){
                    player.put(tokenCard,player.get(tokenCard)-card.get(tokenCard));
                }
                playerCards.put(cardName,playerCards.getOrDefault(cardName,0)+1);
            }
        }
    }

    public static void main(String[] args) {
        TokensCardGames tg = new TokensCardGames();
        Map<String,Map<String,Integer>> cards = new HashMap<>();
        Map<String,Integer> cardA = new HashMap<>();
        cardA.put("WH",2);
        cardA.put("BL",1);
        cardA.put("B",4);
        cards.put("cardA",cardA);

        Map<String,Integer> player = new HashMap<>();
        player.put("WH",4);
        player.put("BL",2);
        player.put("B",5);
        tg.player = player;
        for (String cardName:cards.keySet()){
            tg.card = cards.get(cardName);
            boolean valid = tg.canPurchase();
            System.out.println(valid);
        }

        tg.purchase(cards, player);
        System.out.println(JSONObject.toJSONString(tg.playerCards));
    }
}
```

Discount with card owned

Discount with card owned. Each card has a property of color. If you own cards of a

color. Next time you purchase another card, the price of this color can be discountedby the number of cards you owned. Update the canPurchase and purchase method. (比如手里有三张 Red Card，下一张待购买的卡片 cost 需要 N 个 Red Money，实际购买时 只需要支付 N-3 个 Red Money 就行)

Java 复制代码

```java
public class TokensCardGamesII {

    private Map<String,Integer> card;
    private Map<String,Integer> player;
    private Map<String,Integer> playerCards;

    public TokensCardGamesII(){
        this.card = new HashMap<>();
        this.player = new HashMap<>();
        this.playerCards = new HashMap<>();
    }

    public boolean canPurchase() {
        Map<String,Integer> resultCard = new HashMap<>();
        for (String tokenCard : player.keySet()) {
            if (card.containsKey(tokenCard)) {

                resultCard.put(tokenCard, card.get(tokenCard)
                        - player.get(tokenCard)
                        -playerCards.getOrDefault(tokenCard,0));
                for (String resultCardName : resultCard.keySet()) {
                    if (resultCard.get(resultCardName) <= 0) {
                        return true;
                    }
                }
            }
        }
        return false;
    }

    public void purchase(Map<String,Map<String,Integer>> cards, Map<String
,Integer> player) {
        for (String cardName:cards.keySet()){
            card = cards.get(cardName);
            if (canPurchase()){
                for (String tokenCard:card.keySet()){
                    player.put(tokenCard,player.get(tokenCard)
                            -card.get(tokenCard)
                            +playerCards.getOrDefault(tokenCard,0));
                }
                playerCards.put(cardName,playerCards.getOrDefault(cardName
,0)+1);
            }
        }
    }

    public static void print(Map<String,Integer> map){

        StringBuilder res = new StringBuilder();
        res.append("{");
        int idx = 0;
        for (String key:map.keySet()){
            res.append("\"").append(key).append("\"").append(":").append(map.get(key));
            if (idx<map.size()-1){
                res.append(",");
            }
            idx++;
        }
        res.append("}");
        System.out.println(res);
    }

    /**
     * 假设:
     * card:{"Red":4}
     * player:{"Red":6,"Blue":7}
     * playerCards:{"Red":1}
     * @param args
     */
    public static void main(String[] args) {
        TokensCardGamesII tg = new TokensCardGamesII();
        Map<String,Map<String,Integer>> cards = new HashMap<>();
        Map<String,Integer> redCard = new HashMap<>();
        redCard.put("Red",4);
        cards.put("Red",redCard);

        Map<String,Integer> player = new HashMap<>();
        player.put("Red",7);
        player.put("Blue",2);
        player.put("Black",5);
        tg.player = player;
        for (String cardName: cards.keySet()){
            tg.card = cards.get(cardName);
            boolean valid = tg.canPurchase();
            System.out.println(valid);
        }
        tg.purchase(cards, player);
        print(tg.player);
        print(tg.playerCards);
        tg.purchase(cards, player);
        print(tg.player);
        print(tg.playerCards);
        }
    }
```

Gold color

第四问:Money 增加一种 Gold color，万能色，可以在买卡的时候冲抵任何一种颜色。比如 下一张待购买的卡片 cost 需要 N 个 Red Money。手里只有 N–1 个 Red Money，但是有超 过一个 Gold Money，那也买得起

```java
public class TokensCardGamesIII {

    private Map<String,Integer> card;
    private Map<String,Integer> player;
    private Map<String,Integer> playerCards;
    private final static String GOLD_COLOR = "Gold";

    public TokensCardGamesIII(){
        this.card = new HashMap<>();
        this.player = new HashMap<>();
        this.playerCards = new HashMap<>();
    }

    public boolean canPurchase() {
        Map<String,Integer> resultCard = new HashMap<>();
        for (String tokenCard : player.keySet()) {
            if (card.containsKey(tokenCard)) {

                resultCard.put(tokenCard, card.get(tokenCard)
                        - player.get(tokenCard));
                for (String resultCardName : resultCard.keySet()) {
                    if (resultCard.get(resultCardName) <= 0) {
                        return true;
                    }else {
                        if (player.containsKey(GOLD_COLOR)){
                            int goldNums = player.get(GOLD_COLOR);
                            while (goldNums>0){
                                resultCard.put(resultCardName,resultCard.get(resultCardName)-1);
                                goldNums--;
                                if (resultCard.get(resultCardName)<=0){
                                    return true;
                                }
                            }
                        }
                    }
                }
            }
        }
        return false;
    }

    public void purchase(Map<String,Map<String,Integer>> cards, Map<String,Integer> player) {
        for (String cardName:cards.keySet()){
            card = cards.get(cardName);
            if (canPurchase()){
                for (String tokenCard:card.keySet()){
                    int remainderCard = player.get(tokenCard)
                            -card.get(tokenCard);
                    if(remainderCard>=0){
                        player.put(tokenCard,remainderCard);
                    }else {
                        player.put(tokenCard,0);
                        player.put(GOLD_COLOR, player.get(GOLD_COLOR)+remainderCard);
                    }
                }
                playerCards.put(cardName,playerCards.getOrDefault(cardName,0)+1);
            }
        }
    }

    public static void print(Map<String,Integer> map){

        StringBuilder res = new StringBuilder();
        res.append("{");
        int idx = 0;
        for (String key:map.keySet()){
            res.append("\"").append(key).append("\"").append(":").append(map.get(key));
            if (idx<map.size()-1){
                res.append(",");
            }
            idx++;
        }
        res.append("}");
        System.out.println(res);
    }

    /**
     * 假设:
     * card:{"Red":4}
     * player:{"Red":6,"Blue":7}
     * playerCards:{"Red":1}
     * @param args
     */
    public static void main(String[] args) {
        TokensCardGamesIII tg = new TokensCardGamesIII();
        Map<String,Map<String,Integer>> cards = new HashMap<>();
        Map<String,Integer> redCard = new HashMap<>();
        redCard.put("Red",4);
        cards.put("Red",redCard);

        Map<String,Integer> player = new HashMap<>();
        player.put("Red",6);
        player.put("Blue",2);
        player.put("Gold",5);
        tg.player = player;
        for (String cardName: cards.keySet()){
            tg.card = cards.get(cardName);
            boolean valid = tg.canPurchase();
            System.out.println(valid);
        }
        tg.purchase(cards, player);
        System.out.print("player:");
        print(tg.player);
        System.out.print("playerCards:");
        print(tg.playerCards);
        tg.purchase(cards, player);
        System.out.print("player:");
        print(tg.player);
        System.out.print("playerCards:");
        print(tg.playerCards);
    }
}
```

四子棋

```
                R

R               B

R               B

------------------------------

-4  -3  -2  -1  0   1   2   3   4
```

每次按照位置进行叠加，直到连续4个颜色一直为赢

put(color, pos) -> 返回bool true表示输入的颜色赢

可以使用

{"-4":"RR"}结构去做put

手机按键/电话号

给一个map， {"0":["a","b"], "1":["c"]}

和一个string，"001"

输出所有可能的组合["aac","abc","bac","bbc"]

我一开始就说遍历之后，每个可能的都选一次，然后把所有组成的string加入一个列表返回。当时觉得和permutation很想，就说用backtracking。但是写的时候又写成了recursion 加string concatenation。而且recursion外面还套了个iteration。。。后来debug了好久，屏幕都被print满了才发现recursion外面不应该用iteration，二者取其一就行。于是改对了。此处用了30分钟。。。

Java 复制代码

```java
public class LetterCombinations {

    public LetterCombinations(){
    }

    public List<String> getCombinations(String digits){
        List<String> combinations = new ArrayList<String>();
        if (digits.length() == 0) {
            return combinations;
        }
        Map<String,String> reflectMap = new HashMap<>();
        reflectMap.put("0","ab");
        reflectMap.put("1","c");
        System.out.println(JSONObject.toJSONString(reflectMap));
        backtrack(combinations,reflectMap,digits,0,new StringBuilder());
        return combinations;
    }

    private void backtrack(List<String> combinations,
                           Map<String,String> reflectMap,
                           String digits,
                           int index,
                           StringBuilder path){
        if (index>=digits.length()){
            combinations.add(new String(path));
        }else {
            String digit = String.valueOf(digits.charAt(index));
            String reflect = reflectMap.getOrDefault(digit,"");
            int count = reflect.length();
            for (int i=0;i<count;i++){
                path.append(reflect.charAt(i));
                backtrack(combinations,reflectMap,digits,index+1,
                        path);
                path.deleteCharAt(index);
            }
        }
    }

    public static void main(String[] args) {
        String digits = "001";
        List<String> ans = new LetterCombinations().getCombinations(digits
);
        System.out.println(JSONObject.toJSONString(ans));
    }
}
```

```
45
```

follow up

如果map的key不止一位，如加入一个"01":"z",

输出应该是["aac","abc","bac","bbc","az","bz"]

https://leetcode.cn/problems/letter-combinations-of-a-phone-number/

Java 复制代码

```java
public class LetterCombinationsII {

    public LetterCombinationsII(){
    }

    public List<String> getCombinations(String digits){
        List<String> combinations = new ArrayList<String>();
        if (digits.length() == 0) {
            return combinations;
        }
        Map<String,String> reflectMap = new HashMap<>();
        reflectMap.put("0","ab");
        reflectMap.put("1","c");
        reflectMap.put("01","z");
        System.out.println(JSONObject.toJSONString(reflectMap));
        backtrack(combinations,reflectMap,digits,0,new StringBuilder());
        return combinations;
    }

    private void backtrack(List<String> combinations,
                           Map<String,String> reflectMap,
                           String digits,
                           int index,
                           StringBuilder path){
        if (index>=digits.length()){
            combinations.add(new String(path));
        }else {
            for (int idx = 1;idx<digits.length()-index+1;idx++){
                String digit = digits.substring(index,index+idx);
                String reflect = reflectMap.getOrDefault(digit,"");
                int count = reflect.length();
                for (int i=0;i<count;i++){
                    path.append(reflect.charAt(i));
                    backtrack(combinations,reflectMap,digits,index+idx,
                            path);
                    path.deleteCharAt(index);
                }
            }
        }
    }

    public static void main(String[] args) {
        String digits = "001";
        List<String> ans = new LetterCombinationsII().getCombinations(digits);
```

```
            System.out.println(JSONObject.toJSONString(ans));
        }
    }
```

IP

valid ip address

IP V4:

1、个数是不是4个

2、每个长度不在 1–3之间

3、有前导0 并且长度不为1

4、计算数字

 不是数字

 数字超过255的

IP V6:

1、数量不是8个

2、每个串 长度不在1–4之间

3、遍历

 不是数字并且字母不在 a–f之间

Java 复制代码

```java
public class ValidIp {

    public String validIPAddress(String queryIP) {
        if (queryIP.indexOf('.') >= 0) {
            return isIpV4(queryIP) ? "IPv4" : "Neither";
        } else {
            return isIpV6(queryIP) ? "IPv6" : "Neither";
        }
    }

    public boolean isIpV4(String queryIP) {
        //加-1是防止出现空字符串无法计数 比如192.168.1.1. 后边多了一个点，不加-1会被忽略后边的空串
        String[] split = queryIP.split("\\.",-1);
        //个数不是4个
        if (split.length != 4) {
            return false;
        }
        for (String s : split) {
            //每个长度不在 1-3之间
            if (s.length() > 3 || s.length() == 0) {
                return false;
            }
            //有前导0 并且长度不为1
            if (s.charAt(0) == '0' && s.length() != 1) {
                return false;
            }
            //计算数字
            int ans = 0;
            for (int j = 0; j < s.length(); j++) {
                char c = s.charAt(j);
                //不是数字
                if (!Character.isDigit(c)) {
                    return false;
                }
                ans = ans * 10 + (c - '0');
            }
            //数字超过255
            if (ans > 255) {
                return false;
            }
        }
        return true;
    }

    public boolean isIpV6(String queryIP) {
        String[] split = queryIP.split(":",-1);
        //数量不是8个
        if (split.length != 8) {
            return false;
        }
        for (String s : split) {
            //每个串 长度不在1-4之间
            if (s.length() > 4 || s.length() == 0) {
                return false;
            }
            for (int i = 0; i < s.length(); i++) {
                char c = s.charAt(i);
                //不是数字并且字母不在 a-f之间
                if (!Character.isDigit(c) && !(Character.toLowerCase(c) >= 
'a')
                        || !(Character.toLowerCase(c) <= 'f')) {
                    return false;
                }
            }
        }
        return true;
    }

    public static void main(String[] args) {
        String ip= "192.168.1.1";
        String ans = new ValidIp().validIPAddress(ip);
        System.out.println(ans);
    }
}
```

generate all valid ip

给一个数字string，它可以组成多少个valid的 IP, 输出 eg: input "00123" output: "0.0.1.23" "0.0.12.3"

Java 复制代码

```java
public class RestoreIp {
    public List<String> restoreIpAddresses(String s) {
        int len = s.length();
        List<String> res = new ArrayList<>();
        // 如果长度不够，不搜索
        if (len < 4 || len > 12) {
            return res;
        }

        Deque<String> path = new ArrayDeque<>(4);
        int splitTimes = 0;
        dfs(s, len, splitTimes, 0, path, res);
        return res;
    }

    /**
     * 判断 s 的子区间 [left, right] 是否能够成为一个 ip 段
     * 判断的同时顺便把类型转了
     *
     * @param s
     * @param left
     * @param right
     * @return
     */
    private int judgeIfIpSegment(String s, int left, int right) {
        int len = right - left + 1;

        // 大于 1 位的时候，不能以 0 开头
        if (len > 1 && s.charAt(left) == '0') {
            return -1;
        }

        // 转成 int 类型
        int res = 0;
        for (int i = left; i <= right; i++) {
            res = res * 10 + s.charAt(i) - '0';
        }

        if (res > 255) {
            return -1;
        }
        return res;
    }

    private void dfs(String s, int len, int split, int begin, Deque<String> path, List<String> res) {
        if (begin == len) {
            if (split == 4) {
                res.add(String.join(".", path));
            }
            return;
        }

        // 看到剩下的不够了，就退出（剪枝），len - begin 表示剩余的还未分割的字符串的位数
        if (len - begin < (4 - split) || len - begin > 3 * (4 - split)) {
            return;
        }

        for (int i = 0; i < 3; i++) {
            if (begin + i >= len) {
                break;
            }

            int ipSegment = judgeIfIpSegment(s, begin, begin + i);
            if (ipSegment != -1) {
                // 在判断是 ip 段的情况下，才去做截取
                path.addLast(ipSegment + "");
                dfs(s, len, split + 1, begin + i + 1, path, res);
                path.removeLast();
            }
        }
    }

    public static void main(String[] args) {
        String words = "00123";
        List<String> ans = new RestoreIp().restoreIpAddresses(words);
        System.out.println(JSONObject.toJSONString(ans));
    }

}
```

其他

LC17, LC 62, LC91, LC93, LC468, LC469.