Backtracking related(need to review)

backtracking related 之前忘了.

90. Subsets II

link

思路1 : 每一个递归处理它后面的元素, 处理完了处理后面一个

```
public List<List<Integer>> subsetsWithDup(int[] nums) {
    Arrays.sort(nums);
    List<List<Integer>> res = new ArrayList<>();
    helper(res,new ArrayList<>(),nums,0);
    return res;
}

public void helper(List<List<Integer>> res, List<Integer> ls, int[] nums, int
pos) {
    res.add(new ArrayList<>(ls));
    for(int i=pos;i<nums.length;i++) {
        if(i>pos&&nums[i]==nums[i-1]) continue;
        ls.add(nums[i]);
        helper(res,ls,nums,i+1);
        ls.remove(ls.size()-1);
    }
}
```

思路2 : 每一个递归分成两部分, 加入或不加入

```
public List<List<Integer>> subsetsWithDup(int[] nums) {
    Arrays.sort(nums);
    List<List<Integer>> res = new ArrayList<>();
    helper(res,new ArrayList<>(),nums,0,false);
    return res;
}

public void helper(List<List<Integer>> res, List<Integer> ls, int[] nums, int
pos, boolean choosePre) {
    if(pos==nums.length) {
        res.add(new ArrayList<>(ls));
```

```
            return;
        }
        helper(res,ls,nums,pos+1,false);
        if(pos>=1&&nums[pos]==nums[pos-1]&&!choosePre) return;
        ls.add(nums[pos]);
        helper(res,ls,nums,pos+1,true);
        ls.remove(ls.size()-1);
    }
```

46. Permutations2

link

解法1, 交换法, 本质是12345, 1放在index的所有情况, 然后1跟2交换21345(index+1),这个时候1放在第二位, 重复之前的交换, 如果有重复11234, 第二次一定要建立hashSet来判断, 不能sort了判断前一个合后一个是否相等, 因为index变了

比方说0,0,1,2 当0遇2换的时候 2,0,1,0(index = 1)这个时候有两个0不连续, 会导致重复计数

写错了两个地方

1. 用了上买呢的sort和nums[i] == nums[i - 1]
2. 这个方法应该用index来判断条件而不是i.

```
public List<List<Integer>> permuteUnique(int[] nums) {
    List<List<Integer>> ans = new ArrayList<>();
    if (nums==null || nums.length==0) { return ans; }
    permute(ans, nums, 0);
    return ans;
}

private void permute(List<List<Integer>> ans, int[] nums, int index) {
    if (index == nums.length) {
        List<Integer> temp = new ArrayList<>();
        for (int num: nums) { temp.add(num); }
        ans.add(temp);
        return;
    }
    Set<Integer> appeared = new HashSet<>();
    for (int i=index; i<nums.length; ++i) {
        if (appeared.add(nums[i])) {
            swap(nums, index, i);
            permute(ans, nums, index+1);
```

```
                swap(nums, index, i);
            }
        }
    }

    private void swap(int[] nums, int i, int j) {
        int save = nums[i];
        nums[i] = nums[j];
        nums[j] = save;
    }
```

方法2

一定要加上visited[i - 1] == 0

经过debug实际上call stack是[1,1,2]先加入index0的1,然后第二个1break, 加入2, index是2, 不满足, 真正的1,1,2被加入的时候是 index = 1先加入第二个1, 然后加入第一个1, 再加入2.

visited[i - 1] == 0 或者visited[i - 1] != 0 都可以, 因为如果遇到重复的数, 最起码要加一次

```
    private void dfs(List<List<Integer>> res, List<Integer> current,int[]
    visited ,int times, int []nums){
        if(times == visited.length){
            res.add(new ArrayList(current));
            return;
        }
        for(int i = 0; i < visited.length; ++i){
            if(visited[i] == 1) continue;
            if(i > 0 && nums[i] == nums[i - 1] && visited[i - 1] == 0) continue;
            visited[i] = 1;
            current.add(nums[i]);
            dfs(res, current, visited, times + 1,nums);
            visited[i] = 0;
            current.remove(current.size() - 1);
        }
    }
```

77. Combinations

link

```
    public List<List<Integer>> combine(int n, int k) {
        List<List<Integer>> res = new ArrayList<>();
        helper(1, n, k, res, new ArrayList());
```

```
        return res;
    }
    public void helper(int start, int end, int k, List res, List tmp){
        if(tmp.size() == k){
            res.add(new ArrayList(tmp));
            return;
        }else{
            for(int i = start; i <= end; ++i){
                tmp.add(i);
                helper(i + 1, end, k, res, tmp);
                tmp.remove(tmp.size() - 1);
            }
        }
    }     public List<List<Integer>> combine(int n, int k) {
        List<List<Integer>> res = new ArrayList<>();
        helper(1, n, k, res, new ArrayList());
        return res;
    }
    public void helper(int start, int end, int k, List res, List tmp){
        if(tmp.size() == k){
            res.add(new ArrayList(tmp));
            return;
        }else{
            for(int i = start; i <= end; ++i){
                tmp.add(i);
                helper(i + 1, end, k, res, tmp);
                tmp.remove(tmp.size() - 1);
            }
        }
    }
```

一个巨大的优化是i <= n 改成i <= n - k + 1, 因为前面都不选, 后面就不够了

比如1,2,3,4,5 选4个, 选了一个1, 当i=2, 3可以, 但是当i=4的时候选不够4个了.所以循环到(n - left + 1)就行了.

更详细的总结: link

两种写法

```
    public List<List<Integer>> combine(int n, int k) {
        List<List<Integer>> res = new ArrayList<>();
        helper(1, n, k, res, new ArrayList());
        return res;
    }
    public void helper(int start, int end, int k, List res, List tmp){
        if(tmp.size() == k){
            res.add(new ArrayList(tmp));
            return;
        }else{
            for(int i = start; i <= end - (k - tmp.size()) + 1; ++i){
                tmp.add(i);
```

```
                helper(i + 1, end, k, res, tmp);
                tmp.remove(tmp.size() - 1);
            }
        }
    }
//或者
    public List<List<Integer>> combine(int n, int k) {
        List<List<Integer>> ans = new ArrayList<>();
        dfs(ans, new ArrayList<Integer>(), k, 1, n-k+1);
        return ans;
    }

    private void dfs(List<List<Integer>> ans, List<Integer> list, int kLeft, int
    from, int to) {
        if (kLeft == 0) { ans.add(new ArrayList<Integer>(list)); return; }
        for (int i=from; i<=to; ++i) {
            list.add(i);
            dfs(ans, list, kLeft-1, i+1, to+1);
            list.remove(list.size()-1);
        }
    }
```

93. Restore IP Addresses

最简单的第一次写, 想用一个index加入后面的, 但是不是很简洁, index可以改成substring的形式.

```
    public List<String> restoreIpAddresses(String s) {
        List<String> res = new ArrayList<>();
        if(s.length() > 12 || s.length() < 4) return res;
        helper(res, new ArrayList<String>(), 0, s);
        return res;
    }
    public void helper(List res, List<String> tmp, int index, String s){
        if(tmp.size() == 3){
            tmp.add(s.substring(index));
            String isValid = isValidAddress(tmp);
            if(isValid != null)
                res.add(isValid);
            tmp.remove(tmp.size() - 1);
            return;
        }else{
            for(int i = index; i < s.length(); ++i){
                if(i - index <= 3){
                    tmp.add(s.substring(index, i + 1));
                    helper(res, tmp, i + 1, s);
                    tmp.remove(tmp.size() - 1);
                }
            }
        }
    }
    public String isValidAddress(List<String> list){
        StringBuilder sb = new StringBuilder();
        for(String ip : list){
```

```
            if(ip.length() == 0 || ip.length() > 3) return null;
            if(0 == (ip.charAt(0) - '0') && ip.length() > 1) return null;
            if(Integer.parseInt(ip) > 255) return null;
            sb.append(ip);
            sb.append(".");
        }
        sb.deleteCharAt(sb.length() - 1);
        return sb.toString();
    }
```

131. Palindrome Partitioning

link

最开始写的, 直接用了substring没有用index, 跑的特别慢 7ms

```
    public List<List<String>> partition(String s) {
        List<List<String>> res = new ArrayList<>();
        helper(s, res, new ArrayList());
        return res;
    }

    public void helper(String s, List res, List tmp){
        if(s.length() > 0 && isValid(s)){
            tmp.add(s);
            res.add(new ArrayList(tmp));
            tmp.remove(tmp.size() - 1);
        }
        for(int i = 0; i < s.length(); ++i){
            String left = s.substring(0, i + 1);
            String right = s.substring(i + 1);
            if(isValid(left)){
                tmp.add(left);
                helper(right, res, tmp);
                tmp.remove(tmp.size() - 1);
            }
        }
    }

    public boolean isValid(String s){
        int i = 0, j = s.length() - 1;
        while(i <= j){
            if(s.charAt(i) - 'a' != s.charAt(j) - 'a')
                return false;
            i++;
            j--;
        }
        return true;
    }
```

这样写可能更好一些, 从下一个字符串开始, substring比较耗时, 写在里面比较好, 这个最后一次会加一次所以可以保证前面是回文.

```
    public List<List<String>> partition(String s) {
        List<List<String>> res = new ArrayList<>();
        helper(s, res, new ArrayList(), 0);
        return res;
    }

    public void helper(String s, List res, List tmp, int index){
        if(index == s.length()){
            res.add(new ArrayList(tmp));
            return;
        }
//这里是从index开始
        for(int i = index; i < s.length(); ++i){
            //最开始写在外面, 没有用index判断会慢很多.
            //String sub = s.substring(index, i + 1);
            if(isValid(s, index, i + 1)){
//这里传i或者在判断的时候j--
                tmp.add(s.substring(index, i + 1));
                helper(s, res, tmp, i + 1);
                tmp.remove(tmp.size() - 1);
            }
        }
    }

    public boolean isValid(String s, int i, int j){
        j--;
        while(i <= j){
            if(s.charAt(i) - 'a' != s.charAt(j) - 'a')
                return false;
            i++;
            j--;
        }
        return true;
    }
```

306. Additive Number

link

"1023”会被转换成1, 02, 3导致错误的结果, 得重新考虑, 这个递归里面如何返回值也不太清楚.

842. Split Array into Fibonacci Sequence 与306一模一样

link

写的错误的, 存在的问题, 第一个res.clear()否则会出现res里面元素会多

如果用Integer会出现比 Integer的最大还大的情况, long也是一样

解决0开头的问题

```
//错的
public List<Integer> splitIntoFibonacci(String S) {
        List<Integer> res = new ArrayList<>();
        helper(S, 0, res, new ArrayList());
        return res;
    }
    public boolean helper(String s, int index, List res, List<Long> tmp){
        if(index == s.length()){
//没有用新的list会存在着了结果继续运行
            if(tmp.size() > 2 && isValid(tmp)){
                res.clear();
                res.addAll(tmp);
                return true;
            }
            return false;
        }
        if(s.charAt(index) - '0' == 0){
            tmp.add(0L);
            helper(s, index + 1, res, tmp);
            tmp.remove(tmp.size() - 1);
        }else{
            for(int i = index; i < s.length(); ++i){
                Long value = Long.parseLong(s.substring(index, i + 1));
                if(value > Integer.MAX_VALUE) break;
                tmp.add(value);
                boolean result = helper(s, i + 1, res, tmp);
                tmp.remove(tmp.size() - 1);
                // if(result) return true;
            }
        }
```

```
        return false;
    }
    public boolean isValid(List<Long> tmp){
        for(int i = 2; i < tmp.size(); ++i){
            if(tmp.get(i) > Integer.MAX_VALUE ||tmp.get(i - 1) + tmp.get(i - 2) !=
    tmp.get(i))
                return false;
        }
        return true;
    }
```

偷的

```
public List<Integer> splitIntoFibonacci(String S) {
    List<Integer> ans = new ArrayList<>();
    helper(S, ans, 0);
    return ans;
}

public boolean helper(String s, List<Integer> ans, int idx) {
    if (idx == s.length() && ans.size() >= 3) {
        return true;
    }
    for (int i=idx; i<s.length(); i++) {
        if (s.charAt(idx) == '0' && i > idx) {
            break;
        }
        long num = Long.parseLong(s.substring(idx, i+1));
        if (num > Integer.MAX_VALUE) {
            break;
        }
        int size = ans.size();
        // early termination
        if (size >= 2 && num > ans.get(size-1)+ans.get(size-2)) {
            break;
        }
        if (size <= 1 || num == ans.get(size-1)+ans.get(size-2)) {
            ans.add((int)num);
            // branch pruning. if one branch has found fib seq, return true to upper
    call
            if (helper(s, ans, i+1)) {
                return true;
            }
            ans.remove(ans.size()-1);
        }
    }
    return false;
}
```

797. All Paths From Source to Target

link

注意一开始要把0加进去.

```
public List<List<Integer>> allPathsSourceTarget(int[][] graph) {
    List<List<Integer>> res = new ArrayList<>();
    if(graph.length == 0) return res;
    List<Integer> tmp = new ArrayList<>();
    tmp.add(0);
    helper(graph, tmp, res, 0);
    return res;
}
void helper(int[][] graph, List tmp, List res, int node){
    if(node == graph.length - 1){
        res.add(new ArrayList(tmp));
    }else{
        for(int current : graph[node]){
            tmp.add(current);
            helper(graph, tmp, res, current);
            tmp.remove(tmp.size() - 1);
        }
    }
}
```

偷的build了一个edge 用了两个stack

```
public List<List<Integer>> allPathsSourceTarget(int[][] graph) {
    int N = graph.length;
    List<List<Integer>> lss = new ArrayList<> ();
    Deque<Edge> stack = new ArrayDeque<> ();
    Deque<Integer> path = new ArrayDeque<> ();
    stack.push (new Edge (-1, 0));
    while (!stack.isEmpty ()) {
        Edge cur = stack.pop ();
        while (!path.isEmpty () && path.peekLast() != cur.parent)
            path.removeLast ();
        path.addLast (cur.val);
        if (cur.val == N - 1 || graph[cur.val].length == 0) {
            if (cur.val == N - 1)
                lss.add (new ArrayList<> (path));
            path.removeLast ();
        } else {
            for (int num : graph[cur.val])
                stack.push (new Edge (cur.val, num));
        }
    }
```

```
        return lss;
    }

    static class Edge {
        int parent, val;
        Edge (int p, int v) {
            this.parent = p;
            this.val = v;
        }

        public String toString () {
            return String.format ("[%d->%d]", parent, val);
        }
    }
```

bfs也可以

```
    public List<List<Integer>> allPathsSourceTarget(int[][] graph) {
        List<List<Integer>> result = new ArrayList();
        Queue<List<Integer>> queue = new LinkedList();
        queue.add(Arrays.asList(0));

        int destinationVertex = graph.length - 1;

        while(!queue.isEmpty()) {
            List<Integer> pathSoFar = queue.poll();
            int currentVertex = pathSoFar.get(pathSoFar.size() - 1);
            // check if currentVertex is destinationVertex add pathSoFar in result
            if(currentVertex == destinationVertex) result.add(new
    ArrayList(pathSoFar));
            for(int v : graph[currentVertex]) {
                List<Integer> newPath = new ArrayList(pathSoFar);
                newPath.add(v);
                queue.add(newPath);
            }
        }

        return result;
    }
```