July 29, 2018 题目：294,474,448,316,581,770,725,494,220,623,525,622,539

294

1. Backtracking

```
class Solution {
    public boolean canWin(String s) {
        if (s == null || s.length() < 2) return false;
        for (int i = 0; i < s.length(); i++) {
            if (s.startsWith("++", i)) {
                String str = s.substring(0, i) + "--" + s.subs
tring(i + 2);
                if (!canWin(str)) {
                    return true;
                }
            }
        }
        return false;
    }
}
```

2. Game Theory，找状态数：

```
class Solution {
public:
    bool canWin(string s) {
        vector<int> pool;
        int max_length = 0, d = 0;
        auto i = s.find('+');
        while(i!=string::npos){
            auto j = s.find('-', i);
            if(j==string::npos) j = s.size();
            if(j-i>=2) {
                pool.push_back(j-i);
                max_length = max(max_length, int(j-i));
            }
            i = s.find('+', j);
        }
        vector<int> dp(max_length+1, 0);
        for(int i=2; i<=max_length; ++i){
            set<int> S;
            for(int j=0; j<=i-2 && !dp[i]; ++j){
                S.insert(dp[j] ^ dp[i-j-2]);
            }
            while(S.count(dp[i])) ++dp[i];
        }
        for(auto k: pool) d^=dp[k];
        return d;
    }
};
```

3. recursive backtracking.... similar to the first one...

```
class Solution {
public:
    // to avoid the curse of recursion
    int len;
    string ss;
    bool canWin(string s) {
        ss = s;
        len = s.length();
        return canIWin();
    }
    bool canIWin() {
        // corner case goes here
        if (len < 2) {
            return false;
        }
        int i;
        for (i = 0; i <= len-2; i++) {
            if(ss[i] == '+' && ss[i+1] == '+') {
                ss[i] = '-'; ss[i+1] = '-';
                // player 2 's turn'
                bool wins = !canIWin();
                // player 1 try a different solution
                ss[i]='+'; ss[i+1]='+';
                if (wins) return true;
            }
        }
        return false;// no available move/ opponent wins
    }
};
```

474

1. 背包问题

```
class Solution {
    public int findMaxForm(String[] strs, int m, int n) {
        int[][] dp = new int[m + 1][n + 1];

        for (String str : strs) {
            int zeros = getZeros(str);
            int ones = str.length() - zeros;
            for (int i = m; i >= zeros; i--) {
                for (int j = n; j >= ones; j--) {
                    dp[i][j] = Math.max(dp[i - zeros][j - one
s] + 1, dp[i][j]);
                }
            }
        }
        return dp[m][n];
    }

    public int getZeros(String s) {
        int k = 0;

        for (char c : s.toCharArray()) {
            if (c == '0') k++;
        }
        return k;
    }
}
```

2. 很慢：

```
class Solution {
    typedef pair<int, int> ii;
    typedef vector<int> vi;
    vector<ii> B;
    vector<map<ii, int>> dp;
    int dfs(int i, int m, int n){
        if(i==B.size()) return 0;
        if(dp[i].count(ii(m, n))) return dp[i][ii(m, n)];
        dp[i][ii(m, n)] = dfs(i+1, m, n);
        if(B[i].first<=m && B[i].second<=n) {
            dp[i][ii(m, n)] = max(dp[i][ii(m, n)], 1 + dfs(i+
1, m-B[i].first, n-B[i].second));
        }
        return dp[i][ii(m, n)];
    }
public:
    int findMaxForm(vector<string>& strs, int m, int n) {
        for(string s: strs){
            int m_ = 0, n_ = 0;
            for(char c: s) {
                if(c=='1') ++n_;
                else ++m_;
            }
            B.push_back(ii(m_, n_));
        }
        dp.resize(strs.size());
        return dfs(0, m, n);
    }
};
```

- [ ] Optimize the above method. @Zebo L

448

1. treat the array as sort of hash table, and mark elements as negative using `nums[nums[i] -1] = -nums[nums[i]-1]` to ensure all visited numbers are negative. Then iterate again, positive number is disappeared elements.

```
class Solution:
    def findDisappearedNumbers(self, nums):
        """
        :type nums: List[int]
        :rtype: List[int]
        """
        ret = []
        for i in range(len(nums)):
            index = abs(nums[i]) -1 # because array index star
ts from 0
            nums[index] = -abs(nums[index])
        for i in range(len(nums)):
            if nums[i] > 0:
                ret.append(i+1)
        return ret
```

2. Two pass:

```
class Solution {
public:
    vector<int> findDisappearedNumbers(vector<int>& nums) {
        int n = nums.size(), i = 0;
        while(i<n){
            while(nums[i] != i+1 && nums[nums[i]-1] != nums
[i]){
                swap(nums[nums[i]-1], nums[i]);
            }
            ++i;
        }
        vector<int> ans;
        for(int i=0;i<n;++i) if(nums[i]!=i+1) ans.push_back(i+
1);
        return ans;
    }
};
```

316

7.28做过了

581

1.
```
class Solution {
    public int findUnsortedSubarray(int[] nums) {
        int i = 0, j = nums.length - 1;
        int max = Integer.MIN_VALUE, min = Integer.MAX_VALUE;

        while (i < j && nums[i] <= nums[i + 1]) {
            i++;
        }
        if (i == j) return 0;

        while (i < j && nums[j - 1] <= nums[j]) {
            j--;
        }
        for (int k = i; k <= j; k++) {
            min = Math.min(min, nums[k]);
            max = Math.max(max, nums[k]);
        }

        while (i >= 0 && nums[i] > min) {
            i--;
        }
        while (j < nums.length && nums[j] < max) {
            j++;
        }

        return j - i - 1;
    }
}
```
2. 构造最小值跟最大值序列：

```
class Solution {
public:
    int findUnsortedSubarray(vector<int>& nums) {
        int n = nums.size(), i =0, j = nums.size()-1;
        vector<int> M(n, nums[0]), m(n, nums[n-1]);
        for(int i=1; i<n; ++i) M[i] = max(M[i-1], nums[i]);
        for(int i=n-2; i>=0; --i) m[i] = min(m[i+1], nums[i]);
        while(i<n && nums[i]==m[i]) ++i;
        if(i==n) return 0;
        while(j>=0 && nums[j]==M[j]) --j;
        return j-i+1;
    }
};
```

770

725

1.

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
class Solution {
    public ListNode[] splitListToParts(ListNode root, int k) {
        int len = 0;
        ListNode node = root;
        ListNode[] res = new ListNode[k];
        while (node != null) {
            len++;
            node = node.next;
        }

        int num = len / k;
        if (num * k != len) num++;
        node = root;
        int i = 0;
        int cur = 0;

        while (len != 0 && node != null) {
            cur++;

            if (cur == num && node != null) {
                ListNode tmp = node.next;
                node.next = null;

                res[i++] = root;
                k--;
                if (k == 0) break;
                root = tmp;
                node = root;
                len -= num;
                num = len / k;
                if (num * k != len) num++;
                cur = 0;
            } else node = node.next;
        }
        return res;
    }
}
```

2. 跟楼上类似

```
class Solution {
public:
    vector<ListNode*> splitListToParts(ListNode* root, int k)
{
        int n = 0;
        for(auto p=root; p; p=p->next) ++n;
        vector<ListNode*> ans;
        for(int i=0; i<k; ++i){
            ListNode lead(0);
            ListNode *p = &lead;
            for(int j=0; j<n/k + (i<n%k); ++j){
                p->next = root;
                root = root->next;
                p = p->next;
            }
            p->next = NULL;
            ans.push_back(lead.next);
        }
        return ans;
    }
};
```

494

1. DFS

```
class Solution {
    int count = 0;
    public int findTargetSumWays(int[] nums, int S) {
        helper(nums, 0, 0, S);
        return count;
    }

    public void helper(int[] nums, int i, int target, int S) {
        if (i == nums.length) {
            if (target != S) return;
            count++;
            return;
        }

        int cur = nums[i];
        helper(nums, i + 1, target + cur, S);
        helper(nums, i + 1, target - cur, S);
    }
}
```

2. 同上：

```
class Solution {
    int sum, ans, n;
    void dfs(int i, int cur, int stat, const int&target, vecto
r<int> &nums){
        if(i==n) {
            if(2*cur == target + sum) ++ans;;
            return;
        }
        if(2*cur > sum + target) return;
        dfs(i+1, cur, stat, target, nums);
        dfs(i+1, cur+nums[i], stat | (1<<i), target, nums);
    }
public:
    int findTargetSumWays(vector<int>& nums, int S) {
        sum = ans = 0;
        n = nums.size();
        for(auto k: nums) sum+=k;
        sort(nums.begin(), nums.end());
        dfs(0, 0, 0, S, nums);
        return ans;
    }
};
```

- [ ] 研究以下方法 @Zebo L

```
class Solution {
public:
    int findTargetSumWays(vector<int>& nums, int s) {
        int sum = accumulate(nums.begin(), nums.end(), 0);
        return sum < s || (s + sum) & 1 ? 0 : subsetSum(nums,
(s + sum) >> 1);
    }

    int subsetSum(vector<int>& nums, int s) {
        int dp[s + 1] = { 0 };
        dp[0] = 1;
        for (int n : nums)
            for (int i = s; i >= n; i--)
                dp[i] += dp[i - n];
        return dp[s];
    }
};
```

220

1.

```
class Solution {
    public boolean containsNearbyAlmostDuplicate(int[] nums, i
nt k, int t) {
        for (int i = 0; i < nums.length; i++) {
            for (int j = 1; j <= k; j++) {
                if (i + j < nums.length && Math.abs((long) num
s[i] - (long) nums[i + j]) <= t) return true;
            }
        }
        return false;
    }
}
```

2. Maintain 区间 + sliding window :

```
class Solution {
public:
    bool containsNearbyAlmostDuplicate(vector<int>& nums, int
k, int t) {
        map<long, int> cnt;
        cnt[3*long(INT_MIN)] = 1;
        cnt[3*long(INT_MAX)] = 1;
        int n = nums.size();
        ++k;
        for(int i=0; i<k && i<n; ++i){
            auto it = cnt.lower_bound(long(nums[i]));
            if(it->first - (long)nums[i] <= (long)t) return tr
ue;
            --it;
            if(long(nums[i]) - it->first <= (long)t) return tr
ue;
            cnt[long(nums[i])]++;
        }
        for(int i=k; i<n; ++i){
            cnt[long(nums[i-k])]--;
            if(!cnt[long(nums[i-k])]) cnt.erase(long(nums[i-
k]));
            auto it = cnt.lower_bound(long(nums[i]));
            if(it->first - (long)nums[i] <= (long)t) return tr
ue;
            --it;
            if(long(nums[i]) - it->first <= (long)t) return tr
ue;
            cnt[long(nums[i])]++;
        }
        return false;
    }
};
```

623

1. 暴力tree traversal

```
class Solution {
    public TreeNode addOneRow(TreeNode root, int v, int d) {
        if (root == null) return root;
        if (d == 1) {
            TreeNode newRoot = new TreeNode(v);
            newRoot.left = root;
            return newRoot;
        }
        TreeNode node = root;
        helper(node, v, d, 1);
        return root;
    }
    public void helper(TreeNode node, int v, int d, int depth)
{
        if (node == null) return;
        if (depth == d - 1) {
            TreeNode left = node.left;
            TreeNode right = node.right;

            node.left = new TreeNode(v);
            node.right = new TreeNode(v);
            node.left.left = left;
            node.right.right = right;
            return;
        }
        helper(node.left, v, d, depth + 1);
        helper(node.right, v, d, depth + 1);
    }
}
```

2. 同上，c++ version：

```
class Solution {
    void dfs(TreeNode *root, int dep, const int&d, const int&
v){
        if(!root) return;
        if(dep == d-1){
            TreeNode *l = new TreeNode(v), *r = new TreeNode
(v);
            l->left = root->left;
            r->right = root->right;
            root->left = l;
            root->right = r;
            return;
        }
        dfs(root->left, dep+1, d, v);
        dfs(root->right, dep+1, d, v);
    }
public:
    TreeNode* addOneRow(TreeNode* root, int v, int d) {
        if(d == 1){
            TreeNode *l = new TreeNode(v);
            l->left = root;
            return l;
        }
        dfs(root, 1, d, v);
        return root;
    }
};
```

525

1. 用map keep track of the difference of zeros and ones

```
class Solution {
    public int findMaxLength(int[] nums) {
        Map<Integer, Integer> map = new HashMap<>();
        map.put(0, -1);
        int max = 0;
        int one = 0;
        int zero = 0;
        for (int i = 0; i < nums.length; i++) {
            if (nums[i] == 0) zero++;
            else one++;
            if (map.containsKey(zero - one)) {
                max = Math.max(max, i - map.get(zero - one));
            } else {
                map.put(zero - one, i);
            }
        }
        return max;
    }
}
```

2. 部分和： 遇到`1` `+1`遇到`0` `-1`，最后看哪些部分和为`0`.

```
class Solution {
public:
    int findMaxLength(vector<int>& nums) {
        int n = nums.size(), ans = 0;
        vector<int> P(n+1, 0);
        unordered_map<int, int> pos;
        pos[0] = 0;
        for(int i=1; i<=n; ++i) {
            P[i] = P[i-1] + (nums[i-1]? 1: -1);
            if(pos.count(P[i])) ans = max(ans, i - pos[P[i]]);
            else pos[P[i]] = i;
        }
        return ans;
    }
};
```

622

1. 题简单，但别把Rear 跟 Front 搞混就行了：

```
class MyCircularQueue {
public:
    vector<int> C;
    int n, sz, i, j;
    MyCircularQueue(int k): n(0), sz(k), i(0), j(0), C(vector<
int>(k, 0)) {}
    bool enQueue(int value) {
        if(n==sz) return false;
        C[j%sz] = value;
        ++n;
        ++j;
        return true;
    }
    bool deQueue() {
        if(!n) return false;
        ++i;
        --n;
        return true;
    }
    int Front() {
        if(!n) return -1;
        return C[i%sz];
    }
    int Rear() {
        if(!n) return -1;
        return C[(j-1)%sz];
    }
    bool isEmpty() { return !n; }
    bool isFull() { return n==sz; }
};
```

539

1. 用sort提高efficiency

```
class Solution {
    class Time {
        int hour;
        int minute;
        public Time(int hour, int minute) {
            this.hour = hour;
            this.minute = minute;
        }

        public int getDiff(Time t) {
            return 60 * (this.hour - t.hour) + this.minute -
t.minute;
        }
    }
    public int findMinDifference(List<String> timePoints) {
        List<Time> times = new ArrayList<>();
        for (String s : timePoints) {
            String[] time = s.split(":");
            times.add(new Time(Integer.valueOf(time[0]), Integ
er.valueOf(time[1])));
        }
        Collections.sort(times, new Comparator<Time>(){
            public int compare(Time a, Time b) {
                if (a.hour == b.hour) return a.minute - b.minu
te;
                return a.hour - b.hour;
            }
        });

        Time first = times.get(0);
        times.add(new Time(first.hour + 24, first.minute));

        int min = Integer.MAX_VALUE;
        for (int i = 0; i < timePoints.size(); i++) {
            min = Math.min(min, Math.abs(times.get(i).getDiff
(times.get(i + 1))));
        }
        return min;
    }
}
```

2. Python:

```
class Solution:
    def findMinDifference(self, timePoints):
        ts = sorted([60*int(t[:2]) + int(t[-2:]) for t in timePoints])
        ts += [t + 24 * 60 for t in ts]
        return min([t2 - t1 for t1, t2 in zip(ts[:-1], ts[1:])])
```