July 12, 2018

266

1. Very straightforward
2. 同上:

```
class Solution {
public:
    bool canPermutePalindrome(string s) {
        map<char, int> cnt;
        for(char c:s) cnt[c]++;
        int odd=0;
        for(auto p:cnt) if(p.second%2) ++odd;
        return odd<2;
    }
};
```

116

1. O(n) time, key is the cur.right.next = cur.next.left if cur.next is not null
2. Recursion:

```
class Solution {
public:
    void connect(TreeLinkNode *root) {
        if(!root) return;
        for(auto p1=root->left, p2=root->right; p1&&p2; p1=p1->right, p2=p2->left) p1->next = p2;
        connect(root->left);
        connect(root->right);
    }
};
```

221

1. Very straightforward 2d dp.
2. 1d dp solution:

```
class Solution {
public:
    int maximalSquare(vector<vector<char>>& matrix) {
        if(matrix.empty() || matrix[0].empty()) return 0;
        int n = (int)matrix.size(), m = (int)matrix[0].size(),
ans = 0;
        vector<int> dp(m, 0);
        for(int i=0;i<n;++i) for(int j=0;j<m;++j){
            if(matrix[i][j] == '0') dp[j] = 0;
            else if(!i || !j) dp[j] = 1;
            else if(dp[j]!=dp[j-1]) dp[j] = min(dp[j], dp[j-
1]) + 1;
            else{
                dp[j] = dp[j-1] + (int)(matrix[i-dp[j-1]][j-dp
[j-1]]=='1');
            }
            ans = max(ans, dp[j]);
        }
        return ans * ans;
    }
};
```

788

1. simulation or dp

```
class Solution {
    public int rotatedDigits(int N) {
        int[] dp = new int[N + 1];
        int res = 0;
        for (int i = 0; i < N + 1; i++) {
```

```
            if (i < 10) {
                if (i == 2 || i == 5 || i == 6 || i == 9) {
                    dp[i] = 2;
                    res++;
                }
                if (i == 1 || i == 0 || i == 8) {
                    dp[i] = 1;
                }
            } else {
                int l = i / 10;
                int r = i % 10;
                if (dp[l] == 1 && dp[r] == 1) {
                    dp[i] = 1;
                }else if (dp[l] >= 1 && dp[r] >= 1) {
                    dp[i] = 2;
                    res++;
                }
            }
        }

        return res;
    }
}
```

2. 一个一个数是很简单, 但复杂度是O(N)

```
class Solution {
    set<int> ref={2,5,6,9};
    set<int> bad={3,4,7};
    int good(int N){
        int cnt = 0;
        while(N){
```

```
            int dig = N%10;
            if(bad.count(dig)) return 0;
            cnt += ref.count(dig);
            N/=10;
        }
        return (int)(cnt>0);
    }
public:
    int rotatedDigits(int N) {
        int ans = 0;
        for(int i=1;i<=N;++i) ans += good(i);
        return ans;
    }
};
```

- [ ] Think about 容斥原理 @Zebo L

550

empty

736

- [ ] Solve it later @Zebo L (题太长)

548

1. 超级慢的solution：

```
class Solution {
    unordered_map<int, set<int>> pos;
    vector<unordered_map<int, unordered_map<int, bool> > > dp;
    vector<int> P;
    int N;
    bool dfs(int i, int j, int sum){
        if(j>=N-1) return false;
        if(i==2){
```

```
            return (sum == P[N] - P[j+1]);
        }
        if(dp[i].count(j) && dp[i][j].count(sum)) return dp[i]
[j][sum];
        dp[i][j][sum] = false;
        int target = sum + P[j+1];
        if(pos.count(target)){
            auto ss = pos[target];
            for(auto it=ss.upper_bound(j); it!=ss.end(); ++it)
if((*it) > j+1) {
                dp[i][j][sum] = (dp[i][j][sum] || dfs(i+1, *i
t, sum));
                if(dp[i][j][sum]) return dp[i][j][sum];
            }
        }
        return dp[i][j][sum];
    }
public:
    bool splitArray(vector<int>& nums) {
        if(nums.size()<7) return false;
        N = nums.size();
        P.resize(N+1);
        dp.resize(2);
        P[0] = 0;
        for(int i=0; i<N; ++i){
            P[i+1] = P[i] + nums[i];
            pos[P[i+1]].insert(i+1);
        }
        for(int i=1;i<N;++i) if(dfs(0, i, P[i])) return true;
        return false;
    }
```

```
};
```

- [ ] See Leetcode discussion @Zebo L

2. O(n^2) hashset

```
class Solution {
    public boolean splitArray(int[] nums) {
        if (nums == null || nums.length <= 6) {
            return false;
        }
        int[] sum = new int[nums.length + 1];

        for (int i = 1; i <= nums.length; i++) {
            sum[i] = sum[i - 1] + nums[i - 1];
        }

        for (int j = 3; j < nums.length - 2; j++) {
            Set<Integer> set = new HashSet<>();
            for (int k = j + 2; k < nums.length - 1; k++) {
                if ((sum[k] - sum[j + 1]) == (sum[nums.length]
- sum[k + 1])) {
                    set.add(sum[k] - sum[j + 1]);
                }
            }
            for (int i = 1; i < j - 1; i++) {
                int left_i_sum = sum[i] - sum[0];
                int i_j_sum = sum[j] - sum[i + 1];
                if (left_i_sum != i_j_sum) {
                    continue;
                }
                if (set.contains(left_i_sum)) {
                    return true;
```

```
                }
            }
        }
        return false;
    }
}
```

217

1. straightforward
2. O(n) space solution:

```
class Solution {
public:
    bool containsDuplicate(vector<int>& nums) {
        return set<int>(nums.begin(), nums.end()).size() != nu
ms.size();
    }
};
```

- [ ] Consider O(1) space solution, maybe does not exist @Zebo L

656

1. Forward track 一下即可：

```
class Solution {
    const int inf = 1E8;
public:
    vector<int> cheapestJump(vector<int>& A, int B) {
        int n = A.size();
        if(A[n-1] == -1 || A[0] == -1) return vector<int>{};
        vector<int> dp(n, inf), next(n, n), ans;
        dp[n-1] = A[n-1];
        for(int i=n-2;i>=0;--i) if(A[i]!=-1){
            for(int j=i+1; j<=min(i+B, n-1); ++j) if(dp[j] + A
[i] < dp[i]){
```

```
                next[i] = j;
                dp[i] = dp[j] + A[i];
            }
            if(dp[i]>=inf) return ans;
        }
        for(int k=0;k<n;k=next[k]){
            ans.push_back(k+1);
        }
        return ans;
    }
};
```

2. dp but need to record the path. Update dp from right to left so the path would be the smallest updates. ********** why update the results this way???

718

1. O(n*m) solution:

```
class Solution {
public:
    int findLength(vector<int>& A, vector<int>& B) {
        int n=A.size(), m=B.size(), ans = 0;
        for(int diff=0; diff<m; ++diff){
            for(int i=0, cnt=0; i<n && i+diff<m; ++i){
                if(A[i]==B[i+diff]) {
                    ++cnt;
                    ans = max(ans, cnt);
                }
                else cnt=0;
            }
        }
```

```
        for(int diff=0; diff<n; ++diff){
            for(int j=0, cnt=0; j<m && j+diff<n; ++j){
                if(A[j+diff]==B[j]) {
                    ++cnt;
                    ans = max(ans, cnt);
                }
                else cnt=0;
            }
        }
        return ans;
    }
};
```

- [ ] Think about faster solution @Zebo L

2. dp, the same as the longest common substring.

```
class Solution {
    public int findLength(int[] A, int[] B) {
        int[][] dp = new int[A.length + 1][B.length + 1];
        int res = 0;
        for (int i = 1; i <= A.length; i++) {
            for (int j = 1; j <= B.length; j++) {
                if (A[i - 1] == B[j - 1]) {
                    dp[i][j] = dp[i - 1][j - 1] + 1;
                    res = Math.max(res, dp[i][j]);
                }
            }
        }
        return res;
    }
}
```