Homework 2

Student Name: ZhangLe(张乐)

Student ID: 201628013229047

Question 2

A robber is planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and it will automatically contact the police if two adjacent houses were broken into on the same night.

1. Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight without alerting the police.
2. What if all houses are arranged in a circle?

DP Equation

$dp[i] = max(dp[i-2] + nums[i-1], dp[i-1])$

对于第二问，当房子围成一个圈的时候，如果我们选择抢劫第一个房子，那么我们就不能抢劫最后一个房子；如果抢劫最后一房子，就不能抢劫第一个房子。

即求$max(houseRobber(nums[0...len-1]), houseRobber(nums[1...len]))$

Pseudo-code

1.

Java

```java
public class Solution {
    /**
     * @param nums: An array of non-negative integers.
     * return: The maximum amount of money you can rob tonight
     */
    public long houseRobber(int[] nums) {
        if(A.length==0){
            return 0;
        }
        long dp[]=new long [A.length+1];
        dp[0]=0;
        dp[1]=A[0];
        for(int i=2;i<=A.length;i++){
            dp[i]=Math.max(dp[i-2]+A[i-1],dp[i-1]);
        }
        return dp[dp.length-1];
    }
}
```

2.

```java
public class Solution {
    /**
     * @param nums: An array of non-negative integers.
     * return: The maximum amount of money you can rob tonight
     */
    public int houseRobber2(int[] nums) {
        if(nums.length==0){
            return 0;
        }
        if(nums.length==1)
            return nums[0];
        return (int)Math.max(
             houseRobber(Arrays.copyOfRange(nums, 1, nums.length)),
             houseRobber(Arrays.copyOfRange(nums, 0, nums.length-1))
             );
    }
    public long houseRobber(int[] A) {
        if(A.length==0){
            return 0;
        }
        long dp[]=new long [A.length+1];
        dp[0]=0;
        dp[1]=A[0];
        for(int i=2;i<=A.length;i++){
            dp[i]=Math.max(dp[i-2]+A[i-1],dp[i-1]);
        }
        return dp[dp.length-1];
    }
}
```

Provement

初始化$dp[0] = 0$，$dp[1] = nums[0]$

现在我们要求抢劫前i个房屋获得的最大收益，可以考虑两种情况：抢劫第i个房屋和不抢劫第i个房屋。

当抢劫第i个房屋时，收益为 $dp[i-2] + A[i-1]$。

当不抢劫第i个房屋时，收益为 $dp[i-1]$。

如果取其中最大的即求得抢劫前i个房屋获得的最大收益。

Complexity

只使用一个循环，易得算法的时间复杂度为$O(n)$

Question 3

> 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.

DP Equation

初始化$dp[i] = i - 1$。$dp[i] = min(dp[i], dp[j] + 1)$ 其中 $j < i$ 并且 $s[j...i]$ 回文

Pseudo-code

Java

```java

class Solution {
    /**
     * @param s a string
     * @return an integer
     */
    public int minCut(String s) {
        // write your code here
        int dp[] = new int[s.length() + 1];
        for (int i = 0; i < dp.length; i++) {
            dp[i] = i - 1;
        }
        for (int i = 0; i < dp.length; i++) {
            for (int j = 0; j < i; j++) {
                if (palindrome(s.substring(j, i))) {
                    dp[i] = Math.min(dp[i], dp[j] + 1);
                }
            }

        }
        return dp[dp.length - 1];
    }

    public boolean palindrome(String a) {
        StringBuffer sb = new StringBuffer(a);
        return sb.reverse().toString().equals(a);
    }
}
```

Provement

初始化$dp[i] = i - 1$。现在我们要求$s[0...i]$的最小划分数，可以考虑$s[j...i](j < i)$是否为回文，如果是取$dp[i] = min(dp[i], dp[j] + 1)$，其中$j$所有满足$s[j...i](j < i)$为回文。$dp[i]$既是所求。

Complexity

两重循环，复杂度为n^2 。回文判断复杂度为$O(len(s))$，故总的时间复杂度近似为 $O(n^3)$

Question 4

> A message containing letters from A-Z is being encoded to numbers using the following mapping:
>
> $$A:1$$
> $$B:1$$
> $$\dots$$
> $$Z:26$$
>
> Given an encoded message containing digits, determine the total number of ways to decode it.
>
> For example, given encoded message "12", it could be decoded as "AB" (1 2) or "L" (12).T he number of ways decoding "12" is 2

DP Equation

$$dp[i] = \begin{cases} 0 & s[i] ==' 0' \text{但是} s[i-1] \text{不是}'1' \text{或}'2' \\ dp[i-2] & s[i] ==' 0' \text{且} s[i-1..i] \text{是一个合法编码} \\ dp[i-1] + dp[i-2] & s[i]! =' 0' \text{且} s[i-1..i] \text{是一个合法编码} \\ dp[i-1] & \text{其他} \end{cases}$$

Pseudo-code

```java
public class Solution {
    /**
     * @param s a string,  encoded message
     * @return an integer, the number of ways decoding
     */
    public int numDecodings(String s) {
        if(s.length()==0)
            return 0;
        int dp[]=new int[s.length()+1];
        dp[0]=1;
        dp[1]=s.equals("0")?0:1;
        for(int i=2;i<dp.length;i++){
            if(s.charAt(i-1)=='0'){
                if(s.charAt(i-2)!='1'&&s.charAt(i-2)!='2')
                    dp[i]=0;
                else
                    dp[i]=dp[i-2];

            }
            else if(s.charAt(i-2)=='1')
                dp[i]=dp[i-1]+dp[i-2];
            else if(s.charAt(i-2)=='2'&&s.charAt(i-1)>'0'&&s.charAt(i-1)<='6')
                dp[i]=dp[i-1]+dp[i-2];
            else
                dp[i]=dp[i-1];
        }
        return dp[dp.length-1];
    }
}
```

Provement

根据这里动态规划的递推式，不难证明

Complexity

只使用一个循环，易得算法的时间复杂度为$O(n)$

Question 6

> You have an array for which the i-th element is the price of a given stock on day i.
>
> Design an algorithm and implement it to find the maximum profit. You may complete at most two transactions.

> *Note*: You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

DP Equation

最多允许两次不相交的交易，也就意味着这两次交易间存在某一分界线，考虑到可只交易一次，也可交易零次，故分界线的变化范围为第一天至最后一天，只需考虑分界线两边各自的最大利润，最后选出利润和最大的即可。

对于求解单次交易最大利润问题，等价于找出差值最大的一对波谷和波峰。故需要引入一个索引用于记录当前的波谷，结果即为当前索引值减去波谷的值。

Solution

Java

```java
class Solution {
    /**
     * @param prices: Given an integer array
     * @return: Maximum profit
     */
    public int maxProfit(int[] prices) {
        if(prices.length==1)
            return 0;
        int p=0;
        for(int i=0;i<prices.length;i++){
            p=Math.max(p,
            maxOneTransformProfit(prices, 0, i)+
            maxOneTransformProfit(prices, i+1, prices.length-1)
            );

        }
        return p;
    }
    public int maxOneTransformProfit(int[] prices,int b,int e){
        if(b>=e)
            return 0;
        int p=0;
        int curr=Integer.MAX_VALUE;

        for(int i=b;i<=e;i++){
            p=Math.max(p,prices[i]-curr);
            curr=Math.min(curr,prices[i]);
        }
        return p;
    }
};
```