第二次作业

姓名：梁付槐　　学号：2018Z8013261003

题目一：Money robbing

问题 1

算法：我们定义街上共有 n 座房子，第 i 座房子里钱的价值为 V(i). 设 OPTROB(i)为抢劫者到达第 i 座房子时可以获得的最大价值。那么，如果抢劫者抢劫第 i 座房子，他获得的价值为 OPTROB(i-2)+V(i), 否则他必定需要抢劫第 i-1 座房子，获得的价值为 OPTROB(i-1). 这样的话，子问题就变为抢劫者到达第 i 座房子时获得最大价值即可！

$$DP方程: OPTROB(i) = \max\{OPTROB(i-2)+V(i), OPTROB(i-1)\}$$

伪代码：

```
1  OPTROB(i){
2      if(i==0)
3          return 0;
4      if(i==1)
5          return V(i);
6      OPT=max(OPTROB(i-2)+V(i),OPTROB(i-1));
7      return OPT;
8  }
```

算法的正确性：

设想抢劫者到第 i 座房子时有一个更好的价值 OPTROB'(i)>OPTROB(i), 抢劫者已经决定了抢劫第 i 座房子，则 OPTROB(i)=OPTROB(i-2)+V(i). 那么 OPT'(i)就应是另一中选择，即抢劫第 i-1 座房子，OPTROB'(i)=OPTROB(i-1).
而且 OPTROB(i-1)>OPTROB(i-2)+V(i), 这显然与 OPTROB()函数的定义矛盾！

时间复杂度分析：每一步递归中，只有一层计算，因此 T(n)=O(n).

问题 2

算法：如果房子围成一个圈，那么问题实质上还是一样的。可以分成两个子问题：包含第一座房子但不包含最后一座房子、不包含第一座房子但包含最后一座房子。这样就将问题转化为两个 n-1 规模的子问题，然后只需要在最后再取两者的最大值即可！

伪代码：

```
OPTROBFIRST(i){
    if(i==0)
        return 0;
    if(i==1)
        return V(i);
    OPT=max(OPTROB(i-2)+V(i),OPTROB(i-1));
    return OPT;
}

OPTROBLAST(i){
    if(i==1)
        return 0;
    if(i==2)
        return V(i);
    OPT=max(OPTROB(i-2)+V(i),OPTROB(i-1));
    return OPT;
}

OPTROBCIRCLE(n){
    FIRST=OPTROBFIRST(n-1);
    LAST=OPTROBLAST(n);
    return max(FIRST,LAST);
}
```

算法的正确性：这一问只是用第一问的方法先求两个子问题，然后再合并即可，并没有影响算法的正确性。

时间复杂度分析：两个递归加一次归并，但是递归的复杂度只有一层，T(n)=2T(n-1)+1, 因此 T(n)=O(n).

题目三：Decoding

算法：输入字符串长度为 0 或者输入字符串第一个字符为 0 时，终止求解。对于一般情况，我们可以把问题分解成两个子问题，由下到上、由小到大求解。

$$DP方程: num(i) = num(i-1) + num(i-2).$$

伪代码：

```
DecodingWays(String s){
    n=length(s);
    if(n==0||s[0]=='0')
        retrun 0;
    num[n]=0;//计数数组
    num[0]=1;
    for i in 1:n{
        if 10 <= int(s[i-1:i+1]) <= 26:
            if i >= 2
                num[i] += num[i - 2];
            else
                num[i] += 1;
        if s[i] != '0'
            num[i] += num[i - 1];
    }
    return num[n - 1];
}
```

算法的正确性：由于我们将原问题分为两个子问题，只用一次 for 循环就遍历完整个字符串，子问题有小到大依次求解，方法合理简单。
时间复杂度分析：只用了一次循环, 因此 T(n)=O(n).

题目五：Maximum profit of transactions

算法：我们定义一个 dp[n]数组，定义遍历过程中的最大值 max_price 和最小值 min_price, 用 dp 数组记录遍历过程中的子问题与最低价最大收益。第一遍从左到右遍历，得到 dp 数组。 第二遍从右到左遍历，用 max_profit 记录最高价的最大收益。如果收益增加则更新全局最大收益，遍历结束即得结果。
伪代码：

```
max_profit(prices){
    n = len(prices);
    if (not prices || n < 2)
        return 0;
    min_price = prices[0]
    dp[n] = 0;
    for (i in 1:n){
        dp[i] = max(dp[i - 1], prices[i] - min_price);
        min_price = min(prices[i], min_price);
    }

    max_price = prices[n - 1];
    ans = dp[n - 1];
    max_profit = 0;
    for (i in n-1:0){
        max_profit = max(max_profit, max_price - prices[i]);
        max_price = max(max_price, prices[i]);
        ans = max(ans, max_profit + dp[i - 1]);
    }

    return ans;
}
```

算法的正确性：我们将数组逐个遍历，用了两个 for 循环遍历完整个字符串，对子问题最优求解，然后逐个比较，从而寻得全局最优解。
时间复杂度分析：用了两次循环,但是并没有嵌套，T(n)=2T(n), 因此 T(n)=O(n).