Algorithm Second Homework

Zhang Yuan

Student ID: 2015E8013261189

1 Statement

All codes are accepted by the corresponding problems on Leetcode.

2 Money Robbing in a list

2.1 The optimal sub problems and DP equation

$$MostMoney[i] = max\{MostMoney[i-2] + Money[i], MostMoney[i-1]\}$$

2.2 The explain and code

If we decide to choose the nth item of the list, then we can not choose the (n-1)th item, and in the meantime we should choose the maximum money of the first n-2 items, compared to the maximum money of the first n-1 items, we can find the maximum money of n items.

```
class Solution(object):
    def rob(self, nums):
        """
            :type nums: List[int]
            :rtype: int
            """
        Len=len(nums)
        if Len==0:
            return 0
        A=[]
        for i in range(Len):
            A.append(0)

        for j in range(Len):
            if j==0:
                A[j]=nums[j]
            else:
                A[j]=max(A[j-1],A[j-2]+nums[j])

        return A[-1]
```

2.3 Prove the correctness

Cut And Paste Proof:

If there are some ways MostMoney[i-1] or MostMoney[i-2] better than current cases, then the MostMoney[i] must have a higher result which leads to a contradiction.

2.4 Analyze the complexity

Time Complexity: O(n) using bottom-up method.
Space Complexity: O(n).

3 Money Robbing in a circle

3.1 The optimal sub problems and DP equation

Find an item i in circle, split the circle into a list called Money[i+1:i-1].

$$MostMoneyCircle = max\{Money[i]+MostMoneyList[i+2 : i-2], MostMoneyList[i+1 : i-1]\}$$

3.2 The explain and code

If we decide to choose the ith item, then we should not choose (i-1)th item and the (i+1)th item, other items are in a list, if we do not choose the ith item, then other items are in a list too. We can use above method to solve these sub problems.

```
class Solution(object):
    def rob(self,nums):
        """
            :type nums: List[int]
            :rtype: int
            """
        if len(nums)==0:
            return 0
        def rob_list(nums):
            Len=len(nums)
            if Len==0:
                return 0
            A=[]
            for i in range(Len):
                A.append(0)

            for j in range(Len):
                if j==0:
                    A[j]=nums[j]
                else:
                    A[j]=max(A[j-1],A[j-2]+nums[j])

            return A[-1]
```

```
        return max(rob_list(nums[2:-1])+nums[0],rob_list(nums[1:]))
```

3.3 Prove the correctness

Cut And Paste Proof:

If there are better ways MostMoneyList[i+1:i-1] or MostMoneyList[i+2,i-2], then the MostMoneyCircle must have a higher result which leads to a contradiction.

3.4 Analyze the complexity

Time Complexity: O(n) using bottom-up method.
Space Complexity: O(n).

4 Minimum path sum

4.1 The optimal sub problems and DP equation

$$A[i][j] = min\{A[i-1][j-1] + Value[i][j], A[i-1][j] + Value[i][j]\}$$

$$MinPathSum = min\{A[i][j]\}, 0 \leq j < i$$

4.2 The explain and code

The minimum path sum of current node depends on the minimum path sum of above level's neighbor nodes.

```
class Solution(object):
    def minPathSum(self, grid):
        """
            :type grid: List[List[int]]
            :rtype: int
            """
        if len(grid)==0:
            return 0

        A=[]
        for i in range(len(grid)):
            A_son=[]
            for j in range(i+1):
                A_son.append(0)

            A.append(A_son)

        print A

        for i in range(len(grid)):
            for j in range(i+1):
```

```
                if i==0 and j==0:
                    A[i][j]=grid[i][j]
                elif j==0:
                    A[i][j]=A[i-1][j]+grid[i][j]
                elif j==i:
                    A[i][j]=A[i-1][j-1]+grid[i][j]
                else:
                    A[i][j]=min(A[i-1][j-1],A[i-1][j])+grid[i][j]

        print A
```

4.3 Prove the correctness

Cut And Paste Proof:

If there are better ways A[i-1][j-1] or A[i-1][j], then the minimum path sum must have a lower result which leads to a contradiction.

4.4 Analyze the complexity

Note:i is the height of the triangle.
Time Complexity:$O(i^2)$ using bottom-up method.
Space Complexity:$O(i^2)$.

5 Partition

5.1 The optimal sub problems and DP equation

f[i][j] illustrates the cases that s[i-j:i] is palindrome.

$$f[i][j] = \begin{cases} 1, & j = 1 \\ 1, & j = 2 \quad and \quad s[i] = s[i-1] \\ 1, & j = 3 \quad and \quad s[i] = s[i-2] \\ 1, & f[i-1][j-2] = 1 \quad and \quad s[i] = s[i-j] \\ 0, & otherwise \end{cases}$$

st[i] illustrates the min cut number of s[0:i]. Array key saves all positions flag when f[i][flag]=1.

$$st[i] = \begin{cases} 0, & i = 0 \\ 0, & key[k] = i \\ min(1 + st[i - key[k]]), & otherwise \end{cases}$$

5.2 The explain and code

```
class Solution(object):
    def minCut(self, s):
        """
            :type s: str
```

```
    :rtype: int
    """
s1=[]
for i in range(len(s[0])):
    s1.append(s[0][i])
print s1

f=[]
if len(s1)==0:
    return 0

if len(s1)==1:
    return 0


def Return(list):
    key=[]
    for i in range(len(list)):
        if list[i]==1:
            key.append(i)
    print "Key:",key
    return key

st=[]
key1=[]
for i in range(len(s1)):
    f_son=[]
    for j in range(i+1):
        if i==0:
            f_son.append(1)
        else:
            if j==0:
                f_son.append(1)
            elif j==1:
                if s1[i]==s1[i-1]:
                    f_son.append(1)
                else:
                    f_son.append(0)
            elif j%2==0:
                if j==2:
                    if f[i-1][0]==1 and s1[i]==s1[i-2]:
                        f_son.append(1)
                    else:
                        f_son.append(0)
                else:
                    if f[i-1][j-2]==1 and s1[i]==s1[i-j]:
                        f_son.append(1)
                    else:
```

```
                        f_son.append(0)
                else:
                    if f[i-1][j-2]==1 and s1[i]==s1[i-j]:
                        f_son.append(1)
                    else:
                        f_son.append(0)

        f.append(f_son)
        if i==0:
            st.append(0)
        else:
            key1=Return(f[i])
            value=i+1
            for k in range(len(key1)):
                if key1[k]==i:
                    value=0
                else:
                    if value>1+st[i-key1[k]-1]:
                        value=1+st[i-key1[k]-1]
            st.append(value)

    print "st:",st
    return st[-1]
```

5.3 Prove the correctness

Cut And Paste Proof:

If there are some ways st[i-key[k]] better than current cases, we can obtain a lower min cut using the way which leads to a contradiction.

5.4 Analyze the complexity

Time Complexity: $O(n^2)$ using bottom-up method.
Space Complexity: $O(n^2)$.

6 Sub sequence counting

6.1 The optimal sub problems and DP equation

$$f[i][j] = \begin{cases} 1, & i = 0 \\ 0, & j = 0 \\ f[i-1][j-1] + f[i][j-1], & t[i-1] = s[j-1] \\ f[i][j-1], otherwise \end{cases}$$

6.2 The explain and code

When we find t[i-1]=s[j-1], the sub sequence counts depends on the sub case's counts, which we save it in f[i-1][j-1] and f[i][j-1].

```
class Solution(object):
    def numDistinct(self, s, t):
        """
            :type s: str
            :type t: str
            :rtype: int
            """
        s1=s.encode("ascii")

        t1=t.encode("ascii")

        map=[]

        for i in range(len(t1)+1):
            map_son=[]
            for j in range(len(s1)+1):
                if i==0:
                    map_son.append(1)
                elif j==0:
                    map_son.append(0)
                elif t1[i-1]==s1[j-1]:
                    value=map[i-1][j-1]+map_son[-1]
                    map_son.append(value)
                else:
                    map_son.append(map_son[-1])
            map.append(map_son)
        print map
```

6.3 Analyze the complexity

Time Complexity: O(ST) using bottom-up method.
Space Complexity: O(ST).

7 Decoding

7.1 The optimal sub problems and DP equation

$$sum[i] = \begin{cases} 1, & i = 0 \\ 1, & s[i] = 0 \quad and \quad i < 2 \\ sum[i-2], & s[i] = 0 \\ sum[i-1], & s[i-1] = 0 \\ 1, & i = 1 \quad and \quad s[i-1:i] \quad not \quad in \quad [10, 27] \\ 2, & i = 1 \quad and \quad s[i-1:i] \quad in \quad [10, 27] \\ sum[i-1], & s[i-1:i] \quad not \quad in \quad [10, 27] \\ sum[i-1] + sum[i-2], & s[i-1:i] \quad in \quad [10, 27] \end{cases}$$

7.2 The explain and code

The all cases are a little complex, we must take 0 into consideration, all cases are listed in above equations.

```
class Solution(object):
    def numDecodings(self, s):
        """
            :type s: str
            :rtype: int
            """
        if len(s)==0:
            return 0

        for i in range(len(s)):
            if i==0 and s[i]=='0':
                return 0
            elif i!=len(s)-1 and s[i]=='0' and (s[i-1]=='0' or s[i+1]=='0'
                return 0
            elif s[i]=='0' and s[i-1]!='1' and s[i-1]!='2':
                return 0
            else:
                pass

        sum=[]
        for i in range(len(s)):
            sum.append(0)

        for i in range(len(s)):
            if i==0:
                sum[i]=1
            elif s[i]=="0":
                if i-2<0:
                    return 1
                sum[i]=sum[i-2]
```

```
            elif s[i-1]=="0":
                sum[i]=sum[i-1]
            elif i==1:
                key=int(s[i-1])*10+int(s[i])
                if key>0 and key<27:
                    sum[i]=2
                else:
                    sum[i]=1
            else:
                key=int(s[i-1])*10+int(s[i])
                if key>0 and key<27:
                    sum[i]=sum[i-1]+sum[i-2]
                else:
                    sum[i]=sum[i-1]
        return sum[-1]
```

7.3 Analyze the complexity

Time Complexity: O(n).
Space Complexity: O(n).

8 Maximum profit of transactions

8.1 The optimal sub problems and DP equation

If transaction=1:

$$minP = min(prices[i-1], minP)$$

$$SingleProfit = max(profit, prices[i] - minP)$$

If transaction at most two:

$$profit = max\{SingleProfit[1:i] + SingleProfit[i:n]\}$$

8.2 The code

```
//
//  main.cpp
//  BuyAndSellStock
//
//  Created by zhang yuan on 15/10/19.
//  Copyright © 2015 zhang yuan. All rights reserved.
//

#include <iostream>
#include <vector>

using namespace std;
```

```
//Solution1 try to find the most profit through only one transaction
class Solution1 {
public:
    int maxProfit(vector<int>& prices) {
        if(prices.size() <= 1) {
            return 0;
        }

        int minP = prices[0];

        int profit = prices[1] - prices[0];

        for(int i = 2; i < prices.size(); i++) {
            minP = min(prices[i - 1], minP);
            profit = max(profit, prices[i] - minP);
        }

        if(profit < 0) {
            return 0;
        }

        return profit;
    }
};

//Solution2 try to find the most profit without transaction counts restrict
class Solution2 {
public:
    int maxProfit(vector<int> &prices) {
        int len = (int)prices.size();
        cout<<len<<endl;
        if(len <= 1) {
            return 0;
        }

        int sum = 0;
        for(int i = 1; i < len; i++) {
            if(prices[i] - prices[i - 1] > 0) {
                sum += prices[i] - prices[i - 1];
            }
        }

        return sum;
    }
};

//Solution try to find the most profit at most two transaction
```

```
class Solution{
public:
    int maxProfit(vector<int> &prices)
    {
        int len=int(prices.size());
        cout<<"Len:"<<len<<endl;
        if (len<=1)
        {
            return 0;
        }

        //Try to find the most profit through only one transaction, we can
        int minP=prices[0];

        vector<int> profits_forward(0);
        if (prices[1]-prices[0]>0)
        {
            profits_forward.push_back(prices[1]-prices[0]);
        }
        else{
            profits_forward.push_back(0);
        }


        for (int i=2; i<len; i++) {
            minP=min(prices[i-1],minP);
            profits_forward.push_back(max(profits_forward[profits_forward.s
        }

        //Try to find the most profit at most two transaction, we can obta

        vector<int> profits_backward(0);
        if(prices[len-1]-prices[len-2]>0)
        {
            profits_backward.push_back(prices[len-1]-prices[len-2]);
        }
        else{
            profits_backward.push_back(0);
        }

        int maxP=prices[len-1];
        int Best_Profit=0;

        if (len==2)
        {
        if(prices[1]-prices[0]>0)
        {
            return prices[1]-prices[0];
```

```
        }
            else{
                return 0;
            }
        }
        int Profit_Sum;

        for (int i=len-3; i>-1; i--) {
            maxP=max(prices[i+1],maxP);
            int key=max(profits_backward[profits_backward.size()-1],maxP-pr
            profits_backward.push_back(key);
            if(profits_backward.size()<len-1)
            {
                Profit_Sum=profits_backward[profits_backward.size()-1]+pro
            }
            else{
                Profit_Sum=profits_backward[profits_backward.size()-1];
            }
            if (Best_Profit<Profit_Sum)
            {
                Best_Profit=Profit_Sum;
            }

        }

        return Best_Profit;



    }
};

int main(int argc, const char * argv[]) {
    // insert code here...
    std::cout << "Hello, World!\n";
    int a[6] = {5,4,2,1,5,7};
    vector<int> v(a,a+6);
    Solution s=Solution();
    int k=s.maxProfit(v);
    cout<<k<<endl;
    return 0;
}
```