# Problem Solving Homework(Week 12)

161180162 Xu Zhiming

May 19, 2017

## TC

### 15.1-1

*Proof.* Mathematic Induction:
Base case: $T(0) = 1$, $T(1) = 1 + T(0) = 2$.
Suppose for all $n \leq k$, $T(k) = 2^k$, then we need to prove $T(k+1) = 2^{k+1}$.

$$\begin{aligned}
T(k+1) = 1 + \sum_{j=0}^{k} &= 1 + (T(0) + T(1) + T(2) + \cdots + T(k)) \\
&= 1 + (1 + 2 + \cdots + 2^k) \\
&= 1 + \frac{1 - 2^{k+1}}{1 - 2} \\
&= 1 + 2^{k+1} - 1 \\
&= 2^{k+1}
\end{aligned}$$

□

### 15.1-3

```
REVISED-CUT-ROT()
let r[0..n] and s[0..n] be new arrays
r[0]=0
for j=1 to n
        q=-INF
        is_cut=false
        for i=1 to j
                if q<p[i]+r[j-i]-c      //c is the cutting cost
                        q=p[i]+r[j-i]-c
                        s[j]=i
                if j==i
                        if      q<=p[i]+r[j-i]&&is_cut
                                s[j]=i
        r[j]=q
return r[n] and s[n]
```

### 15.2-2

```
MATRIX_CHAIN_MULTIPLY(A,s,i,j)
if j==i
        return A[i]
if j==i+1
        return Matrix_MULTIPLY(A[i],A[j])
Matrix M1=MATRIX_CHAIN_MULTIPLY(A,s,i,s[i][j])
Matrix M2=MATRIX_CHAIN_MULTIPLY(A,s,s[i][j]+1,j)
return Matrix_MULTIPLY(t1,t2)
```

**15.2-4**

$n^2$ vertexes and $n^3$ sides connecting them.

**15.3-3**

Yes, it can be proved by CUT-PASTE method.

**15.3-5**

*Proof.* The overall best solution may not be the local best solution. For example, if in a local best solution A ,it needs to cut the rod to $l_i - 1$ i-length small pieces. But in another local best solution B, it also needs to cut the rod to more than one i-length pieces. When combining these two solutions, it exceeds the maximum number of all i-length pieces. So the combination is not appropriate and shouldn't be a solution. □

**15.3-6**

The first case, when $c_k = 0$ for all $k = 1, 2, \ldots, n$:

*Proof.* Suppose a set $E_{1n}$, which means one optimal exchange way. Let's say $E_{in}$ consists of a number of order pairs $\{e_{1i_1}, e_{i_1i_2}, \ldots, e_{i_kn}\}$, $a_{mn}$ means an exchange from Currency m(denoted as $C_m$) to Currency n($C_n$). Obviously $card(E) \leq n - 1$ (Since we don't need to exchange for a certain kind of currency for more than one time) Then we consider one pair of exchanging, namely $e_{pq}$ and $e_{qr}$. From these two exchange, our currency turn into $C_r$ from $C_p$. We will prove this exchange is optimal using cut-and-paste. If we have a better way $OpE_{pr}$ to exchange $C_p$ to $C_r$, we can use that. Since the other ways are thought to be optimal, substitute $e_{pq}$ and $e_{qr}$ with $OpE_{pr}$ will generate a better global optimal solution $E_{in}^{'}$, contradicting our assumption. Therefore, it exhibits optimal substructure. □

The second case, when the commissions $c_k$ are arbitrary values:

*Proof.* Consider an extreme scenario, where $c_m = x$ and $c_{m+1} = \infty$. Still use the assumptions in case 1. First, we don't take the commissions into account, we generate a global optimal solution $E_{in}$, which requires $m+1$ exchanges. Then we reconsider it and find that the commission for $m + 1$ times exchange is far too high. So we need to make changes to it. We directly exchange $C_p$ to $C_r$, obtaining a solution $E_{in}^{''}$ better than $E_{in}$. This solution is obviously not locally optimal. Therefore, it doesn't exhibit optimal substructure. □

**15.4-3**

```
REVISED-LCS-LENGTH(X,Y)
m=X.length
n=Y.length
let DP[0..m][0..n] be a new array
for i=0 to m
        DP[i][0]=0
for i=0 to n
        DP[0][i]=0
for i=1 to m
        for j=1 to n
                if X[i]==Y[j]
                        DP[i][j]=DP[i-1][j-1]+1
                else
                        DP[i][j]=max{DP[i-1][j],DP[i][j-1]}
return DP[m][n]
```

**15.4-5**

```
LONGEST_ASCENDING_STRING(S)
let R be a new array
R=S
len=S.length
QUICKSORT(S,1,len)
return REVISED-LCS-LENGTH(S,R)
```

**15.5-1**

```cpp
#include <iostream>
using namespace std;
const int MAX = 9999;
const int n = 5;
double p[n + 1] = { -1,0.15,0.1,0.05,0.1,0.2 };
double q[n + 1] = { 0.05,0.1,0.05,0.05,0.05,0.1 };
int root[n + 1][n + 1];//
double w[n + 2][n + 2];//
double e[n + 2][n + 2];//
void Optimal_BST(double *p, double *q, int n)
{
        for (int i = 1; i <= n + 1; ++i)
        {
                w[i][i - 1] = q[i - 1];
                e[i][i - 1] = q[i - 1];
        }
        for (int len = 1; len <= n; ++len)
        {
                for (int i = 1; i <= n - len + 1; ++i)
                {
                        int j = i + len - 1;
                        e[i][j] = MAX;
                        w[i][j] = w[i][j - 1] + p[j] + q[j];
                        for (int k = i; k <= j; ++k)
                        {
                                double temp = e[i][k - 1] + e[k + 1][j] + w[i][j];
                                if (temp < e[i][j])
                                {
                                        e[i][j] = temp;
                                        root[i][j] = k;
                                }
                        }
                }
        }
}
void Print()
{
        cout << "The roots: " << endl;
        for (int i = 1; i <= n; ++i)
        {
                for (int j = 1; j <= n; ++j)
                {
                        cout << root[i][j] << " ";
                }
                cout << endl;
        }
        cout << endl;
}
void PrintOptimalBST(int i, int j, int r)
{
        int rootChild = root[i][j];
        if (rootChild == root[1][n])
        {
                cout << "k" << rootChild << " is the root" << endl;
                PrintOptimalBST(i, rootChild - 1, rootChild);
                PrintOptimalBST(rootChild + 1, j, rootChild);
```

```
			return;
		}
		if (j < i - 1)
			return;
		else if (j == i - 1)
		{
			if (j < r)
				cout << "d" << j << " is" << " k" << r << "'s left child" << endl;
			else
				cout << "d" << j << " is" << " k" << r << "'s right child" << endl;
			return;
		}
		else
		{
			if (rootChild < r)
				cout << "k" << rootChild << " is" << " k" << r << "'s left child" <<
			else
				cout << "k" << rootChild << " is" << " k" << r << "'s right child" <<
		}
		PrintOptimalBST(i, rootChild - 1, rootChild);
		PrintOptimalBST(rootChild + 1, j, rootChild);
}
int main()
{
		Optimal_BST(p, q, n);
		Print();
		cout << "The optimal BST: " << endl;
		PrintOptimalBST(1, n, -1);
}
```

**15-4**

```
#include <iostream>
#include <cstring>
using namespace std;
const int INT_MAX=1<<30
int GIVE_LINES(int p[], int j,char *str[]);
void PRINT_NEATLY (int l[],char *str[],int n,int M)
{
		int **remaining,i,j;
		remaining=new int*[n+1];
		for ( i=0;i<=n;i++)
			extras[i]=new int[n+1];
		int **price;
		price=new int*[n+1];
		for ( i=0;i<=n;i++)
			lc[i]=new int[n+1];
		int *c=new int[n+1];
		int *p=new int[n+1];
		for (i = 1; i <= n; i++)
			remaining[i][i] = M - l[i-1];//
			for (j = i+1; j <= n; j++)
				remaining[i][j] = remaining[i][j-1] - l[j-1] - 1;
		for (i = 1; i <= n; i++)
			for (j = i; j <= n; j++)
			{
				if (remaining[i][j] < 0)
					price[i][j] =INF;
```

```
                else if (j == n && remaining[i][j] >= 0)
                    price[i][j] = 0;
                else
                    price[i][j] = remaining[i][j]*remaining[i][j]*remaining[i][j]
            }
    c[0] = 0;
    for (j = 1; j <= n; j++)
    {
        c[j] = INF;
        for (i = 1; i <= j; i++)
            if (c[i-1] != INF&&lc[i][j] != INF && (c[i-1] + lc[i][j] < c[j]))
            {
                c[j] = c[i-1] + lc[i][j];
                p[j] = i;
            }

    }
    GIVE_LINES(p, n, str);
}

int GIVE_LINES(int p[], int j,char *str[])
{
    int k,i=p[j];
    if (i == 1)
        k = 1;
    else
        k = GIVE_LINES (p, i-1,str) + 1;
    cout<<"Line No."<<k<<" ";
    for (int t=i;t<=j;t++)
    {
        cout<<str[t-1]<<" ";
    }
    cout<<endl;
    return k;
}
int main()
{
    const int n=10;
    const int M=8;
    char* str[n]={"abc","CS","so","qwert","a","opaque","knight","tree","Ace","fgo"};
    int l[n]={0};
    for (int i=0;i<n;i++)
    {
        l[i]=strlen(str[i]);
    }
    PRINT_NEATLY (l,str,n,M);
    return 0;
}
```

The time cost is $O(n^2)$, where $n$ is length of the sequence.