Problem Set 7

吴政亿 151220129 南大计科

Problem 7.1

定义一个二维数组 temp, 其中 temp[i][j]表示前 i 个数字中是否存在子集和为 j 的子集。那么可以得到一个状态转移方程 temp[i][j] = temp[i-1][j] | temp[i-1][j-A[i]]。

```
bool Algorithm_7_1(int A[n], int S){
    bool temp[1……n][1……S];
    for(int i=1;i<=n;i++){
        for(int j=1;j<=S;j++){
            if(i == 1)
                if(A[i] == j)
                    temp[i][j] = true;
                Else
                    temp[i][j] = false;

            else{
                if(A[i] > j)
                    temp[i][j] = temp[i-1][j];
                Else
                    temp[i][j] = temp[i-1][j] | temp[i-1][j-A[i]];
            }
        }
    }
    return temp[n][S];
}
```

Problem 7.2

定义一个一维数组 temp，那么 temp[i]就表示变为将 i 变为 1 最少需要多少个操作。初始化 temp[1] = 0;首先分析对于任意数字 n，n/3 下降的速度大于 n/2 大于等于 n-1，因此我们采取贪心策略，可以得到一个状态转移方程

```
          temp[i/3] + 1 (i%3 == 0)
temp[i] =  temp[i/2] + 1 (i%3 != 0 && i%2 == 0)
          temp[i-1] + 1 (i%3 != 0 && i%2 != 0)
```

```
bool Algorithm_7_2(int A[n]){
    bool temp[1……n];
    temp[1] = 0;
    for(int i=2;i<=n;i++){
        if(i%3 == 0
            temp[i] = temp[i/3] + 1;
        else if(i%2 == 0)
            temp[i] = temp[i/2] + 1;
```

```
        else
            temp[i] = temp[i-1] + 1;
    }
    return temp[n];
}
```

Problem 7.3

定义两个一维数组 liss[n]与 pre[n]，liss[i]为以 A[i]最大元素的最长非递减子序列的大小，pre[j]为以该元素作为最大元素的递增序列中该元素的前驱节点，得到一个状态转移方程 $\text{liss}[i] = \max\{\text{liss}[k] + 1(k < i \ \&\& \ A[k] \leq A[i]), \text{liss}[m](m < i \ \&\& \ A[m] > A[i])\}$

```
int LISS(int array[n]){
    int i, j, k, max;
    int liss[n];
    int pre[n];

    for(i = 0; i < n; ++i){
        liss[i] = 1;
        pre[i] = i;
    }

    for(i = 1, max = 1, k = 0; i < n; ++i){
        for(j = 0; j < i; ++j){
          if(array[j] <= array[i] && liss[j]+1>liss[i]){
                liss[i] = liss[j] + 1;
                pre[i] = j;

                if(max < liss[i]){
                    max = liss[i];
                    k = i;
                }
            }
        }
    }
    return max;
}
```

Problem 7.5

1. 引进一个二维数组 c[m][n]，用 c[i][j]记录 X[i]与 Y[j] 的 LCS 的长度，b[i][j]记录 c[i][j]是通过哪一个子问题的值求得的，以决定搜索的方向。状态转移方程为：

$$c[i][j] = \begin{cases} 0 & (i = 0 \ || \ j = 0) \\ c[i-1][j-1] + 1 & (i,j > 0 \ \&\& \ x_i = y_i) \\ \text{Max}\{c[i][j-1], c[i-1][j]\} & (i,j > 0 \ \&\& \ x_i \neq y_i) \end{cases}$$

之后只需要先将 c[0][]与 c[][0]初始化为 0，然后两层循环即可。

2. 变化是在 X 中可以重复出现，那么只需要稍加改动状态转移方程即可：

$$c[i][j] = \begin{cases} 0 & (i = 0 \ || \ j = 0) \\ c[i][j-1] + 1 & (i,j > 0 \ \&\& \ x_i = y_i) \\ Max\{c[i][j-1], c[i-1][j]\} & (i,j > 0 \ \&\& \ x_i \neq y_i) \end{cases}$$

3. 只需要定义一个变量 n 用来计算重复的次数即可，init n = 0；

$$c[i][j] = \begin{cases} 0 & (i = 0 \ || \ j = 0) \\ c[i][j-1] + 1 \ \text{并且 n = 0} & (I,j > 0 \ \&\& \ x_i = y_I \ \&\& \ n = k) \\ c[i][j-1] + 1 \ \text{并且 n++} & (i,j > 0 \ \&\& \ x_i = y_i \ \&\& \ n < k) \\ Max\{c[i][j-1], c[i-1][j]\} \ \text{并且 n = 0} & (i,j > 0 \ \&\& \ x_i \neq y_i) \end{cases}$$

Problem 7.6

首先使用 problem 7.5 中的算法求解最长公共子序列的长度为l，最后最短共同超序列的长度为$m + n - l$。

```
int p(char a, char b){
    return (a == b)? 1 : 2;
}

int Algorithm_7_6(char A[m], char B[n]){
    int a[m+1][n+1];
    for(int i = 0; i <= m; i++)
        a[i][0] = i;
    for(int j = 0; j<= n; j++)
        a[0][j] = j;
    for(int i = 1; i <= m; i++){
     for(int j = 1; j<= n; j++)
      a[i][j]=max{[i-1][j]+1,a[i-1][j-1]+p(A[i],B[j]),a[i][j]+1};
    }
    return a[m][n];
}
```

Problem 7.8

引进一个二维数组 c[m][n]，用 c[i][j]记录 T[i]与 T[n-j]为首字母的最长连续子串长度。那么得到状态转移方程：

$$T[i][j] = T[i-1][j-1] + 1$$

```
int Algorithm_7_8(char T[n]){
    int a[n][n];
    for(int i = 1; i < n; i++){
        a[i][1] = (T[i]==T[1])?1 : 0;
        a[1][i] = (T[1]==T[i])?1 : 0;
    }
    for(int i = 2; i <= n; i++){
        for(int j = 2; j + i < n+1; j++)
```

```
            a[i][j] = (T[i]==T[j])? T[i-1][j-1] + 1 : 0;
    }
    return max in a[][];
}
```

Problem 7.10

1. 求给定字符串与它的逆序字符串的最长子序列长度即为答案。
2. 初始化计数变量 count = 0;设字符串 S 的逆序子序列为 S’,

```
While(S 不为空){
    求解 S 与 S’的最长公共连续子序列为 T;
    S = S – T;
    S’ = S’ – T;
    Count++;
}
```

最后 count 值即为最少的回文数量。

Problem 7.11

```
int Algorithm_7_11(int i, int j){
    if(a[i][j] is calculated)
        return a[i][j];
    if(there is no m between i to j)
        a[i][j] = 0;
    else{
        int min = INF;
        for(all m form i to j){
            if(min > a[i][m] + a[m+1][j] + cost(i,j,m))
                min = a[i][m] + a[m+1][j] + cost(i,j,m);
        }
        a[i][j] = min;
    }
    return a[i][j];
}
```

Problem 7.13

定义一个二维数组，a[i][0]代表以 i 为根的子树中，不包括 i 的时候的最小顶点覆盖;a[i][1]为包含了 i 为最小顶点覆盖时的最小值。有状态转移方程：

$$a[i][0] = a[i->lchild][1] + a[i->rchild][1];$$

$$a[i][1] = 1 + \min\{a[i.lchild][1], a[i.lchild][0]\} + \min\{a[i.rchild][1], a[i.rchild][0]\}$$

那么我们初始化叶节点的 a[i][0] = 0;a[i][1] = 1;

```
int calculated_have_root(node u){
    if(u == NULL)
        return 0;
```

```
    if(a[u][1] is not calculated){
        int m1 = min(calculated_have_root(i.lchild),calculate_no_root(l.lchild));
        int m2 = min(calculated_have_root(i.rchild),calculate_no_root(l.rchild));
        a[u][1] = 1 + m1 + m2;
    }
    return a[u][1];
}
int calculate_no_root(node u){
    if(u == NULL)
        return 0;
    if(a[u][0] is not calculated)
        a[u][0] = calculated_have_root(u.lchild)+calculated_have_root(u.rchild);
    return a[u][0];
}
int Algorithm_7_13(V, E, root){
    int a[n][2];
    calculate_no_root(root);
    calculate_have_root(root);
    return min(a[n][0],a[n][1]);
}
```

Problem 7.15

定义一个一维数组 a[n]，其中 a[i]为到达旅店 i 时受到的最小的惩罚和，另外再定义一个一维数组 pre[n]用来表示前驱旅馆。

```
int Algorithm_7_15(int A[n]){
    int a[n], pre[n];
    for(int i=1;i<=n;i++){
        a[i] = INF;
        pre[i] = -1;
    }
    a[1] = 0;
    for(int i=2;i<=n;i++){
        for(int j=i-1;A[i]-A[j]<=200;j--){
            if(a[j] + (200-A[i]+A[j])*(200-A[i]+A[j]) < a[i]){
                a[i] = a[j] + (200-A[i]+A[j])*(200-A[i]+A[j]);
                pre[i] = j;
            }
        }
    }
    return a[n];
}
```

Problem 7.16

```
int Algorithm_7_16(int m[n], int p[n]){
    int a[n];
    for(int i=1;i<=n;i++)
        a[i] = 0;
    a[1] = 0;
    for(int i=2;i<=n;i++)
        for(int j=i-1;m[i]-m[j]<=k;j--)
            if(a[j] + p[i] > a[i])
                a[i] = a[j] + p[i];
    return a[n];
}
```

Problem 7.17

```
int Algorithm_7_17(){
    int a[n][n];
    for(int i=n;i>=1;i--)
      for(int j=n;j>=1;j--){
        a[i][j]=score[i][j];
        if(i == n && j == n)
          continue;
        else if(i == n && j != n)
          a[i][j] = max{a[i][j+1] + a[i][j], a[i][j]};
        else if(i != n && j == n)
          a[i][j] = max{a[i+1][j] + a[i][j], a[i][j]};
        else
          a[i][j] = max{a[i+1][j] + a[i][j], a[i][j+1] + a[i][j]};
        }
}
```