**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]]。

|  |  |
| --- | --- |
| 1 | *bool* Algorithm\_7\_1(*int* *A*[n], *int* *S*){ |
| 2 | *bool* temp[1……n][1……S]; |
| 3 | for(*int* i=1;i<=n;i++){ |
| 4 | for(*int* j=1;j<=S;j++){ |
| 5 | if(i == 1) |
| 6 | if(A[i] == j) |
| 7 | temp[i][j] = true; |
| 8 | Else |
| 9 | temp[i][j] = false; |
| 10 |  |
| 11 | else{ |
| 12 | if(A[i] > j) |
| 13 | temp[i][j] = temp[i-1][j]; |
| 14 | Else |
| 15 | temp[i][j] = temp[i-1][j] | temp[i-1][j-A[i]]; |
| 16 | } |
| 17 | } |
| 18 | } |
| 19 | return temp[n][S]; |
| 20 | } |

**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)

|  |  |
| --- | --- |
| 1 | *bool* Algorithm\_7\_2(*int* *A*[n]){ |
| 2 | *bool* temp[1……n]; |
| 3 | temp[1] = 0; |
| 4 | for(*int* i=2;i<=n;i++){ |
| 5 | if(i%3 == 0 |
| 6 | temp[i] = temp[i/3] + 1; |
| 7 | else if(i%2 == 0) |
| 8 | temp[i] = temp[i/2] + 1; |
| 9 | else |
| 10 | temp[i] = temp[i-1] + 1; |
| 11 | } |
| 12 | return temp[n]; |
| 13 | } |

**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

|  |  |
| --- | --- |
| 1 | *int* LISS(*int* *array*[n]){ |
| 2 | *int* i, j, k, max; |
| 3 | *int* liss[n]; |
| 4 | *int* pre[n]; |
| 5 |  |
| 6 | for(i = 0; i < n; ++i){ |
| 7 | liss[i] = 1; |
| 8 | pre[i] = i; |
| 9 | } |
| 10 |  |
| 11 | for(i = 1, max = 1, k = 0; i < n; ++i){ |
| 12 | for(j = 0; j < i; ++j){ |
| 13 | if(array[j] <= array[i] && liss[j]+1>liss[i]){ |
| 14 | liss[i] = liss[j] + 1; |
| 15 | pre[i] = j; |
| 16 |  |
| 17 | if(max < liss[i]){ |
| 18 | max = liss[i]; |
| 19 | k = i; |
| 20 | } |
| 21 | } |
| 22 | } |
| 23 | } |
| 24 | return max; |
| 25 | } |

**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：

0

c[i][j] = c[i-1][j-1] + 1

Max{c[i][j-1],c[i-1][j]}

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。

1. 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：

0

c[i][j] = c[i][j-1] + 1

Max{c[i][j-1],c[i-1][j]}

1. 只需要定义一个变量n用来计算重复的次数即可，init n = 0；

0

c[i][j] = c[i][j-1] + 1 并且n = 0

c[i][j-1] + 1 并且n++

Max{c[i][j-1],c[i-1][j]} 并且n = 0

**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，最后最短共同超序列的长度为。

|  |  |
| --- | --- |
| 1 | *int* p(*char* *a*, *char* *b*){ |
| 2 | return (a == b)? 1 : 2; |
| 3 | } |
| 4 |  |
| 5 | *int* Algorithm\_7\_6(*char* *A*[m], *char* *B*[n]){ |
| 6 | *int* a[m+1][n+1]; |
| 7 | for(*int* i = 0; i <= m; i++) |
| 8 | a[i][0] = i; |
| 9 | for(*int* j = 0; j<= n; j++) |
| 10 | a[0][j] = j; |
| 11 | for(*int* i = 1; i <= m; i++){ |
| 12 | for(*int* j = 1; j<= n; j++) |
| 13 | a[i][j]=max{[i-1][j]+1,a[i-1][j-1]+p(A[i],B[j]),a[i][j]+1}; |
| 14 | } |
| 15 | return a[m][n]; |
| 16 | } |

**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：

|  |  |
| --- | --- |
| 1 | *int* Algorithm\_7\_8(*char* *T*[n]){ |
| 2 | *int* a[n][n]; |
| 3 | for(*int* i = 1; i < n; i++){ |
| 4 | a[i][1] = (T[i]==T[1])?1 : 0; |
| 5 | a[1][i] = (T[1]==T[i])?1 : 0; |
| 6 | } |
| 7 | for(*int* i = 2; i <= n; i++){ |
| 8 | for(*int* j = 2; j + i < n+1; j++) |
| 9 | a[i][j] = (T[i]==T[j])? T[i-1][j-1] + 1 : 0; |
| 10 | } |
| 11 | return max in a[][]; |
| 12 | } |

**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**

|  |  |
| --- | --- |
| 1 | *int* Algorithm\_7\_11(*int* *i*, *int* *j*){ |
| 2 | if(a[i][j] is calculated) |
| 3 | return a[i][j]; |
| 4 | if(there is no m between i to j) |
| 5 | a[i][j] = 0; |
| 6 | else{ |
| 7 | *int* min = INF; |
| 8 | for(all m form i to j){ |
| 9 | if(min > a[i][m] + a[m+1][j] + cost(i,j,m)) |
| 10 | min = a[i][m] + a[m+1][j] + cost(i,j,m); |
| 11 | } |
| 12 | a[i][j] = min; |
| 13 | } |
| 14 | return a[i][j]; |
| 15 | } |

**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：

那么我们初始化叶节点的a[i][0] = 0;a[i][1] = 1;

|  |  |
| --- | --- |
| 1 | *int* calculated\_have\_root(node *u*){ |
| 2 | if(u == NULL) |
| 3 | return 0; |
| 4 | if(a[u][1] is not calculated){ |
| 5 | *int* m1 = min(calculated\_have\_root(i.lchild),calculate\_no\_root(l.lchild)); |
| 6 | *int* m2 = min(calculated\_have\_root(i.rchild),calculate\_no\_root(l.rchild)); |
| 7 | a[u][1] = 1 + m1 + m2; |
| 8 | } |
| 9 | return a[u][1]; |
| 10 | } |
| 11 | *int* calculate\_no\_root(node *u*){ |
| 12 | if(u == NULL) |
| 13 | return 0; |
| 14 | if(a[u][0] is not calculated) |
| 15 | a[u][0] = calculated\_have\_root(u.lchild)+calculated\_have\_root(u.rchild); |
| 16 | return a[u][0]; |
| 17 | } |
| 18 | *int* Algorithm\_7\_13(*V*, *E*, *root*){ |
| 19 | *int* a[n][2]; |
| 20 | calculate\_no\_root(root); |
| 21 | calculated\_have\_root(root); |
| 22 | return min(a[n][0],a[n][1]); |
| 23 | } |

**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。

|  |  |
| --- | --- |
| 1 | *int* Algorithm\_7\_15(*int* *A*[n]){ |
| 2 | *int* a[n], pre[n]; |
| 3 | for(*int* i=1;i<=n;i++){ |
| 4 | a[i] = INF; |
| 5 | pre[i] = -1; |
| 6 | } |
| 7 | a[1] = 0; |
| 8 | for(*int* i=2;i<=n;i++){ |
| 9 | for(*int* j=i-1;A[i]-A[j]<=200;j--){ |
| 10 | if(a[j] + (200-A[i]+A[j])\*(200-A[i]+A[j]) < a[i]){ |
| 11 | a[i] = a[j] + (200-A[i]+A[j])\*(200-A[i]+A[j]); |
| 12 | pre[i] = j; |
| 13 | } |
| 14 | } |
| 15 | } |
| 16 | return a[n]; |
| 17 | } |

**Problem 7.16**

|  |  |
| --- | --- |
| 1 | *int* Algorithm\_7\_16(*int* *m*[n], *int* *p*[n]){ |
| 2 | *int* a[n]; |
| 3 | for(*int* i=1;i<=n;i++) |
| 4 | a[i] = 0; |
| 5 | a[1] = 0; |
| 6 | for(*int* i=2;i<=n;i++) |
| 7 | for(*int* j=i-1;m[i]-m[j]<=k;j--) |
| 8 | if(a[j] + p[i] > a[i]) |
| 9 | a[i] = a[j] + p[i]; |
| 10 | return a[n]; |
| 11 | } |

**Problem 7.17**

|  |  |
| --- | --- |
| 1 | *int* Algorithm\_7\_17(){ |
| 2 | *int* a[n][n]; |
| 3 | for(*int* i=n;i>=1;i--) |
| 4 | for(*int* j=n;j>=1;j--){ |
| 5 | a[i][j]=score[i][j]; |
| 6 | if(i == n && j == n) |
| 7 | continue; |
| 8 | else if(i == n && j != n) |
| 9 | a[i][j] = max{a[i][j+1] + a[i][j], a[i][j]}; |
| 10 | else if(i != n && j == n) |
| 11 | a[i][j] = max{a[i+1][j] + a[i][j], a[i][j]}; |
| 12 | else |
| 13 | a[i][j] = max{a[i+1][j] + a[i][j], a[i][j+1] + a[i][j]}; |
| 14 | } |
| 15 | } |