第三次作业

分析题

3-1

对于数组$arr[1:n]$设$num[i]$表示以$arr[i]$为结尾元素的最长递增子序列的长度。

则对于$num[k]$，其子问题有两种情况，满足最优子结构：

1. 当$arr[k] < min(arr[1:k-1])$， 即此时$arr[k]$为前k个元素中最小的，此时若以$arr[k]$为结尾，则最长递增子序列只能为$arr[k]$，此时$num[k]=1$;
2. 当$arr[k] \geq min(arr[1:k-1])$， 即此时$num[k]$最大值由$k$之前比$arr[k]$小的值组成的最长递增子序列加上$arr[k]$组成， 此时， $num[k] = \max_{arr[i] \leq arr[k], 1 \leq i < k}\{num[i]\} + 1$；

对于边界条件，即$num[1]=1$，此时共有n个子问题，对于第k个子问题，又需要遍历$num[j], j=\{1,2,3,...,k-1\}$的值，即得到$num[1:n]$需要$O(n^2)$的时间复杂度。

得到$num[1:n]$之后， 需找到$num[1:n]$的最大值，并对前面元素做一次回溯，得到最长递增子序列，此时其时间复杂度为$O(n)$， 即此问题的最终复杂度为$O(n^2)$。

3-2

对于数组$arr[1:i]$，设k为其最大递增子序列长度，由于此时长度为k的最大递增子序列可存在多个，此时另引入一个数组b，$b[k]$代表长度为k 的最大递增子序列的最小值，目的是为了使得i后面的元素更可能比$b[k]$大，使得k更容易扩展。由于长度为$k-1$的最小值必不大于长度为k的最小值，反之则长度为$k-1$的序列构不成长度为k的序列，即$b[k]$单调递增。

对于第$i+1$个元素，如果$arr[i+1] \geq b[k]$，则$arr[i+1]$可以扩展到$arr[1:i]$的最长递增子序列中，此时$arr[1:i+1]$最大递增子序列长度为$k+1$，其对应的长度为$k+1$的最大递增子序列最小值$b[k+1]=arr[i+1]$，仍满足最大递增子序列要求。

如果$arr[i+1] < b[1]$，即$arr[i+1] = \min(arr[1:i+1])$，此时$b[1]=arr[i+1]$，使得最大递增长度为1时永远是最小值。

对于其它情况，由于$b[k]$有序，可通过二分查找使得$b[j-1] \leq arr[i+1] < b[j], j=2,3,...,k$，此时更新$b[j]=a[i+1]$，其不影响最大递增序列的长度同时又使得$b[j]$此时长度为$j$的最小值，保证了其最优子问题结构。

至此可通过遍历一遍$arr[1:n]$即可找到其最大递增子序列长度，此时需要遍历n次，再通过回溯得到最长递增子序列。由于对于第k次遍历，时间复杂度最大的时二分查找，为$\Omega(\log n)$，即总时间复杂的为$nO(\log n)+O(n)=nO(\log n)$。

设计题

3-1 独立任务最优调度问题

当完成k个任务时，设机器A花费了x时间，此时机器B花费时间的最小值为关于x,k的一个函数，令其为$f(k,x)$。

对于第k个任务，其由第$k-1$个任务与机器A与机器B所花费时间有关，动态性在于第k个任务在机器A还是机器B执行。

1. 若第k个任务在机器A执行，花费了x时间，此时机器B花费时间最小值直接为机器A执行第$k-1$个任务所决定的，在执行第$k-1$任务时，A花费时间为$x-a_k$。即此时$f(k,x)=f(k-1,x-a_k)$;

2. 若第k个任务在机器B执行,此时机器A在执行k任务的时间与执行$k-1$任务时间相同，令其为x，则机器B花费时间最小值为模型完成第$k-1$个任务时间加上机器B执行第k个任务的时间b_k， 即$f(k,x)=f(k-1,x)+b_k$;

由于上述模型满足最优子结构，即可以用动态规划求解，其递归方程为:

$$f(k,x)=\min\{f(k-1,x-a_k),f(k-1,x)+b_k\},k\in\{1,2,3,\dots,n\}$$

由于执行完n个任务时间最多不超过$sum=\min\{\sum_{k=1}^{n}a_k,\sum_{k=1}^{n}b_k\}$，将所有可能时间离散化，对于完成前$n$个任务机器A需要时间$x$, 机器B需要最小时间$f(n,x)$，此时为了确保前$n$个任务完成，则需要取$\max\{x,f(n,x)\}$，由于$x$值未知，则需要对$x$值遍历一遍并取最小值，使得完成$k$个任务所需时间最小，即最终优化目标:

$$\min\{\max\{x,f(n,x)\},x\in\{0,1,2,3,4,\dots,sum\}\}$$

对于$f(k,x)=\min\{f(k-1,x-a_k),f(k-1,x)+b_k\}$，则此时意为前$k$个任务没有在机器A执行，因为花费时间为$x<a_k$，即$f(k,x)=\min\{f(k-1,x-a_k),f(k-1,x)+b_k\}=f(k-1,x)+b_k$，边界条件:

$$f(1,x)=b_1,x<a_1$$
$$f(k,x)=f(k-1,x)+b_k,x<a_k$$

复杂度:

由伪代码知此代码复杂度高的在于递推式，其需要找到n任务，由于对于每个任务，时间需要从0遍历到sum，由于sum由执行时间求得，对于未知输入其亦是位置的，即时间复杂度在$O(n*\min\{\sum_{k=1}^{n}a_k,\sum_{k=1}^{n}b_k\})$。

伪代码:

```
//伪代码
int dynamic_schedule(){
    int a,b;
    int minTime=inf;
    int sum;
    int f[MAXTIME][n+1]={0};

    Read(&a,&b);//从input.txt读取a,b, 复杂度为O(n)

    sum = min(sum(a), sum(b));//获取时间最高上界

    for(int k=1;k<=n;k++){  //代表n个任务, 复杂度为O(n*sum)
        for(int i=0;i<=sum;i++){
            f[k][i]=f[k-1][i]+b[k];
            if(i>=a[k]){              //当前时间是否满足其在机器A上运行
                f[k][i]=f[k][i]<f[k-1][i-a[k]]?f[k][i]:f[k-1][i-a[k]];
            }
        }
    }

    for(int i=0;i<=sum;i++){          //遍历f数组, 取最大最小值, 复杂度为O(sum)
        int t=(f[n][i]>i?f[n][i]:i);
        if(minTime>t)
            minTime=t;
    }

    Write(minTime)//输出数据到output.txt, 复杂度为O(1)
    return minTime;
```

```
}
```

3-8 乘法表问题

用动态规划思想考虑此问题，其动态性来源是第一次分割的位置，在此任务中可令$a=0,b=1,c=2$，令$num[i,j,k]$表示$arr[i:j]$得到k值的个数，i,j为其动态性来源。由于$c*a=a,a*c=a,b*c=a$，即对于最终结果为a的递归式由所有分割，每次分割由三种可能值为a的累加组成:

$$num[i,j,0]=\sum_{k=i}^{j}(num[i,k,2]*num[k,j,0]+num[i,k,0]*num[k,j,2]+num[i,k,1]*num[k,j,2])$$

同理可得:

$$num[i,j,1]=\sum_{k=i}^{j}(num[i,k,0]*num[k,j,0]+num[i,k,0]*num[k,j,1]+num[i,k,1]*num[k,j,1])$$
$$num[i,j,2]=\sum_{k=i}^{j}(num[i,k,1]*num[k,j,0]+num[i,k,2]*num[k,j,1]+num[i,k,2]*num[k,j,2])$$

最终对于序列arr，其值为a的个数为

$$num[1,n,0]$$

对于边界条件，即序列只有一个元素：

$$\begin{cases} num[i,i,k]=1, arr[k]=k \\ num[i,i,k]=0, arr[k]\neq k \end{cases}$$

复杂度：

由于i需要遍历n次，j需要遍历$n-i$次，k为常数不考虑，即总时间为$n*\sum_{i=1}^{n}(n-i)$，即时间复杂度为$O(n^2)$.

伪代码：

```
//伪代码
int dynamic_product(){
    int a, b, c, n;
    a = 0;
    b = 1;
    c = 2;
    char *array;

    n = Read(array);//读取input.txt数据，复杂度为O(n)
    for (int r = 2; r <= n; r++)
    {//接着从长度为 2 的子字符串计算，直至计算好整串 array
        for (int i = 1; i <= n; i++)
        {
            int j = i + r - 1;//计算array[i:j]
            for (int k = i; k <= j; k++)
            {//根据题目中的表，计算加括号法
                result[i][j][a] += result[i][k][a] * result[k + 1][j][c] + result[i][k][b] * result[k + 1][j][c] + result[i][k][c] * result[k + 1][j][a];
                result[i][j][b] += result[i][k][a] * result[k + 1][j][a] + result[i][k][a] * result[k + 1][j][b] + result[i][k][b] * result[k + 1][j][b];
```

```
                    result[i][j][c] += result[i][k][b] * result[k + 1][j][a] + result[i][k][c] * result[k + 1][j][b] + result[i][k][c] * result[k + 1][j][c];
                }
            }
        }
        write(n);//输出到output.txt，复杂度为O(n)
        return result[1][n][0];
}
```

编程实现题

3-2 编辑距离问题

对于两个字符串A, B，长度分别为m, n，令其最小编辑距离为$min_dis(m, n)$，对于A_m，B_n的比对，将A_m删除等价于将B_n后面插入一个A_m，同理将B_n删除等价于将A_m后面插入一个B_n，即插入操作与删除操作等价。因此只考虑删除与替换操作。

当$A[m] == B[n]$时，此时不需要编辑，当$A[m] != B[n]$时，此时有三种操作，删除$A[m]$，删除$B[n]$与替换，取两者最小值，此为动态性来源，满足最优子结构，即最后递推式为:

$$min\_dis(m, n) = min\_dis(m-1, n-1), if\ A[m] = B[n]$$
$$min\_dis(m, n) = \min\{min\_dis(m-1, n), min\_dis(m, n-1), min\_dis(m-1.n-1)\}, if\ A[m] \neq B[n]$$

边界条件:

$$mis\_dis(m, 0) = m$$
$$mis\_dis(0, n) = n$$

由于$min_dis(m-1, n)$包含$min_dis(m-1, n-1)$，即上述算法存在大量重叠子问题，可通过由底向上向上的方法优化，即先从边界开始算出各个子问题，然后根据子问题，计算出原问题。对于长度为

复杂度分析:

对于长度为m, n的两个序列，其共有$m * n$个子问题，由于每个子问题只计算一遍，即时间复杂度为$O(m * n)$，维持这个子问题空间复杂度为$O(m * n)$。

具体代码见附件。

程序直接运行edit_dis.exe。程序以当前文件夹的input.txt作为输入，output.txt作为输出。