摘要

本次作业主要设计了如何寻找一个最长的严格递增的子列。

1 设计思路（动态规划）

1. 首先将数列存到一个数组中 `double array[n]={`a_1，a_2，a_3，...，a_n`}` 用 $d[i]$ 表示以数组中第 i 个元素结尾的最长严格递增子列的长度，那么 $d[i] = max(d[i], d[j] + 1)$, 如果 $1 \leq j \leq i$, $a[j] < a[i]$（说明 `array[i]` 能接在 `array[j]` 后面）。之后初始化一个元素全为 1 的数组 `d`，这里不妨设下标都是从 1 开始（更符合生活中的习惯）。

2. 伪代码如下：

```
            double array[n]={a1, a2,a3,...an}
            int d[n];
            std::fill(d,d+n,1);
            for(int i=2; i<=n; ++i){
                for(int j=1; j<=i; ++j){
                    if(array[i]>array[j]){
                        d[i]=max(array[i],array[j]+1);
                    }
                }
            }
            int length=max(d);
```

Listing 1: 找到最长递增子列长度

这样要求一条最长递增子列的长度，只需要返回数组 `d` 中的最大值即可。

3. 如果要求出某一个最长递增子列，只要找到 `d[i]` 中对应 $length, length - 1, ..., 1$ 的下标 $i_{length}, ...i_1$，然后确保是递增子列即可，具体的伪代码如下：

```
            int i=length;
            double x=0.0;
            std::queue<double> q;
            for(j=n;j>0; --j){
                if(i==length && d[j]==i){
                    q.push(array[j]);
                    --i;
                    x=array[j]
                }
                else if(i<length && d[j]==i && array[j]<x){
                    q.push(array[j]);
                    --i;
                    x=array[j];
                }
            }
```

最后让队列的元素出队就得到一个严格最长递增子列。

2 算法的复杂度分析

显然这个算法的空间复杂度是 $O(n)$，由于要找到最长子序列相当于要遍历一遍二维矩阵，因此时间复杂度也是 $O(n^2)$

3 一个简单的例子

假设现在有一个数组，相应的 `array` 和 `d` 分别如下，可以看出最长的递增子列长度是 6，能找出的一个递增子列是 `{1,3,4,8,9,10}`

i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
array[i]	10	1	5	3	6	4	8	9	7	10
d[i]	1	1	2	2	3	3	4	5	4	6