专题训练

完善程序

（坐标统计）输入n个整点在平面上的坐标。对于每个点，可以控制所有位于它左下方的点（即x、y坐标都比它小），它可以控制的点的数目称为“战斗力”。依次输出每个点的战斗力，最后输出战斗力最高的点的编号（如果若干个点的战斗力并列最高，输出其中最大的编号）

手算样例

输入样例：

```
3
1 1
2 2
3 4
```

输出多少？

输出样例：

```
0
1
2
3
```

手算样例

输入样例：

```
6
1 -1
-5 0
-5 -1
2 2
-6 -3
2 5
```

输出多少？

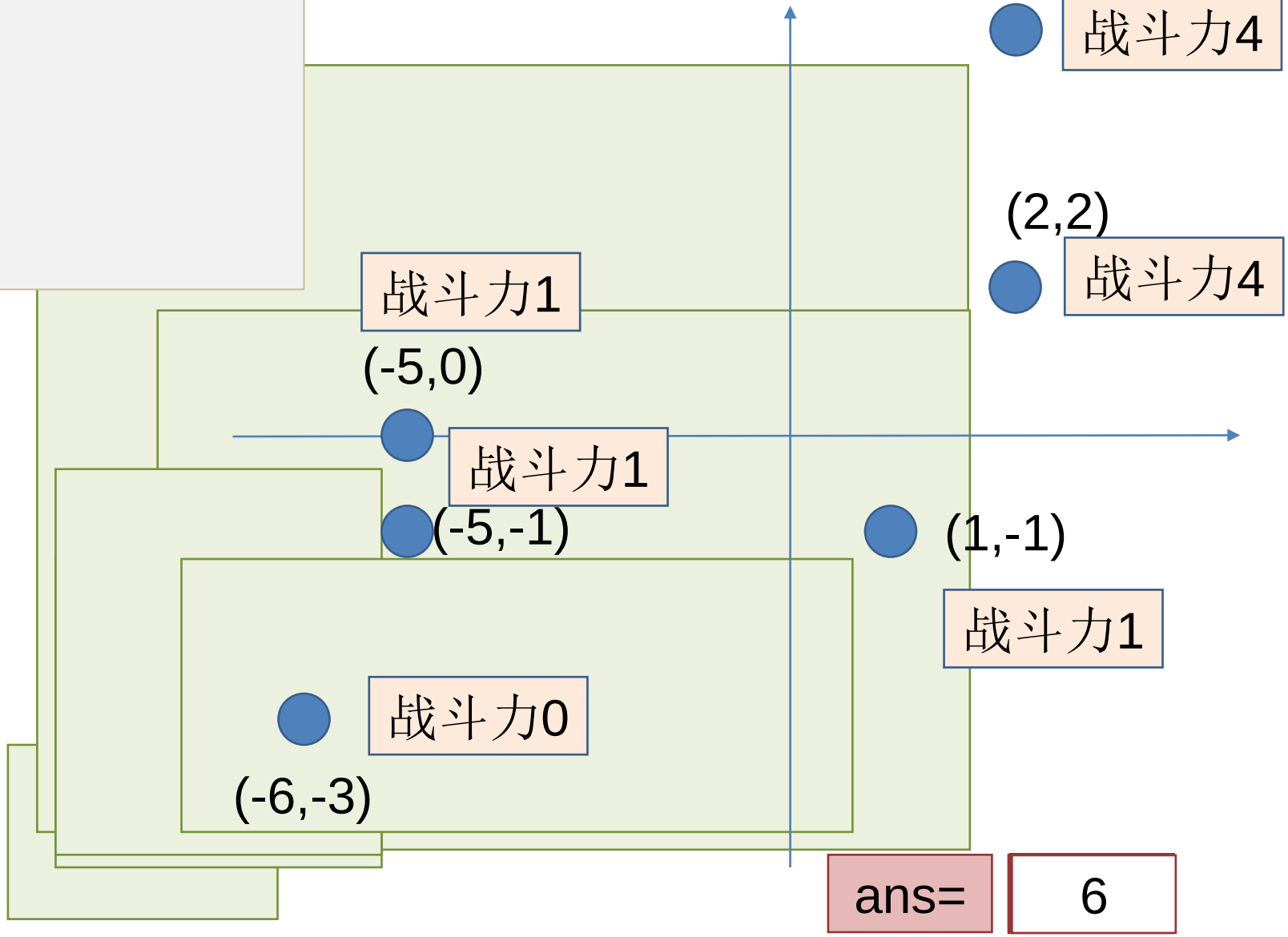
输出样例：

```
1
1
1
4
0
4
6
```

手算样例

输入样例：

```
6
1 -1
-5 0
-5 -1
2 2
-6 -3
2 5
```



5分钟

完善程序

```
1  #include <iostream>
2  using namespace std;
3  const int SIZE=100;
4  int x[SIZE],y[SIZE],f[SIZE];
5  int n,i,j,max_f,ans;
6  int main(){
7      cin>>n;
8      for(i=1;i<=n;i++) cin>>x[i]>>y[i];
9      max_f=0;
10     for(i=1;i<=n;i++){
11         f[i]=_____(1)_____;
12         for(j=1;j<=n;j++){
13             if(x[j]<x[i]&&_____(2)_____)
14                 _____(3)_____;
15         }
16         if(_____(4)_____){
17             max_f=f[i];
18             _____(5)_____;
19         }
20     }
21     for(i=1;i<=n;i++) cout<<f[i]<<endl;
22     cout<<ans<<endl;
23     return 0;
24 }
```

1	识别变量
常见变量名 翻译循环变量 根据变量名的英文推断	

2	找出关键语句
控制结构(for, if) 常见算法的基本操作 函数参数、返回值	

3	理解代码段作用
翻译解释代码段	

解释变量作用

完善程序

```
#include <iostream>
using namespace std;
const int SIZE=100;
int x[SIZE],y[SIZE],f[SIZE];
int n,i,j,max_f,ans;
int main(){
    cin>>n;
    for(i=1;i<=n;i++) cin>>x[i]>>y[i];
    max_f=0;
    for(i=1;i<=n;i++){
        f[i]=_____(1)_____;
        for(j=1;j<=n;j++){
            if(x[j]<x[i]&&_____(2)_____)
                _____(3)_____;
        }
        if(_____(4)_____){
            max_f=f[i];
            _____(5)_____;
        }
    }
    for(i=1;i<=n;i++) cout<<f[i]<<endl;
    cout<<ans<<endl;
    return 0;
}
```

n表示点的数量

x[i]表示第i个点的横坐标

y[i]表示第i个点的纵坐标

f[i]表示i号点的战斗力值

max_f表示最大战斗力值

ans表示目前最大战斗力的点的编号

关键语句

完善程序

```
#include <iostream>
using namespace std;
const int SIZE=100;
int x[SIZE],y[SIZE],f[SIZE];
int n,i,j,max_f,ans;
int main(){
    cin>>n;
    for(i=1;i<=n;i++) cin>>x[i]>>y[i];
    max_f=0;
    for(i=1;i<=n;i++){
        f[i]=_____(1)_____;
        for(j=1;j<=n;j++){
            if(x[j]<x[i]&&_____(2)_____)
                _____(3)_____;
        }
        if(_____(4)_____){
            max_f=f[i];
            _____(5)_____;
        }
    }
    for(i=1;i<=n;i++) cout<<f[i]<<endl;
    cout<<ans<<endl;
    return 0;
}
```

将每个点i的横纵坐标分别读取数组x[i]和y[i]

关键语句

完善程序

```
#include <iostream>
using namespace std;
const int SIZE=100;
int x[SIZE],y[SIZE],f[SIZE];
int n,i,j,max_f,ans;
int main(){
    cin>>n;
    for(i=1;i<=n;i++) cin>>x[i]>>y[i];
    max_f=0;
    for(i=1;i<=n;i++){
        f[i]=_____(1)_____;
        for(j=1;j<=n;j++){
            if(x[j]<x[i]&&_____(2)_____)
                _____(3)_____;
        }
        if(_____(4)_____){
            max_f=f[i];
            _____(5)_____;
        }
    }
    for(i=1;i<=n;i++) cout<<f[i]<<endl;
    cout<<ans<<endl;
    return 0;
}
```

对每个点i
都将位于它**左下方的点**
的个数统计到f[i]中
表示“战斗力”

关键语句

完善程序

```
#include <iostream>
using namespace std;
const int SIZE=100;
int x[SIZE],y[SIZE],f[SIZE];
int n,i,j,max_f,ans;
int main(){
    cin>>n;
    for(i=1;i<=n;i++) cin>>x[i]>>y[i];
    max_f=0;
    for(i=1;i<=n;i++){
        f[i]=_____(1)_____;
        for(j=1;j<=n;j++){
            if(x[j]<x[i]&&_____(2)_____)
                _____(3)_____;
        }
        if(_____(4)_____){
            max_f=f[i];
            _____(5)_____;
        }
    }
    for(i=1;i<=n;i++) cout<<f[i]<<endl;
    cout<<ans<<endl;
    return 0;
}
```

打擂台法
如果当前点“战斗力”
超过擂主值max_f

更新擂主值max_f
并更新擂主编号

代码段作用

完善程序

```
1  #include <iostream>
2  using namespace std;
3  const int SIZE=100;
4  int x[SIZE],y[SIZE],f[SIZE];
5  int n,i,j,max_f,ans;
6  int main(){
7      cin>>n;
8      for(i=1;i<=n;i++) cin>>x[i]>>y[i];
9      max_f=0;
10     for(i=1;i<=n;i++){
11         f[i]=_____(1)_____;
12         for(j=1;j<=n;j++){
13             if(x[j]<x[i]&&_____(2)_____)
14                 _____(3)_____;
15         }
16         if(_____(4)_____){
17             max_f=f[i];
18             _____(5)_____;
19         }
20     }
21     for(i=1;i<=n;i++) cout<<f[i]<<endl;
22     cout<<ans<<endl;
23     return 0;
24 }
```

将每个点i的横纵坐标输入到x[i]和y[i]

双重循环
统计每个点i的
左下方点的个数f[i]

打擂台法
维护擂主值和编号

输出每个点f[i]
并输出最大值编号ans

完善程序

```
#include <iostream>
using namespace std;
const int SIZE=100;
int x[SIZE],y[SIZE],f[SIZE];
int n,i,j,max_f,ans;
int main(){
    cin>>n;
    for(i=1;i<=n;i++) cin>>x[i]>>y[i];
    max_f=0;
    for(i=1;i<=n;i++){
        f[i]=_____(1)_____;
        for(j=1;j<=n;j++){
            if(x[j]<x[i]&&_____(2)_____)
                _____(3)_____;
        }
        if(_____(4)_____){
            max_f=f[i];
            _____(5)_____;
        }
    }
    for(i=1;i<=n;i++) cout<<f[i]<<endl;
    cout<<ans<<endl;
    return 0;
}
```

战斗力值初始化为0

完善程序

```
#include <iostream>
using namespace std;
const int SIZE=100;
int x[SIZE],y[SIZE],f[SIZE];
int n,i,j,max_f,ans;
int main(){
    cin>>n;
    for(i=1;i<=n;i++) cin>>x[i]>>y[i];
    max_f=0;
    for(i=1;i<=n;i++){
        f[i]=_____(1)_____;
        for(j=1;j<=n;j++){
            if(x[j]<x[i]&&_____(2)_____)
                _____(3)_____;
        }
        if(_____(4)_____){
            max_f=f[i];
            _____(5)_____;
        }
    }
    for(i=1;i<=n;i++) cout<<f[i]<<endl;
    cout<<ans<<endl;
    return 0;
}
```

点j需要同时满足
横纵坐标值都比点i小

完善程序

```
#include <iostream>
using namespace std;
const int SIZE=100;
int x[SIZE],y[SIZE],f[SIZE];
int n,i,j,max_f,ans;
int main(){
    cin>>n;
    for(i=1;i<=n;i++) cin>>x[i]>>y[i];
    max_f=0;
    for(i=1;i<=n;i++){
        f[i]=_____(1)_____;
        for(j=1;j<=n;j++){
            if(x[j]<x[i]&&_____(2)_____)
                _____(3)_____;
        }
        if(_____(4)_____){
            max_f=f[i];
            _____(5)_____;
        }
    }
    for(i=1;i<=n;i++) cout<<f[i]<<endl;
    cout<<ans<<endl;
    return 0;
}
```

如果符合条件
在左下方
那么f[i]就自增1

完善程序

```
#include <iostream>
using namespace std;
const int SIZE=100;
int x[SIZE],y[SIZE],f[SIZE];
int n,i,j,max_f,ans;
int main(){
    cin>>n;
    for(i=1;i<=n;i++) cin>>x[i]>>y[i];
    max_f=0;
    for(i=1;i<=n;i++){
        f[i]=_____(1)_____;
        for(j=1;j<=n;j++){
            if(x[j]<x[i]&&_____(2)_____)
                _____(3)_____;
        }
        if(_____(4)_____){
            max_f=f[i];
            _____(5)_____;
        }
    }
    for(i=1;i<=n;i++) cout<<f[i]<<endl;
    cout<<ans<<endl;
    return 0;
}
```

打擂台
条件是目前点的武力值f[i]
大于等于擂主值max_f

思考：等号可不可省略

如果若干个点的战斗力并列最高，输出其中最大的编号

完善程序

```
1  #include <iostream>
2  using namespace std;
3  const int SIZE=100;
4  int x[SIZE],y[SIZE],f[SIZE];
5  int n,i,j,max_f,ans;
6  int main(){
7      cin>>n;
8      for(i=1;i<=n;i++) cin>>x[i]>>y[i];
9      max_f=0;
10     for(i=1;i<=n;i++){
11         f[i]=_____(1)_____;
12         for(j=1;j<=n;j++){
13             if(x[j]<x[i]&&_____(2)_____)
14                 _____(3)_____;
15         }
16         if(_____(4)_____){
17             max_f=f[i];
18             _____(5)_____;
19         }
20     }
21     for(i=1;i<=n;i++) cout<<f[i]<<endl;
22     cout<<ans<<endl;
23     return 0;
24 }
```

观察22行输出了ans
结合题意 要求输出
“战斗力最高点的编号”
可知ans需要维护最大值
编号

完善程序

（子矩阵）输入一个n1*m1的矩阵a，和n2*m2的矩阵b，问a中是否存在子矩阵和b相等。若存在，输出所有子矩阵左上角的坐标；若不存在输出“There is no answer”。

手算样例

输入样例：

```
4 4
1 2 3 4
3 4 1 4
2 2 3 4
1 4 1 4
2 3
2 3 4
4 1 4
```

手算样例

1,2

1	2	3	4
3	4	1	4
2	2	3	4
1	4	1	4

3,2

2	3	4
4	1	4

输出多少？

输出样例：

1 2

3 2

手算样例

输入样例：

```
3 3
1 2 3
3 4 1
2 2 3
2 2
1 3
2 4
```

输出多少？

输出样例：

```
There is no answer
```

5分钟

完善程序

```
#include <iostream>
using namespace std;
const int SIZE = 50;
int n1, m1, n2, m2, a[SIZE][SIZE], b[SIZE][SIZE];
int main(){
    int i, j, k1, k2;
    bool good, haveAns;
    cin >> n1 >> m1;
    for(i = 1; i <= n1; i++)
        for(j = 1; j <= m1; j++)
            cin >> a[i][j];
    cin >> n2 >> m2;
    for(i = 1; i <= n2; i++)
        for(j = 1; j <= m2; j++)
            ____(1)____;
    haveAns = false;
    for(i = 1; i <= n1 - n2 + 1; i++)
        for(j = 1; j <= ____(2)____; j++){
            ____(3)____;
            for(k1 = 1; k1 <= n2; k1++)
                for(k2 = 1; k2 <= ____(4)____; k2++){
                    if(a[i + k1 - 1][j + k2 - 1] != b[k1][k2])
                        good = false;
                }
            if(good){
                cout << i << ' ' << j << endl;
                ____(5)____;
            }
        }
    if(!haveAns)
        cout << "There is no answer" << endl;
    return 0;
}
```

1	识别变量
常见变量名 翻译循环变量 根据变量名的英文推断	

2	找出关键语句
控制结构(for, if) 常见算法的基本操作 函数参数、返回值	

3	理解代码段作用
翻译解释代码段	

完善程序

```
#include <iostream>
using namespace std;
const int SIZE = 50;
int n1, m1, n2, m2, a[SIZE][SIZE], b[SIZE][SIZE];
int main(){
    int i, j, k1, k2;
    bool good, haveAns;
    cin >> n1 >> m1;
    for(i = 1; i <= n1; i++)
        for(j = 1; j <= m1; j++)
            cin >> a[i][j];
    cin >> n2 >> m2;
    for(i = 1; i <= n2; i++)
        for(j = 1; j <= m2; j++)
            ____(1)____;
    haveAns = false;
    for(i = 1; i <= n1 - n2 + 1; i++)
        for(j = 1; j <= ____(2)____; j++){
            ____(3)____;
            for(k1 = 1; k1 <= n2; k1++)
                for(k2 = 1; k2 <= ____(4)____; k2++){
                    if(a[i + k1 - 1][j + k2 - 1] != b[k1][k2])
                        good = false;
                }
            if(good){
                cout << i << ' ' << j << endl;
                ____(5)____;
            }
        }
    if(!haveAns)
        cout << "There is no answer" << endl;
    return 0;
}
```

解释变量的作用

n1,m1表示a矩阵行、列数

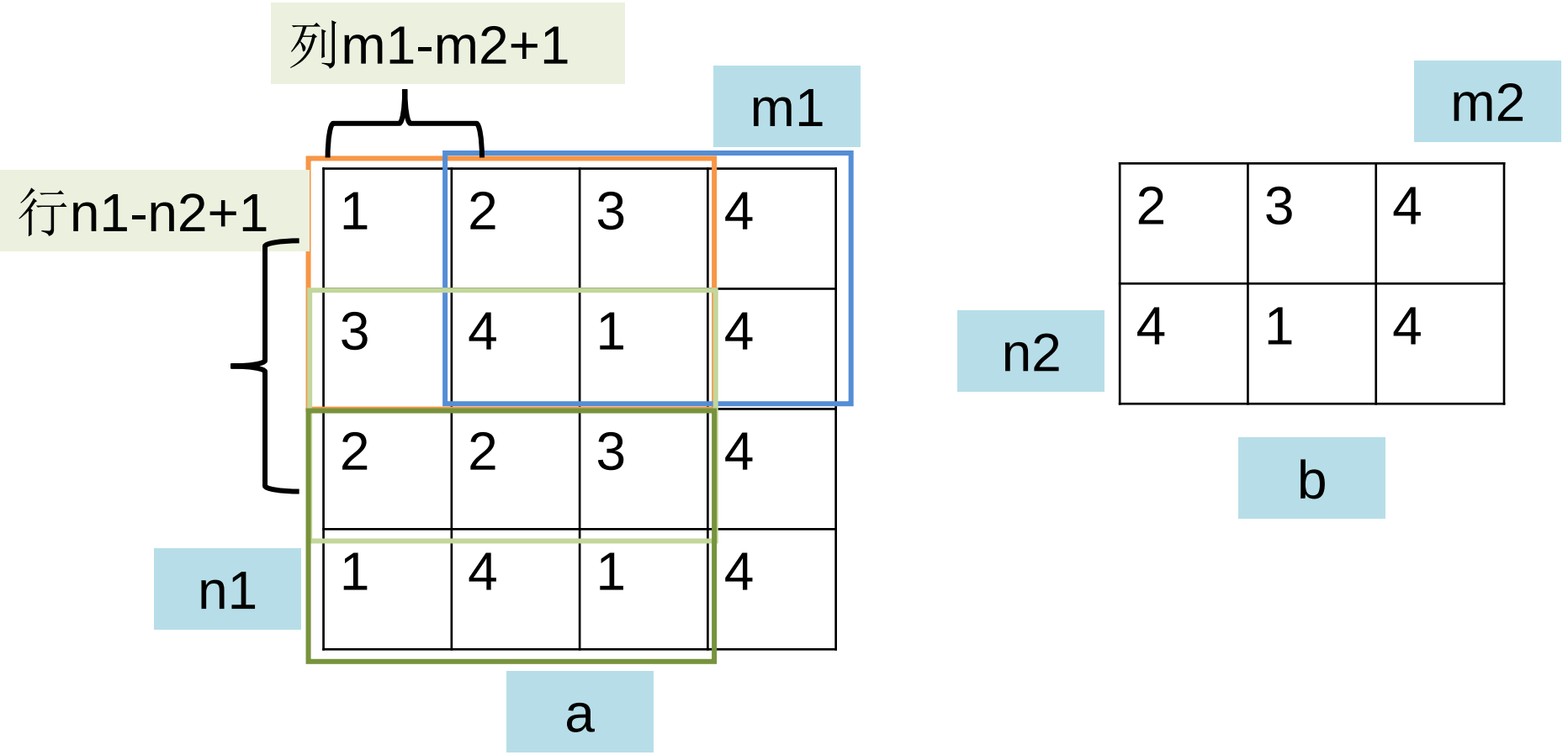
n2,m2表示b矩阵行、列数

haveANS表示是否存在b是a的子矩阵

good表示a的当前子矩阵是否和b矩阵相同

i,j,k1,k2循环变量

变量解释



完善程序

```
#include <iostream>
using namespace std;
const int SIZE = 50;
int n1, m1, n2, m2, a[SIZE][SIZE], b[SIZE][SIZE];
int main(){
    int i, j, k1, k2;
    bool good, haveAns;
    cin >> n1 >> m1;
    for(i = 1; i <= n1; i++)
        for(j = 1; j <= m1; j++)
            cin >> a[i][j];
    cin >> n2 >> m2;
    for(i = 1; i <= n2; i++)
        for(j = 1; j <= m2; j++)
            ____(1)____;
    haveAns = false;
    for(i = 1; i <= n1 - n2 + 1; i++)
        for(j = 1; j <= ____(2)____; j++){
            ____(3)____;
            for(k1 = 1; k1 <= n2; k1++)
                for(k2 = 1; k2 <= ____(4)____; k2++){
                    if(a[i + k1 - 1][j + k2 - 1] != b[k1][k2])
                        good = false;
                }
            if(good){
                cout << i << ' ' << j << endl;
                ____(5)____;
            }
        }
    if(!haveAns)
        cout << "There is no answer" << endl;
    return 0;
}
```

关键语句

循环枚举：
i和j在枚举什么？
k1和k2又在循环什么

i，j枚举子矩阵的左上角坐标

k1，k2循环查看子矩阵中各坐标

完善程序

```
#include <iostream>
using namespace std;
const int SIZE = 50;
int n1, m1, n2, m2, a[SIZE][SIZE], b[SIZE][SIZE];
int main(){
    int i, j, k1, k2;
    bool good, haveAns;
    cin >> n1 >> m1;
    for(i = 1; i <= n1; i++)
        for(j = 1; j <= m1; j++)
            cin >> a[i][j];
    cin >> n2 >> m2;
    for(i = 1; i <= n2; i++)
        for(j = 1; j <= m2; j++)
            ____(1)____;
    haveAns = false;
    for(i = 1; i <= n1 - n2 + 1; i++)
        for(j = 1; j <= ____(2)____; j++){
            ____(3)____;
            for(k1 = 1; k1 <= n2; k1++)
                for(k2 = 1; k2 <= ____(4)____; k2++){
                    if(a[i + k1 - 1][j + k2 - 1] != b[k1][k2])
                        good = false;
                }
            if(good){
                cout << i << ' ' << j << endl;
                ____(5)____;
            }
        }
    if(!haveAns)
        cout << "There is no answer" << endl;
    return 0;
}
```

关键语句

good=true说明什么？

说明找到a矩阵中和b矩阵匹配情况。

题目中有“若不存在则输出.....”

因此需要记录存在匹配情况以便后续分类输出。

完善程序

```
#include <iostream>
using namespace std;
const int SIZE = 50;
int n1, m1, n2, m2, a[SIZE][SIZE], b[SIZE][SIZE];
int main(){
    int i, j, k1, k2;
    bool good, haveAns;
    cin >> n1 >> m1;
    for(i = 1; i <= n1; i++)
        for(j = 1; j <= m1; j++)
            cin >> a[i][j];
    cin >> n2 >> m2;
    for(i = 1; i <= n2; i++)
        for(j = 1; j <= m2; j++)
            ____(1)____;
    haveAns = false;
    for(i = 1; i <= n1 - n2 + 1; i++)
        for(j = 1; j <= ____(2)____; j++){
            ____(3)____;
            for(k1 = 1; k1 <= n2; k1++)
                for(k2 = 1; k2 <= ____(4)____; k2++){
                    if(a[i + k1 - 1][j + k2 - 1] != b[k1][k2])
                        good = false;
                }
            if(good){
                cout << i << ' ' << j << endl;
                ____(5)____;
            }
        }
    if(!haveAns)
        cout << "There is no answer" << endl;
    return 0;
}
```

代码段作用

输入矩阵a、矩阵b

枚举每个子矩阵进行匹配

输出子矩阵左上角坐标

没有匹配的情况单独考虑

完善程序

（**二叉查找树**）二叉查找树具有如下性质：每个节点的值都大于其左子树上所有节点的值、小于其右子树上所有节点的值。试判断一棵树是否为二叉查找树。
输入的第一行包含一个整数 n，表示这棵树有 n 个顶点，编号分别为 1,2,…,n其中编号为 1 的为根结点。之后的第 i 行有三个数 value，left_child，right_child，分别表示该节点关键字的值、左子节点的编号、右子节点的编号；如果不存在左子节点或右子节点，则用 0 代替。输出 1 表示这棵树是二叉查找树，输出 0 则表示不是。

手算样例

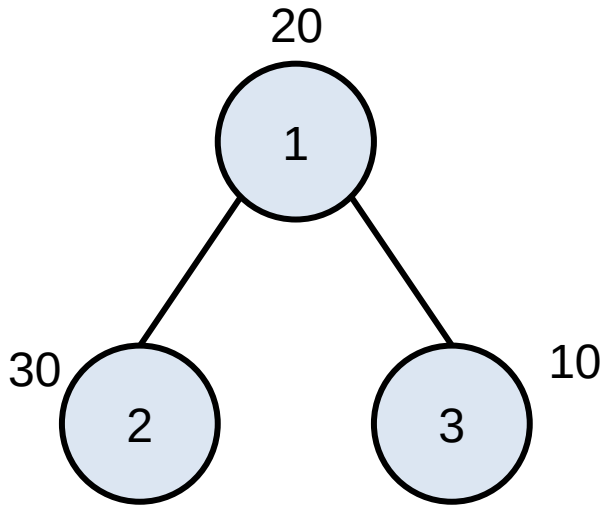
输入样例：

```
3
20 2 3
30 0 0
10 0 0
```

输出多少？

输出样例：

```
0
```



手算样例

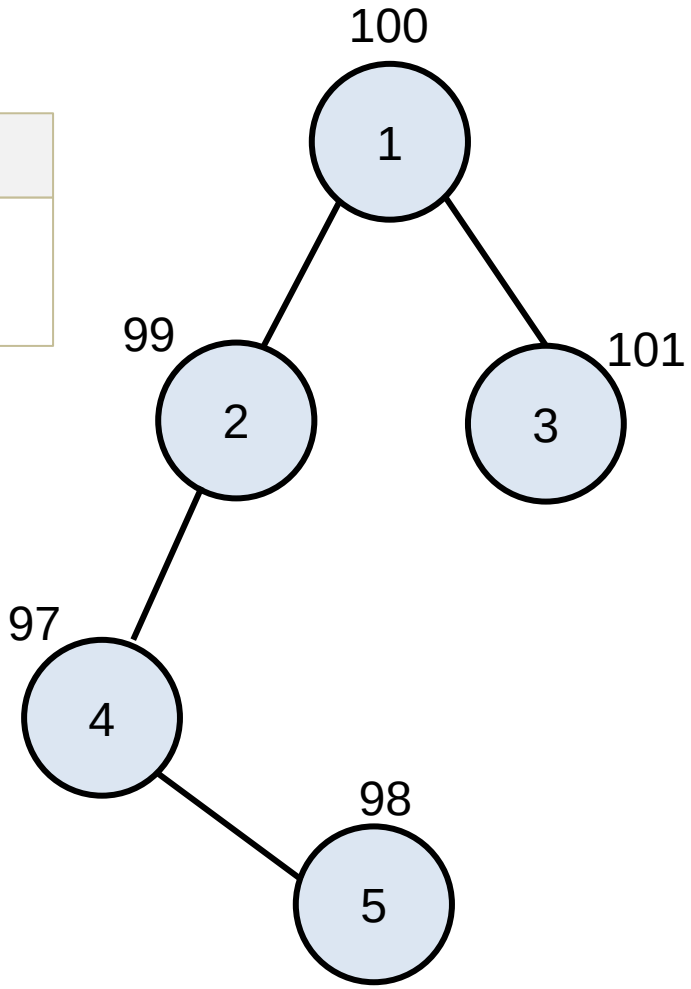
输入样例：

```
5
100 2 3
99 4 0
101 0 0
97 0 5
98 0 0
```

输出多少？

输出样例：

```
1
```



5分钟

完善程序

```
#include <iostream>
using namespace std;
const int SIZE=100;
const int INFINITE=1000000;
struct node {int left_child, right_child, value;};
node a[SIZE];
int is_bst(int root, int lower_bound, int upper_bound) {
    if(root==0) return 1;
    int cur=a[root].value;
    if( (cur>lower_bound) && (____(1)____) &&
    (is_bst(a[root].left_child, lower_bound, cur)==1) &&
    (is_bst(____(2)____, ____(3)____, ____(4)____)==1) )
        return 1;
    return 0;
}
int main(){
    int i,n;
    cin>>n;
    for(i=1;i<=n;i++)
        cin>>a[i].value>>a[i].left_child>>a[i].right_child;
    cout<<is_bst(____(5)____, -INFINITE, INFINITE);
    return 0;
}
```

1	识别变量
常见变量名 翻译循环变量 根据变量名的英文推断	

2	找出关键语句
控制结构(for, if) 常见算法的基本操作 函数参数、返回值	

3	理解代码段作用
翻译解释代码段	

二叉查找树BST

解释变量作用

完善程序

```
1  #include <iostream>
2  using namespace std;
3  const int SIZE=100;
4  const int INFINITE=1000000;
5  struct node {int left_child, right_child, value;};
6  node a[SIZE];
7  int is_bst(int root, int lower_bound, int upper_bound) {
8      if(root==0) return 1;
9      int cur=a[root].value;
10     if( (cur>lower_bound) && (____(1)____) &&
11     (is_bst(a[root].left_child, lower_bound, cur)==1) &&
12     (is_bst(____(2)____, ____(3)____, ____(4)____)==1) )
13         return 1;
14     return 0;
15 }
16 int main(){
17     int i,n;
18     cin>>n;
19     for(i=1;i<=n;i++)
20         cin>>a[i].value>>a[i].left_child>>a[i].right_child;
21     cout<<is_bst(____(5)____, -INFINITE, INFINITE);
22     return 0;
23 }
```

a[i]表示第i号结点
每个结点有左儿子编号、右儿子编号、值三个属性

is_bst判断以root为根的子树是否是二叉查找树

root指根结点编号

lower_bound表示结点值应满足的下限

upper_bound表示结点值应满足的上限

cur是本层结点root的值

二叉查找树BST

关键语句

完善程序

```
1  #include <iostream>
2  using namespace std;
3  const int SIZE=100;
4  const int INFINITE=1000000;
5  struct node {int left_child, right_child, value;};
6  node a[SIZE];
7  int is_bst(int root, int lower_bound, int upper_bound) {
8      if(root==0) return 1;
9      int cur=a[root].value;
10     if( (cur>lower_bound) && (____(1)____) &&
11     (is_bst(a[root].left_child, lower_bound, cur)==1) &&
12     (is_bst(____(2)____, ____(3)____, ____(4)____)==1) )
13         return 1;
14     return 0;
15 }
16 int main(){
17     int i,n;
18     cin>>n;
19     for(i=1;i<=n;i++)
20         cin>>a[i].value>>a[i].left_child>>a[i].right_child;
21     cout<<is_bst(____(5)____, -INFINITE, INFINITE);
22     return 0;
23 }
```

定义结构体node
包含其左儿子编号、右儿子编号、值三个属性
定义结构体数组a并输入

二叉查找树BST

关键语句

完善程序

```
#include <iostream>
using namespace std;
const int SIZE=100;
const int INFINITE=1000000;
struct node {int left_child, right_child, value;};
node a[SIZE];
int is_bst(int root, int lower_bound, int upper_bound) {
    if(root==0) return 1;
    int cur=a[root].value;
    if( (cur>lower_bound) && (____(1)____) &&
    (is_bst(a[root].left_child, lower_bound, cur)==1) &&
    (is_bst(____(2)____, ____(3)____, ____(4)____)==1) )
        return 1;
    return 0;
}
int main(){
    int i,n;
    cin>>n;
    for(i=1;i<=n;i++)
        cin>>a[i].value>>a[i].left_child>>a[i].right_child;
    cout<<is_bst(____(5)____, -INFINITE, INFINITE);
    return 0;
}
```

is_bst函数是递归调用
三个参数分别表示本层结点编号、上限值、下限值

二叉查找树BST

关键语句

完善程序

```
1  #include <iostream>
2  using namespace std;
3  const int SIZE=100;
4  const int INFINITE=1000000;
5  struct node {int left_child, right_child, value;};
6  node a[SIZE];
7  int is_bst(int root, int lower_bound, int upper_bound) {
8      if(root==0) return 1;
9      int cur=a[root].value;
10     if( (cur>lower_bound) && (____(1)____) &&
11     (is_bst(a[root].left_child, lower_bound, cur)==1) &&
12     (is_bst(____(2)____, ____(3)____, ____(4)____)==1) )
13         return 1;
14     return 0;
15 }
16 int main(){
17     int i,n;
18     cin>>n;
19     for(i=1;i<=n;i++)
20         cin>>a[i].value>>a[i].left_child>>a[i].right_child;
21     cout<<is_bst(____(5)____, -INFINITE, INFINITE);
22     return 0;
23 }
```

is_bst函数有三个返回语句

首先当root==0
叶子结点上一层没有左孩子
或右孩子结点
结束递归直接返回给上一层1

其次当满足if中条件时
返回1 否则返回0

二叉查找树BST

关键语句

完善程序

```
1  #include <iostream>
2  using namespace std;
3  const int SIZE=100;
4  const int INFINITE=1000000;
5  struct node {int left_child, right_child, value;};
6  node a[SIZE];
7  int is_bst(int root, int lower_bound, int upper_bound) {
8      if(root==0) return 1;
9      int cur=a[root].value;
10     if( (cur>lower_bound) && (____(1)____) &&
11     (is_bst(a[root].left_child, lower_bound, cur)==1) &&
12     (is_bst(____(2)____, ____(3)____, ____(4)____)==1) )
13         return 1;
14     return 0;
15 }
16 int main(){
17     int i,n;
18     cin>>n;
19     for(i=1;i<=n;i++)
20         cin>>a[i].value>>a[i].left_child>>a[i].right_child;
21     cout<<is_bst(____(5)____, -INFINITE, INFINITE);
22     return 0;
23 }
```

观察if成立的条件

① 需要结点值大于下限

② 需要结点值小于上限

③ 调用is_bst函数，将左子结点编号传入，需要返回1

④调用is_bst函数，将右子结点编号传入，需要返回1

哪种二叉树遍历顺序

前序	根左右
中序	左根右
后序	左右根

二叉查找树BST

关键语句

完善程序

```
1  #include <iostream>
2  using namespace std;
3  const int SIZE=100;
4  const int INFINITE=1000000;
5  struct node {int left_child, right_child, value;};
6  node a[SIZE];
7  int is_bst(int root, int lower_bound, int upper_bound) {
8      if(root==0) return 1;
9      int cur=a[root].value;
10     if( (cur>lower_bound) && (____(1)____) &&
11     (is_bst(a[root].left_child, lower_bound, cur)==1) &&
12     (is_bst(____(2)____, ____(3)____, ____(4)____)==1) )
13         return 1;
14     return 0;
15 }
16 int main(){
17     int i,n;
18     cin>>n;
19     for(i=1;i<=n;i++)
20         cin>>a[i].value>>a[i].left_child>>a[i].right_chi
21     cout<<is_bst(____(5)____, -INFINITE, INFINITE);
22     return 0;
23 }
```

第一个is_bst函数调用
将本层结点左子结点编号传入
该左子结点值下限只要保持和
父节点一样就行（**思考为什么**）
上限调整为不超过本层结点
（该左子结点父亲）的值cur

第二个is_bst函数调用
将本层结点右子结点编号传入
该右子结点值上限只要保持和
父节点一样就行（**同样的原因**）
下限调整为不小于本层结点
（该左子结点父亲）的值cur

完善程序

```
1  #include <iostream>
2  using namespace std;
3  const int SIZE=100;
4  const int INFINITE=1000000;
5  struct node {int left_child, right_child, value;};
6  node a[SIZE];
7  int is_bst(int root, int lower_bound, int upper_bound) {
8      if(root==0) return 1;
9      int cur=a[root].value;
10     if( (cur>lower_bound) && (____(1)____) &&
11     (is_bst(a[root].left_child, lower_bound, cur)==1) &&
12     (is_bst(____(2)____, ____(3)____, ____(4)____)==1) )
13         return 1;
14     return 0;
15 }
16 int main(){
17     int i,n;
18     cin>>n;
19     for(i=1;i<=n;i++)
20         cin>>a[i].value>>a[i].left_child>>a[i].right_child;
21     cout<<is_bst(____(5)____, -INFINITE, INFINITE);
22     return 0;
23 }
```

定义结构体数组

递归调用is_bst函数
检查每节点值是否都大于其左子树上所有节点值、小于其右子树上所有节点值
如果是返回1，否则返回0

将每个node的值、左儿子编号和右儿子编号依次输入

主函数内递归调用is_bst函数的入口，从根结点开始

完善程序

```
1  #include <iostream>
2  using namespace std;
3  const int SIZE=100;
4  const int INFINITE=1000000;
5  struct node {int left_child, right_child, value;};
6  node a[SIZE];
7  int is_bst(int root, int lower_bound, int upper_bound) {
8      if(root==0) return 1;
9      int cur=a[root].value;
10     if( (cur>lower_bound) && (____(1)____) &&
11     (is_bst(a[root].left_child, lower_bound, cur)==1) &&
12     (is_bst(____(2)____, ____(3)____, ____(4)____)==1) )
13         return 1;
14     return 0;
15 }
16 int main(){
17     int i,n;
18     cin>>n;
19     for(i=1;i<=n;i++)
20         cin>>a[i].value>>a[i].left_child>>a[i].right_child;
21     cout<<is_bst(____(5)____, -INFINITE, INFINITE);
22     return 0;
23 }
```

该层结点的值cur需要小于参数传入的上限upper_bound

完善程序

第一个参数应传入该层结点的右子结点编号
即a[root].right_child

```
1  #include <iostream>
2  using namespace std;
3  const int SIZE=100;
4  const int INFINITE=1000000;
5  struct node {int left_child, right_child, value;};
6  node a[SIZE];
7  int is_bst(int root, int lower_bound, int upper_bound) {
8      if(root==0) return 1;
9      int cur=a[root].value;
10     if( (cur>lower_bound) && (____(1)____) &&
11     (is_bst(a[root].left_child, lower_bound, cur)==1) &&
12     (is_bst(____(2)____, ____(3)____, ____(4)____)==1) )
13         return 1;
14     return 0;
15 }
16 int main(){
17     int i,n;
18     cin>>n;
19     for(i=1;i<=n;i++)
20         cin>>a[i].value>>a[i].left_child>>a[i].right_child;
21     cout<<is_bst(____(5)____, -INFINITE, INFINITE);
22     return 0;
23 }
```

完善程序

第二个参数应传入右儿子下限
右儿子不能小于父亲值
所以下限应是cur

```
#include <iostream>
using namespace std;
const int SIZE=100;
const int INFINITE=1000000;
struct node {int left_child, right_child, value;};
node a[SIZE];
int is_bst(int root, int lower_bound, int upper_bound) {
    if(root==0) return 1;
    int cur=a[root].value;
    if( (cur>lower_bound) && (____(1)____) &&
    (is_bst(a[root].left_child, lower_bound, cur)==1) &&
    (is_bst(____(2)____, ____(3)____, ____(4)____)==1) )
        return 1;
    return 0;
}
int main(){
    int i,n;
    cin>>n;
    for(i=1;i<=n;i++)
        cin>>a[i].value>>a[i].left_child>>a[i].right_child;
    cout<<is_bst(____(5)____, -INFINITE, INFINITE);
    return 0;
}
```

完善程序

第三个参数应传入右儿子上限
右儿子只需要继承父亲上限
下层沿用本层upper_bound

```
1  #include <iostream>
2  using namespace std;
3  const int SIZE=100;
4  const int INFINITE=1000000;
5  struct node {int left_child, right_child, value;};
6  node a[SIZE];
7  int is_bst(int root, int lower_bound, int upper_bound) {
8      if(root==0) return 1;
9      int cur=a[root].value;
10     if( (cur>lower_bound) && (____(1)____) &&
11     (is_bst(a[root].left_child, lower_bound, cur)==1) &&
12     (is_bst(____(2)____, ____(3)____, ____(4)____)==1) )
13         return 1;
14     return 0;
15 }
16 int main(){
17     int i,n;
18     cin>>n;
19     for(i=1;i<=n;i++)
20         cin>>a[i].value>>a[i].left_child>>a[i].right_child;
21     cout<<is_bst(____(5)____, -INFINITE, INFINITE);
22     return 0;
23 }
```

完善程序

```
#include <iostream>
using namespace std;
const int SIZE=100;
const int INFINITE=1000000;
struct node {int left_child, right_child, value;};
node a[SIZE];
int is_bst(int root, int lower_bound, int upper_bound) {
    if(root==0) return 1;
    int cur=a[root].value;
    if( (cur>lower_bound) && (____(1)____) &&
    (is_bst(a[root].left_child, lower_bound, cur)==1) &&
    (is_bst(____(2)____, ____(3)____, ____(4)____)==1) )
        return 1;
    return 0;
}
int main(){
    int i,n;
    cin>>n;
    for(i=1;i<=n;i++)
        cin>>a[i].value>>a[i].left_child>>a[i].right_child;
    cout<<is_bst(____(5)____, -INFINITE, INFINITE);
    return 0;
}
```

从结点1（也即根结点）开始递归调用

完善程序

（双链表）对于一个1到n的排列Q（即1到n中每一个数在Q中出现了恰好一次），令q_i为
第i个位置之后第一个比Q_i值更大的位置，如果不存在这样的位置，则$q_i = n + 1$。
举例来说，如果$n = 5$且Q为1 5 4 2 3，则q为2 6 6 5 6。
下列程序读入了排列Q，使用双向链表求解了答案。试补全程序。
数据范围 $1 \le n \le 10^5$

手算样例

输入样例：

```
4
1 2 3 4
```

输出多少？

输出样例：

```
2 3 4 5
```

手算样例

输入样例：

```
8
1 3 7 4 6 2 8 5
```

输出多少？

输出样例：

```
2 3 7 5 7 7 9 9
```

5分钟

完善程序

```
1  #include<iostream>
2  using namespace std;
3  const int N=10010;
4  int n;
5  int L[N], R[N], a[N];
6  int main() {
7      cin>>n;
8      for(int i=1; i<=n; ++i) {
9          int x;
10         cin>>x;
11         _____(1) _____;
12     }
13     for(int i=1; i<=n; ++i) {
14         R[i] = _____(2) _____;
15         L[i] = i-1;
16     }
17     for(int i=1; i<=n; ++i) {
18         L[_____(3) _____] = L[a[i]];
19         R[L[a[i]]] = R[_____(4) _____];
20     }
21     for(int i=1; i<=n; ++i)
22         cout<<_____(5) _____<<" ";
23     cout<<endl;
24     return 0;
25 }
```

1	识别变量
常见变量名 翻译循环变量 根据变量名的英文推断	

2	找出关键语句
控制结构(for, if) 常见算法的基本操作 函数参数、返回值	

3	理解代码段作用
翻译解释代码段	

完善程序

```
1  #include<iostream>
2  using namespace std;
3  const int N=10010;
4  int n;
5  int L[N], R[N], a[N];
6  int main() {
7      cin>>n;
8      for(int i=1; i<=n; ++i) {
9          int x;
10         cin>>x;
11         _____(1) _____;
12     }
13     for(int i=1; i<=n; ++i) {
14         R[i] = _____(2) _____;
15         L[i] = i-1;
16     }
17     for(int i=1; i<=n; ++i) {
18         L[_____(3) _____] = L[a[i]];
19         R[L[a[i]]] = R[_____(4) _____];
20     }
21     for(int i=1; i<=n; ++i)
22         cout<<_____(5) _____<<" ";
23     cout<<endl;
24     return 0;
25 }
```

解释变量的作用

n	表示序列长度
x	表示第i个数字
a[x]	表示数字x在序列中的位置
L[i]	表示位置i的前驱
R[i]	表示位置i的后继

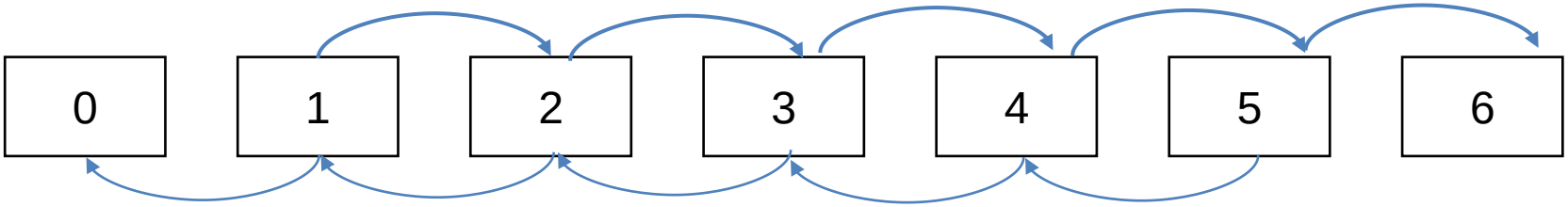
完善程序

输入：
5
1 5 4 2 3

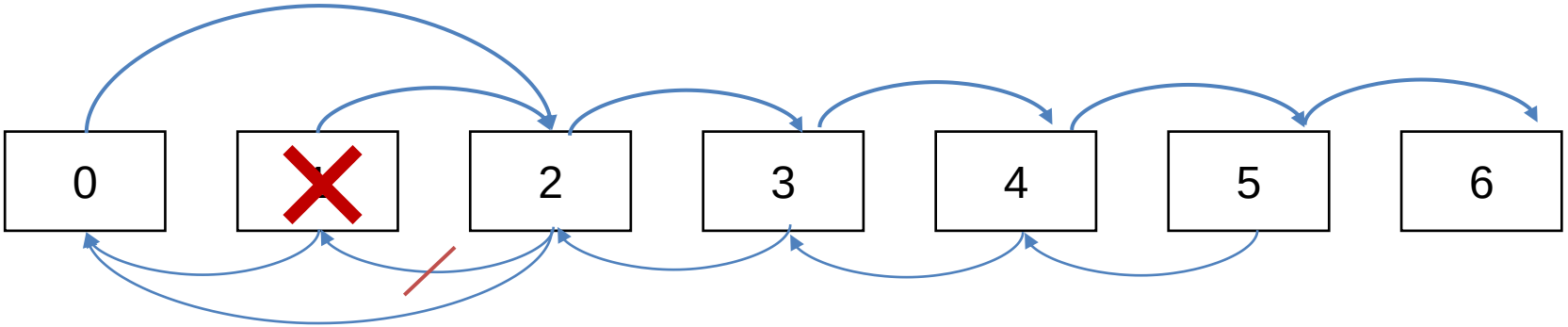
输出：
2 6 6 5 6

i	0	1	2	3	4	5	6
a		1	4	5	3	2	
L		0	1	2	3	4	
R		2	3	4	5	6	

完善程序



完善程序



删除当前最小数字1

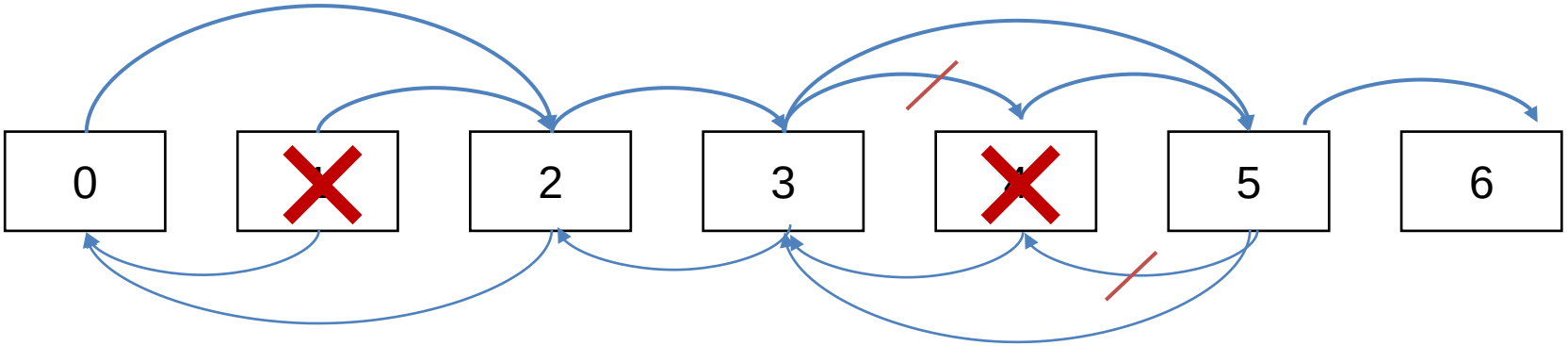
1的位置a[i]=1

从链表中删除a[i]

a[i]的后继的前驱变成a[i]的前驱

a[i]的前驱的后继变成a[i]的后继

完善程序



删除当前最小数字2

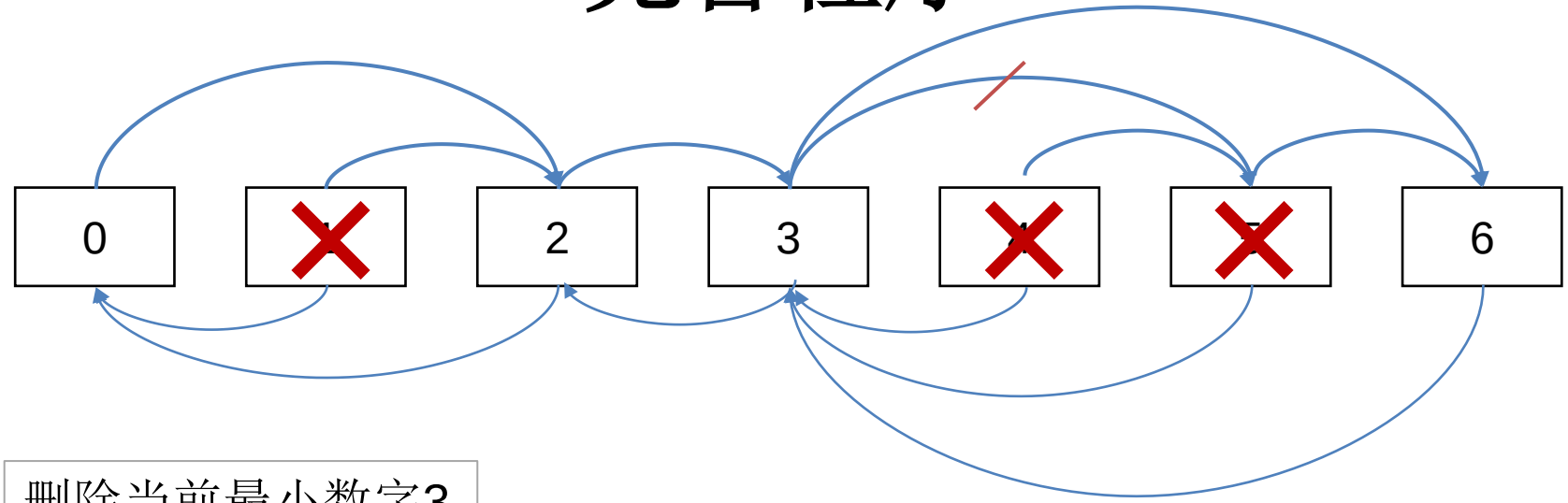
2的位置a[2]=4

从链表中删除a[i]

a[i]的后继的前驱变成a[i]的前驱

a[i]的前驱的后继变成a[i]的后继

完善程序



删除当前最小数字3

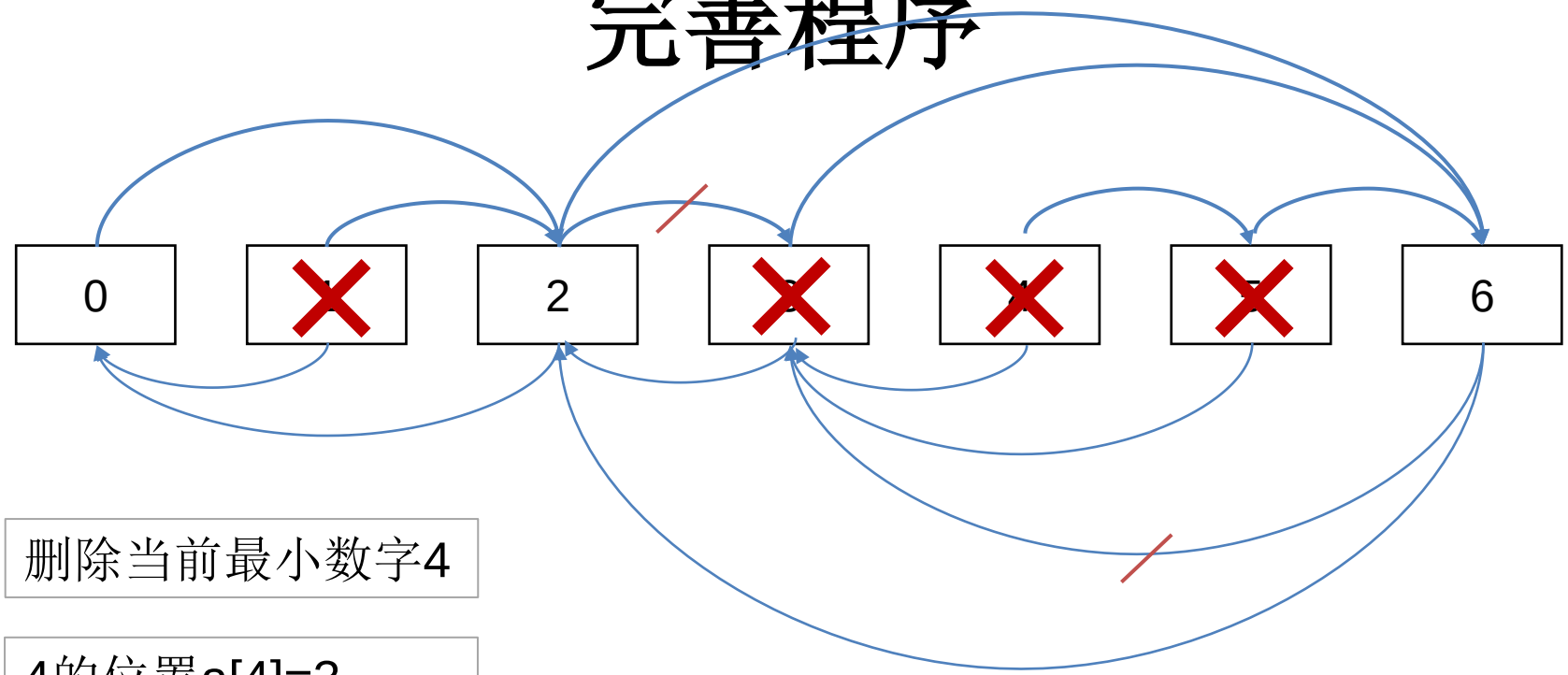
3的位置a[3]=5

从链表中删除a[i]

a[i]的后继的前驱变成a[i]的前驱

a[i]的前驱的后继变成a[i]的后继

完善程序



删除当前最小数字4

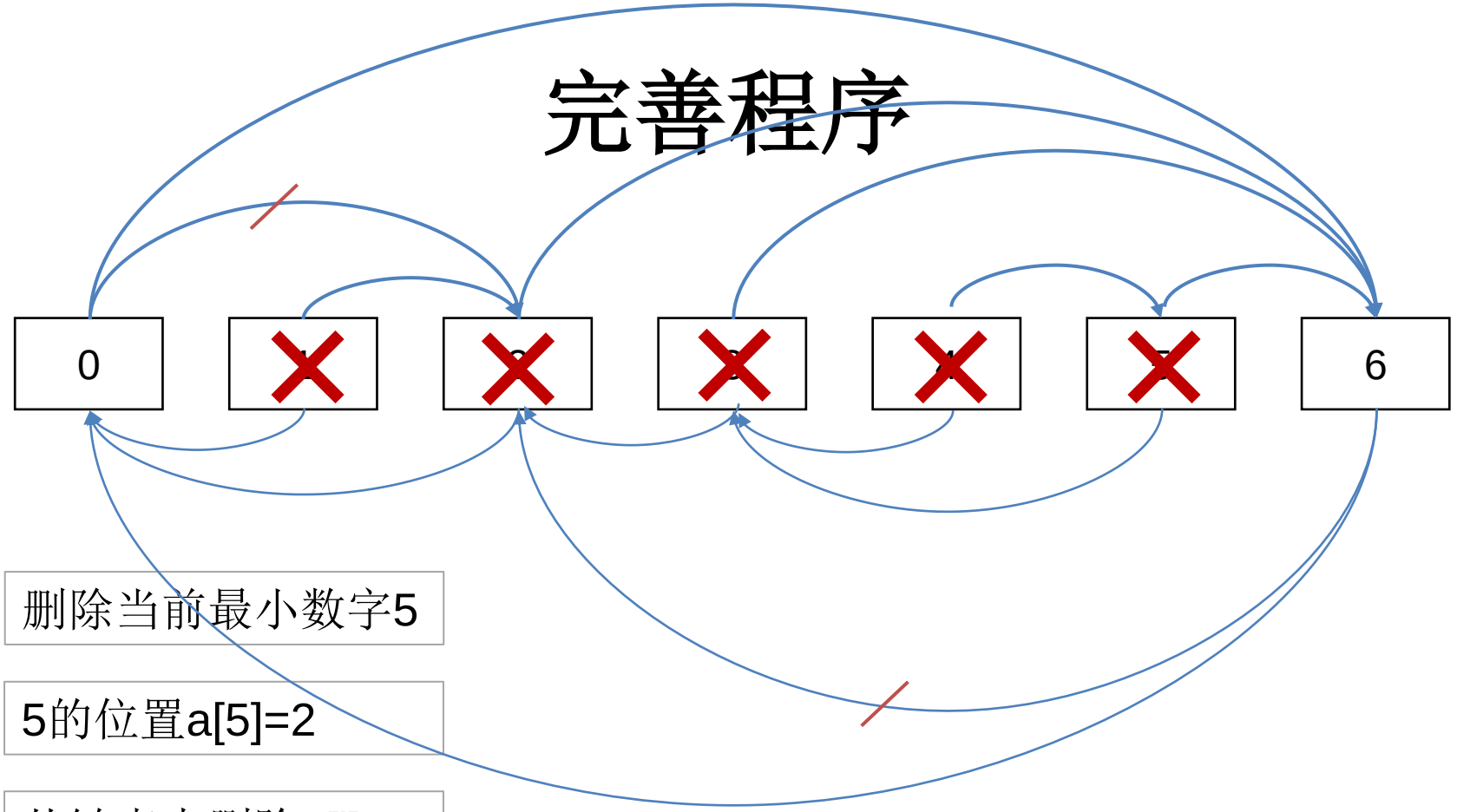
4的位置a[4]=3

从链表中删除a[i]

a[i]的后继的前驱变成a[i]的前驱

a[i]的前驱的后继变成a[i]的后继

完善程序



删除当前最小数字5

5的位置a[5]=2

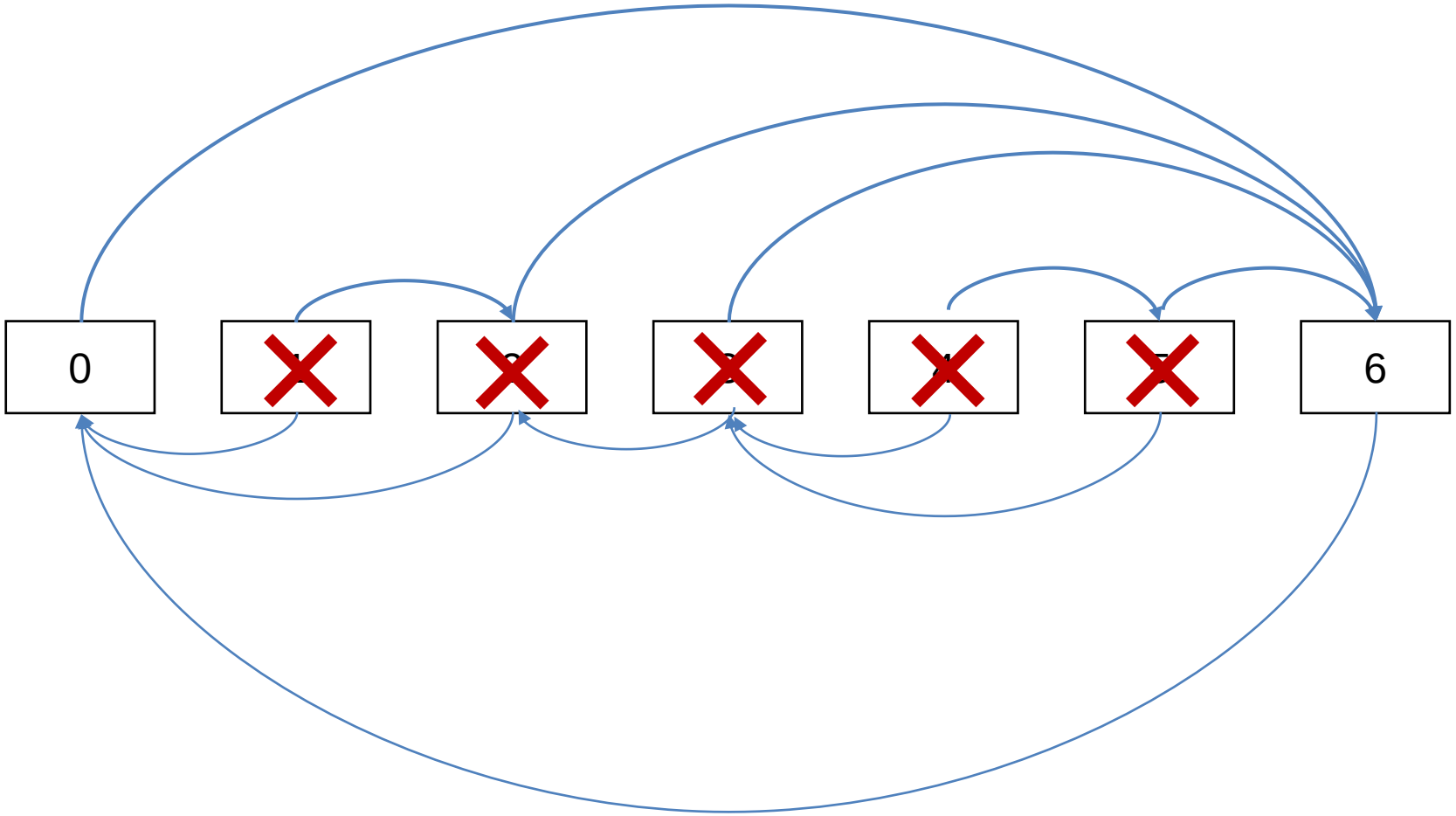
从链表中删除a[i]

a[i]的后继的前驱变成a[i]的前驱

a[i]的前驱的后继变成a[i]的后继

完善程序

输出：	2	6	6	5	6



完善程序

```
1  #include<iostream>
2  using namespace std;
3  const int N=10010;
4  int n;
5  int L[N], R[N], a[N];
6  int main() {
7      cin>>n;
8      for(int i=1; i<=n; ++i) {
9          int x;
10         cin>>x;
11         ______(1) ______;
12     }
13     for(int i=1; i<=n; ++i) {
14         R[i] = ______(2) ______;
15         L[i] = i-1;
16     }
17     for(int i=1; i<=n; ++i) {
18         L[______(3) ______] = L[a[i]];
19         R[L[a[i]]] = R[______(4) ______];
20     }
21     for(int i=1; i<=n; ++i)
22         cout<<______(5) ______<<" ";
23     cout<<endl;
24     return 0;
25 }
```

关键语句

a[i]=x还是a[x]=i
如果a[i]=x为什么不直接cin>>a[i]

记录数字x在序列中的位置

完善程序

```
#include<iostream>
using namespace std;
const int N=10010;
int n;
int L[N], R[N], a[N];
int main() {
    cin>>n;
    for(int i=1; i<=n; ++i) {
        int x;
        cin>>x;
        ______(1) ______;
    }
    for(int i=1; i<=n; ++i) {
        R[i] = ______(2) ______;
        L[i] = i-1;
    }
    for(int i=1; i<=n; ++i) {
        L[______(3) ______] = L[a[i]];
        R[L[a[i]]] = R[______(4) ______];
    }
    for(int i=1; i<=n; ++i)
        cout<<______(5) ______<<" ";
    cout<<endl;
    return 0;
}
```

关键语句

右侧第一个比位置i上数字大的位置

a[i]、L[i]还是R[i]?

完善程序

```
1  #include<iostream>
2  using namespace std;
3  const int N=10010;
4  int n;
5  int L[N], R[N], a[N];
6  int main() {
7      cin>>n;
8      for(int i=1; i<=n; ++i) {
9          int x;
10         cin>>x;
11         ______(1) ______;
12     }
13     for(int i=1; i<=n; ++i) {
14         R[i] = ______(2) ______;
15         L[i] = i-1;
16     }
17     for(int i=1; i<=n; ++i) {
18         L[______(3) ______] = L[a[i]];
19         R[L[a[i]]] = R[______(4) ______];
20     }
21     for(int i=1; i<=n; ++i)
22         cout<<______(5) ______<<" ";
23     cout<<endl;
24     return 0;
25 }
```

关键语句

位置i的后继为i+1

位置i的前驱为i-1

完善程序

```
1  #include<iostream>
2  using namespace std;
3  const int N=10010;
4  int n;
5  int L[N], R[N], a[N];
6  int main() {
7      cin>>n;
8      for(int i=1; i<=n; ++i) {
9          int x;
10         cin>>x;
11         _____(1) _____;
12     }
13     for(int i=1; i<=n; ++i) {
14         R[i] = _____(2) _____;
15         L[i] = i-1;
16     }
17     for(int i=1; i<=n; ++i) {
18         L[_____(3) _____] = L[a[i]];
19         R[L[a[i]]] = R[_____(4) _____];
20     }
21     for(int i=1; i<=n; ++i)
22         cout<<_____(5) _____<<" ";
23     cout<<endl;
24     return 0;
25 }
```

关键语句

L[a[i]]数字i所在位置在序列的前驱

R[L[a[i]]数字i所在位置序列位置的前驱的后继

完善程序

```
1  #include<iostream>
2  using namespace std;
3  const int N=10010;
4  int n;
5  int L[N], R[N], a[N];
6  int main() {
7      cin>>n;
8      for(int i=1; i<=n; ++i) {
9          int x;
10         cin>>x;
11         ______(1) ______;
12     }
13     for(int i=1; i<=n; ++i) {
14         R[i] = ______(2) ______;
15         L[i] = i-1;
16     }
17     for(int i=1; i<=n; ++i) {
18         L[______(3) ______] = L[a[i]];
19         R[L[a[i]]] = R[______(4) ______];
20     }
21     for(int i=1; i<=n; ++i)
22         cout<<______(5) ______<<" ";
23     cout<<endl;
24     return 0;
25 }
```

代码段作用

输入记录x出现的位置

构建静态双链表

删除节点

输出后继

完善程序

```
1  #include<iostream>
2  using namespace std;
3  const int N=10010;
4  int n;
5  int L[N], R[N], a[N];
6  int main() {
7      cin>>n;
8      for(int i=1; i<=n; ++i) {
9          int x;
10         cin>>x;
11                 1          ;
12     }
13     for(int i=1; i<=n; ++i) {
14         R[i] = _____(2) _____;
15         L[i] = i-1;
16     }
17     for(int i=1; i<=n; ++i) {
18         L[_____(3) _____] = L[a[i]];
19         R[L[a[i]]] = R[_____(4) _____];
20     }
21     for(int i=1; i<=n; ++i)
22         cout<<_____(5) _____<<" ";
23     cout<<endl;
24     return 0;
25 }
```

输入数字x出现的位置

完善程序

```
1  #include<iostream>
2  using namespace std;
3  const int N=10010;
4  int n;
5  int L[N], R[N], a[N];
6  int main() {
7      cin>>n;
8      for(int i=1; i<=n; ++i) {
9          int x;
10         cin>>x;
11         a[x] = i;
12     }
13     for(int i=1; i<=n; ++i) {
14         R[i] = i+1;
15         L[i] = i-1;
16     }
17     for(int i=1; i<=n; ++i) {
18         L[_____(3) _____] = L[a[i]];
19         R[L[a[i]]] = R[_____(4) _____];
20     }
21     for(int i=1; i<=n; ++i)
22         cout<<_____(5) _____<<" ";
23     cout<<endl;
24     return 0;
25 }
```

1

2

创建双链表

完善程序

```
#include<iostream>
using namespace std;
const int N=10010;
int n;
int L[N], R[N], a[N];
int main() {
    cin>>n;
    for(int i=1; i<=n; ++i) {
        int x;
        cin>>x;
        a[x] = i;
    }
    for(int i=1; i<=n; ++i) {
        R[i] = i+1;
        L[i] = i-1;
    }
    for(int i=1; i<=n; ++i) {
        L[R[a[i]]] = L[a[i]];
        R[L[a[i]]] = R[a[i]];
    }
    for(int i=1; i<=n; ++i)
        cout<<_____(5) _____<<" ";
    cout<<endl;
    return 0;
}
```

1

2

3

4

从小到大删除数字

完善程序

```
#include<iostream>
using namespace std;
const int N=10010;
int n;
int L[N], R[N], a[N];
int main() {
    cin>>n;
    for(int i=1; i<=n; ++i) {
        int x;
        cin>>x;
        a[x] = i;
    }
    for(int i=1; i<=n; ++i) {
        R[i] = i+1;
        L[i] = i-1;
    }
    for(int i=1; i<=n; ++i) {
        L[a[i]] = L[a[i]];
        R[L[a[i]]] = R[a[i]];
    }
    for(int i=1; i<=n; ++i)
        cout<<R[i]<<" ";
    cout<<endl;
    return 0;
}
```

输出i的后继

作业：完善程序

626 坐标统计

229 子矩阵

2007 静态双链表