问题 A: 1234 方阵

编程打印如下规律的 n*n 方阵。输入 n，按规律输出方阵。

方阵规律如下图：使左对角线和右对角线上的元素为 0,它们上方的元素为 1,左方的元素为 2,下方元素为 3,右方元素为 4,下图是一个符合条件的 5 阶矩阵。

```
0 1 1 1 0
2 0 1 0 4
2 2 0 4 4
2 0 3 0 4
0 3 3 3 0
```

输入格式

正整数 n（<=100）。

输出格式

所需的方阵。

输入	输出
10	0 1 1 1 1 1 1 1 1 0 2 0 1 1 1 1 1 1 0 4 2 2 0 1 1 1 1 0 4 4 2 2 2 0 1 1 0 4 4 4 2 2 2 2 0 0 4 4 4 4 2 2 2 2 0 0 4 4 4 4 2 2 2 0 3 3 0 4 4 4 2 2 0 3 3 3 3 0 4 4 2 0 3 3 3 3 3 3 0 4 0 3 3 3 3 3 3 3 3 0
5	0 1 1 1 0 2 0 1 0 4 2 2 0 4 4 2 0 3 0 4 0 3 3 3 0

问题 B: 果树

ZZ 有一个美丽的花园，那里生长着很多果树，每年都会收获丰收。但最近小偷开始在晚上潜入花园，经常偷水果。ZZ 不能在花园里过夜，看守水果，因为花园里没有房子！ZZ 攒了一段时间，最后他决定盖房子。剩下的很简单：他应该选择在花园的哪个部分建造房子。晚上，他坐在桌前，画了花园的平面图。在平面图上，花园表示为一个长方形的方格田，大小为 n×m，每个方格田是边长为 1 的正方形。每个方格田最多只能种植一棵树，也可以不种树。ZZ 想找到一个大小为 a×b 正方形的矩形地块来建造房屋，此时地块边界应沿着分隔正方形的网格线。所有生长在建筑地块上的树木都必须被砍掉。ZZ 非常喜欢他的花园，所以帮他选择建筑地块的位置，这样砍伐的树木的数量就会尽可能少。

输入格式

第一行包含两个整数 n 和 m （1≤n，m≤50），它们表示花园位置。接下来的 n 行包含 m 个数字 0 或 1，用于描述方案中的花园。零表示一棵树不在这个正方形上生长，1 表示有一棵正在生长的树。最后一行包含两个整数 a 和 b （1≤a，b≤50）。 请注意，ZZ 可以选择构建 a×b 矩形以及 b×a 矩形，即长度为 a 的地块一侧可以平行于长度为 n 的花园一侧，也可以平行于长度为 m 的花园一侧。

输出格式

打印需要砍伐的最小树木数量，以选择大小为 a×b 的地块来建造房屋。保证始终可以找到至少一个批次位置，即 a≤n 和 b≤m，或 a≤m，b≤n。

输入	输出
2 2 1 0 1 1 1 1	0
4 5 0 0 1 0 1 0 1 1 1 0 1 0 1 0 1 1 1 1 1 1 2 3	2

数据范围与提示

在第二个示例中，左上角的正方形是 （1，1），右下角的正方形是 （3，2）

问题 C: 挤奶

John 正在考虑改变他给奶牛挤奶的时候分配牛奶桶的方式。他认为这最终能使得他使用数量更少的桶，然而他不清楚具体是多少。请帮助他！

John 有 N 头奶牛（1≤N≤100），方便起见编号为 1…N。 第 i 头奶牛需要从时间 si 到时间 ti 之间挤奶，并且挤奶过程中需要用到 bi 个桶。于是多头奶牛可能在同一时刻都在挤奶；如果这样，他们不能使用相同的桶。也就是说，一个在第 i 头奶牛挤奶时用的桶不可以被任何在时间 si 到时间 ti 之间挤奶的其他奶牛使用。当然，这个桶在这段时间之外可以被其他奶牛所使用。为了简化他的工作，保证在任一时刻，至多只有一头奶牛开始或是结束挤奶（也就是说，所有的 si 和 ti 各不相同）。

John 有一个储藏室，里面有依次编号为 1、2、3、……的桶。在他的挤奶策略中，当某一头奶牛（比如说，奶牛 i）开始挤奶（在时间 si），FJ 就跑到储藏室取出编号最小的 bi 个桶分配给第 i 头奶牛用来挤奶。

请求出 John 需要在储藏室中存放多少个桶才能使得他能够顺利地给所有奶牛挤奶。

输入的第一行包含 N。以下 N 行，每行描述了一头奶牛，包含三个空格分隔的数 si，ti，和 bi。 其中 si 和 ti 均为 1…1000 之间的整数，bi 为 1…10 之间的整数。

1≤N≤100，

输出格式

输出一个整数，为 FJ 需要的桶的数量。

输入	输出
3 4 10 1 8 13 3 2 6 2	4

在这个例子中，FJ 需要 4 个桶：他用桶 1 和桶 2 来给奶牛 3 挤奶（从时刻 2 开始）。他用桶 3 给奶牛 1 挤奶（从时刻 4 开始）。

当奶牛 2 在时刻 8 开始挤奶时，桶 1 和桶 2 可以再次利用，然而桶 3 不可以，所以他会使用桶 1、桶 2 和桶 4。

问题 D:　吃草大会

一场别开生面的牛吃草大会就要在 Farmer John 的农场举办了！
世界各地的奶牛将会到达当地的机场，前来参会并且吃草。具体地说，有 N 头奶牛到达了机场（$1 \leqslant N \leqslant 10^5$），其中奶牛 i 在时间 ti（$0 \leqslant ti \leqslant 10^9$）到达。Farmer John 安排了 M（$1 \leqslant M \leqslant 10^5$）辆大巴来机场接这些奶牛。每辆大巴可以乘坐 C 头奶牛（$1 \leqslant C \leqslant N$）。Farmer John 正在机场等待奶牛们到来，并且准备安排到达的奶牛们乘坐大巴。当最后一头乘坐某辆大巴的奶牛到达的时候，这辆大巴就可以发车了，大巴车没有坐满也可以发车。Farmer John 想要做一个优秀的主办者，所以并不想让奶牛们在机场等待过长的时间。如果 Farmer John 合理地协调这些大巴，等待时间最长的奶牛等待的时间的最小值是多少？一头奶牛的等待时间等于她的到达时间与她乘坐的大巴的发车时间之差。输入保证 $MC \geqslant N$。

输入的第一行包含三个空格分隔的整数 N，M，和 C。第二行包含 N 个空格分隔的整数，表示每头奶牛到达的时间。

输出一行，包含所有到达的奶牛中的最大等待时间的最小值。

输入	输出
6 3 2 1 1 10 14 4 3	4

如果两头时间 1 到达的奶牛乘坐一辆巴士，时间 3 和时间 4 到达的奶牛乘坐乘坐第二辆，时间 10 和时间 14 到达的奶牛乘坐第三辆，
那么等待时间最长的奶牛等待了 4 个单位时间（时间 10 到达的奶牛从时间 10 等到了时间 14）。

问题 E: 桃花树

编程大神 zz 后院里种了 n 棵桃花树，每棵都有美学值 Ci(0≤Ci≤200)。zz 大神在每天上学前都会来赏花。zz 大神不仅是信息学霸，也是生物学霸，他懂得如何欣赏桃花：一种桃花树看一遍过，一种桃花树最多看 Pi(0≤Pi≤100) 遍，一种桃花树可以看无数遍。但是看每棵桃花树都有一定的时间 Ti(0≤Ti≤100)。zz 大神离去上学的时间只剩下一小会儿了。求解看哪几棵桃花树能使美学值最高且 z 大神能准时（或提早）去上学。

输入共 n+1 行：第 1 行：现在时间 Ts （几时：几分），去上学的时间 Te （几时：几分），zz 大神院子里有几棵桃花树 n。这里的 Ts ，Te 格式为：hh:mm，其中 0≤hh≤23，0≤mm≤59，0≤mm≤59，且 hh,mm,nhh,mm,n 均为正整数。

第 2 行到第 n+1 行，每行三个正整数：看完第 i 棵树的耗费时间 Ti ，第 i 棵树的美学值 Ci，看第 i 棵树的次数 Pi（Pi=0 表示无数次，Pi 是其他数字表示最多可看的次数 Pi ）。

输出只有一个整数，表示最大美学值。

输入	输出
6:50 7:00 3 2 1 0 3 3 1 4 5 4	11

提示

Te−Ts≤1000（即开始时间距离结束时间不超过 1000 分钟），n≤10000。保证 Te,Ts 为同一天内的时间。

样例解释：赏第一棵桃花树一次，赏第三棵桃花树 2 次。

后面还有程序阅读

阅读程序 1

```
#include <iostream>
#include <string>
using namespace std;
int lefts[20], rights[20], father[20];
string s1, s2, s3;
int n, ans;
void calc(int x, int dep)
{
    ans = ans + dep*(s1[x] - 'A' + 1);
    if (lefts[x] >= 0) calc(lefts[x], dep+1);
    if (rights[x] >= 0) calc(rights[x], dep+1);
}
void check(int x)
{
    if (lefts[x] >= 0) check(lefts[x]);
    s3 = s3 + s1[x];
    if (rights[x] >= 0) check(rights[x]);
}
void dfs(int x, int th)
{
    if (th == n)
    {
        s3 = "";
        check(0);
        if (s3 == s2)
        {
            ans = 0;
            calc(0, 1);
            cout<<ans<<endl;
      }
      return;
  }
  if (lefts[x] == -1 && rights[x] == -1)
  {
      lefts[x] = th;
```

```
36        father[th] = x;
37        dfs(th, th+1);
38        father[th] = -1;
39        lefts[x] = -1;
40    }
41    if (rights[x] == -1)
42    {
43        rights[x] = th;
44        father[th] = x;
45        dfs(th, th+1);
46        father[th] = -1;
47        rights[x] = -1;
48    }
49    if (father[x] >= 0)
50        dfs(father[x], th);
51 }
52 int main( )
53 {
54     cin>>s1;
       cin>>s2;
55     n = s1.size();
57     memset(lefts, -1, sizeof(lefts));
58      memset(rights, -1, sizeof(rights));
59     memset(father, -1, sizeof(father));
60     dfs(0, 1);
61 }
```

判断题

(1)将 23 行去掉,程序运行结果与原来一致. ( )

(2)该程序时间复杂度为 $O(n^3)$。 ( )

(3)此程序主要执行的算法思路深度优先搜索. ( )

●选择题

(4)（4 分）当输入为 ABCDEF\n BCAEDF,输出为( )。

A. 54 B. 55 C. 56 D. 57

(5) （4 分)当输入 S1 为 AAAAAAAAAA 时,将第 29 行改成输出最小值,则最小值是多少( )

A. 28 B. 29 C. 30 D. 31

(6) （4 分)当输入 S1 为 AAAAAAAAAA 时,将第 29 行改成输出最大值,输出的最大值是多 少( )

A. 54 B. 55 C. 56 D. 57

完善程序 2：

2. （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。

```
#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)
{
    int cur;
    if (root == 0)
        return 1;
    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)<<endl;
    return 0;
}
```

选择题

(1)1 处应填(　　)。

A.cur<upper_bound　B.cur<=upper_bound　C.cur>upperbound　D.cur>=upper_bound

(2)2 处应填(　　)。

A. left_child　　B.right_child　　C. a[root].left_child　　D. a[root].right_child

(3)3 处应填(　　)。

A. left_child　B. right_child　　C. cur+1　　D.cur

(4)④处应填(　　)。

A. left_child　　B. upper_bound-1　　C. right_child　　D.upper_bound

(5)⑤处应填(　　)。

A. -1　　B.0　　C. 1　　D.2