

目录

一、 质因数.....	3
1、 数的分类.....	3
2、 质数判断.....	3
3、 最大公约数.....	5
4、 最小公倍数.....	5
5、 练习.....	6
二、 排序.....	7
1、 排序算法介绍.....	7
2、 冒泡排序.....	8
3、 计数排序.....	9
4、 快速排序.....	10
5、 sort()函数.....	11
6、 练习.....	11
三、 结构体.....	13
1、 结构体概念.....	13
2、 结构体的定义.....	13
3、 结构体的使用.....	14
4、 练习.....	14
四、 二维数组.....	16
1、 二维数组的概念.....	16
2、 二维数组的数据读入.....	17
3、 课堂练习.....	17
4、 课后练习.....	19
五、 指针.....	22
1、 指针的定义.....	22
2、 指针实现地址拷贝.....	22
3、 &、*、及它们的连用.....	23
4、 优先级.....	24
5、 指针的作用.....	24
6、 函数多返回值.....	25
8、 数组的指针应用.....	26
7、 函数中通过指针控制变量的值.....	26
9、 练习.....	27
六、 进制转换.....	28
1、 进位计数制.....	28
2、 常见进制.....	28
3、 常见进制转换.....	29
4、 课堂练习.....	29
5、 课后练习.....	30
七、 高精度计算.....	31
1、 高精度计算的概念.....	31
2、 加法.....	31

3、减法.....	32
4、乘法.....	33
5、除法（高精度大整数除以 int 型整数）	34
6、课堂练习.....	34
八、贪心算法.....	35
1、贪心算法的概念.....	35
2、贪心算法过程.....	35
3、贪心算法不一定是最优解.....	35
4、课堂练习.....	36
九、递推算法.....	38
1、递推介绍.....	38
2、数塔问题.....	38
3、课堂练习.....	40
4、课后练习.....	41
十、递归算法.....	42
1、递归介绍：	42
2、递归算法通用解决思路.....	42
3、递归算法一般用于解决三类问题：	43
4、课堂练习.....	43
5、课后练习.....	44
十一、深度优先搜索.....	46
1、深度优先搜索的特点.....	46
2、深搜常见题型.....	46
3、回溯算法.....	46
4、回溯与深搜的区别.....	47
5、回溯法算法的两个框架.....	47
6、课堂练习.....	48
7、课后练习.....	51
十二、广度优先搜索.....	53
1、广度优先搜索的特点.....	53
2、深搜和广搜的区别.....	53
3、线性表-队列.....	53
4、广搜模板.....	54
5、课堂练习.....	54
6、课后练习.....	55
十三、栈.....	58
1、栈是什么.....	58
2、栈的操作.....	58
3、STL 容器适配器-stack.....	59
4、练习.....	60

一、质因数

1、数的分类



2、质数判断

质数概念：

质数又称素数。一个大于 1 的自然数，除了 1 和它自身外，不能被其他自然数整除的数叫做质数；否则称为合数。

质因数概念：

质因数是指能整除给定正整数的质数，同时也是这个数的约数。

质数的判断：

(1) 朴素法

利用 for 循环从 2 到 $n-1$ 依次判断是否能够整除 n ，如果有一次能够整除则为合数，否则为质数。

➤ 朴素法优化

利用 for 循环从 2 到 \sqrt{n} 依次判断是否能够整除 n, 如果有一次能够整除则为合数, 否则为质数。

➤ 朴素法可以这样优化的原因?

课堂练习: isPrime.cpp

输入一个大于 2 且小于 10000 的正整数 n, 如果它是质数就输出自身, 如果不是请输出这个数的最小质因数

(2) 埃拉托斯特尼筛法 (eratosthenes.cpp)

➤ 基本思想: 质数的倍数一定不是质数

➤ **实现方法:** 用一个长度为 N+1 的数组保存信息 (true 表示质数, false 表示非质数), 先假设所有的数都是质数 (初始化为 true), 从第一个质数 2 开始, 把 2 的倍数都标记为非质数 (置为 false), 一直到大于 N; 然后进行下一趟, 找到 2 后面的下一个质数 3, 进行同样的处理, 直到最后, 数组中依然为 true 的数即为质数。

➤ 时间复杂度 $O(N \log \log N)$

➤ 空间复杂度为 $O(N)$

➤ 埃式筛法的编程实现?

(1) 欧拉筛法 (euler.cpp)

➤ **优化:** 埃氏筛法的缺陷: 对于一个合数, 有可能被筛多次。例如 $30 = 2 * 15 = 3 * 10 = 5 * 6 \dots$ 那么如何确保每个合数只被筛选一次呢?

➤ **基本思想:** 在埃氏筛法的基础上, 让每个合数只被它的最小质因子筛选一次, 以达到不重复的目的。

3、最大公约数

概念：

最大公因数，也称最大公约数、最大公因子，指两个或多个整数共有约数中最大的一个。

最大公约数求法：

两个整数的最大公约数是能够同时整除它们的最大的正整数。辗转相除法基于如下原理：两个整数的最大公约数等于其中较小的数和两数的余数的最大公约数。

算法：

```
int Gcd(int a, int b) {  
    if(b == 0) {  
        return a;  
    }  
    return Gcd(b, a % b);  
}
```

4、最小公倍数

概念：

两个或两个以上的数公有的倍数叫做这几个数的公倍数，其中最小的一个叫做这几个数的最小公倍数。

最小公倍数的求法：

1) 枚举法求最小公倍数：

判断最大数能否分别整除两个数，否则让最大数每次增加一倍

2) 利用最大公约数求最小公倍数：

公式：

$$a * b = \text{gcd}(a, b) * \text{lcm}(a, b)$$

已知其三即可求其一。

5、练习

1. 可重复的质因数分解 primeDupli1.cpp

$$36 = 2 * 2 * 3 * 3$$

$$32 = 2 * 2 * 2 * 2 * 2$$

2. 不可重复的质因数分解 primeDupli2.cpp

$$36 : 2, 3$$

$$32 : 2$$

3. 求差 lgsub.cpp

输入两个数的值，求两个数的最小公倍数和最大公约数的差

二、排序

1、排序算法介绍

概念：

排序是将一批无序的记录(数据)重新排列成按关键字有序的记录序列的过程。

作用：

- 无序的数据组合变成有序的数据。
- 有序的数据利于数据定位和处理。

常见排序算法：

- 快速排序
- 插入排序
- 选择排序
- 桶排序
- 基数排序
- 归并排序
- 拓扑排序
- 堆排序
- ...

稳定性：

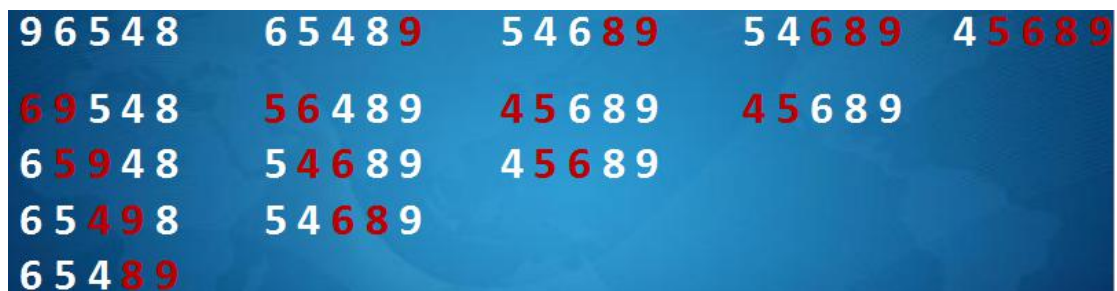
即当两个相同的元素同时出现于某个序列之中，则经过一定的排序算法之后，两者在排序前后的相对位置不发生变化。换言之，即便是两个完全相同的元素，它们在排序过程中也是各有区别的，不允许混淆不清。

- 初始状态：1 69(1) 69(2) 78 2 8 45
- 稳定排序：1 2 8 45 69(1) 69(2) 78
- 不稳定排序：1 2 8 45 69(2) 69(1) 78

2、冒泡排序

冒泡排序算法的原理：

- 1、比较相邻的元素。如果第一个比第二个大，就交换他们两个。
- 2、对每一对相邻元素做同样的工作，从开始第一对到结尾的最后一对。最后的元素会是最大的数。
- 3、针对所有的元素重复以上的步骤，除了最后一个。持续每次对越来越少的元素重复上面的步骤，直到没有任何一对数字需要比较。



这个算法的名字由来是因为越小的元素会经由交换慢慢“浮”到数列的顶端（升序或降序排列），就如同碳酸饮料中二氧化碳的气泡最终会上浮到顶端一样，故名“冒泡排序”。

课堂练习 1: bubbleSort.cpp

输入 n ($1 < n \leq 1000$)

再输入 n 个数进行降序排序，用冒泡排序实现。

课堂练习 2: bubbleSort2.cpp

将一组数据去重后升序输出。

3、计数排序

计数排序：一种基于数组下标的排序方法。

- 利用了数组下标有序的性质
- 但排序数据的值域受限于数组的长度
- 排序后可自动去重

计数排序中数据的存储方式：

		排序 1 2 2 5 6 3 3 3 ?						
初始状态：	值：	0	0	0	0	0	0	0
	下标：	0	1	2	3	4	5	6
结束状态：	值：	0	1	2	3	0	1	1
	下标：	0	1	2	3	4	5	6

计数排序中数据的输出：

只需要按照数组下标顺序依次输出值不为 0 的下标即可。

实现方法：

```
9 void ParseIn () {
10     int curInt = 0;
11     while(cin >> curInt) {
12         _myList[curInt]++;
13     }
14 }
15
16 void Core() {
17     for(int i = 0; i < 10000; i++) {
18         if(_myList[i] != 0) {
19             cout << i << " ";
20         }
21     }
22 }
```

思考：不去重的计数排序怎么写？

4、快速排序

分治法的基本思想：

分治法的基本思想是：将原问题分解为若干个规模更小但结构与原问题相似的子问题。递归地解这些子问题，然后将这些子问题的解组合为原问题的解。

快速排序的基本思想：

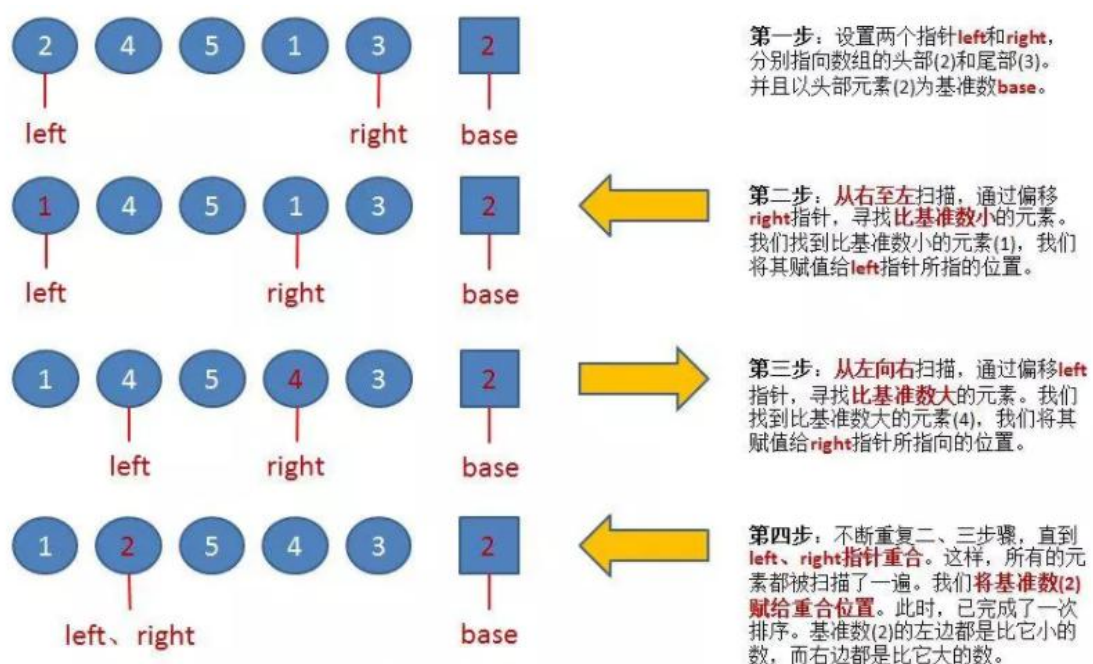
设当前待排序的无序区为 $R[\text{low} \dots \text{high}]$ ，利用分治法可将快速排序的基本思想描述为：

(1) 分解：

在 $R[\text{low} \dots \text{high}]$ 中任选一个值作为基准 (pivot)，以此基准将当前无序区划分为左、右两个较小的子区间 $R[\text{low} \dots \text{pivotPos}-1]$ 和 $R[\text{pivotPos}+1 \dots \text{high}]$ ，并使左边子区间中所有的值均小于等于基准 (pivot)，右边的子区间中所有值均大于等于 pivot，而基准 pivot 则位于正确的位置 (pivotPos) 上，它无须参加后续的排序。

(2) 求解：

通过递归调用快速排序对左、右子区间 $R[\text{low} \dots \text{pivotpos}-1]$ 和 $R[\text{pivotpos}+1 \dots \text{high}]$ 快速排序。



5、sort()函数

- 时间复杂度 $O(N\log N)$ ，内部实现：快排
- `#include <algorithm>`
- `Cmp()`
- `sort()` //不稳定排序
- `stable_sort()` //稳定排序

`sort(vector 容器中起始元素的迭代器, vector 容器中末尾元素的迭代器, Cmp 函数地址)` \\动态数组

`sort(起始地址, 起始地址+排序长度, Cmp 函数地址)` \\静态数组

6、练习

1. 奇数排序 bubbleSort3. cpp

给定一个长度为 n (不大于 500) 的正整数序列，请将其中所有奇数取出，并按降序输出。

输入：

10

1 3 2 6 5 4 9 8 7 10

输出：

9 7 5 3 1

2. 单词排序 word. cpp

输入一行单词序列，相邻单词之间由 1 个或多个空格间隔，请按照字典序输出这些单词。重复单词只输出一次

输入：

一行单词序列，最少一个单词，最多 100 个单词，每个单词长度不超过 50，单词之间用至少一个空格隔开，数据不含字符和空格以外的其他字符。

输出：

按字典序从小到大输出这些单词

样例输入：

She wants to go to Peking University to study Chinese

样例输出：

Chinese
Peking
She
University
go
study
to
wants

3. 整数奇偶排序 sort.cpp

给定 n 个整数的序列，要求对其重新排序。排序要求：

- (1) 奇数在前，偶数在后；
- (2) 奇数按从大到小排序；
- (3) 偶数按从小到大排序

输入：

一行，包含 n 个整数，彼此以一个空格分开，每个整数的范围是大于等于 0，小于等于 100

输出

按照要求排序后输出一行，包含排序后的 n 个整数，数与数之间以一个空格分开

样例输入：

4 7 3 13 11 12 0 47 34 98

样例输出：

47 13 11 7 3 0 4 12 34 98

三、结构体

1、结构体概念

- 结构体可以认为是我们自己定义的数据类型
- 这个类型可以容纳很多基础数据类型

2、结构体的定义

- 假设我们想在游戏中定义一个人物：
- 其人物属性有：血量、攻击力和人物名称
- 正确定义方式：

```
struct hero{  
    int blood;  
    int attack;  
    string name;  
};
```

- 在定义类型的时候注意**不要**提前赋值
- 错误的结构体定义：

```
struct hero{  
    int blood = 1;  
    int attack = 2;  
    string name = "yase";  
};
```

3、结构体的使用

使用结构体的时候可以理解为：使用自己创造的数据类型去定义变量。

例如：

```
7 struct hero{  
8     int blood;  
9     int attack;  
10    string name;  
11 };  
12 int main () {  
13     hero gailun;  
14     cin >> gailun.blood >> gailun.attack >> gailun.name;  
15     cout << gailun.blood << gailun.attack << gailun.name;  
16     return 0;  
17 }
```

➤ 自定义类型数组怎么写？

➤ 结构体排序如何实现？

➤ Compare 函数怎么更改？

4、练习

练习 1：

学生期末成绩：

姓名

语文

数学

输入 n 个同学的 2 门课成绩，并输出总分。

练习 2:

plant.cpp

问题描述:

校园的小树林里有 n 棵树，小成发现，每棵树上有个小牌子，写着这棵树是哪年种的。他想知道，哪一年种的最多。你可以帮忙吗？

输入文件:

文件的第一行，是一个整数 n ($1 \leq n \leq 100$)，代表 n 棵树。第二行，是 n 个空格隔开的正整数，依次表示每棵树的种植年份，数值在 1990 到 2021 之间。

输出文件:

文件共一行，是一个或若干个整数，表示种树最多的年份。如果答案不止一个，按从小到大的顺序输出，中间用空格隔开。

样例输入 1:

```
5
2010 2013 2014 2010 2014
```

样例输出 1:

```
2010 2014
```

样例输入 2:

```
7
1998 1998 2003 2002 2004 2004 1998
```

样例输出 2:

```
1998
```

四、二维数组

1、二维数组的概念

(1) 二维数组的定义

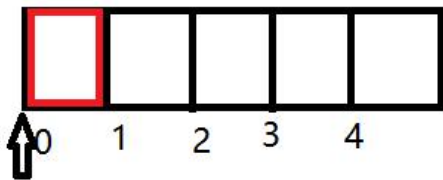
二维数组定义的一般形式是：`dataType arrayName[length1][length2]`。

其中，`dataType` 为数据类型，`arrayName` 为数组名，`length1` 为第一维下标的长度，`length2` 为第二维下标的长度。

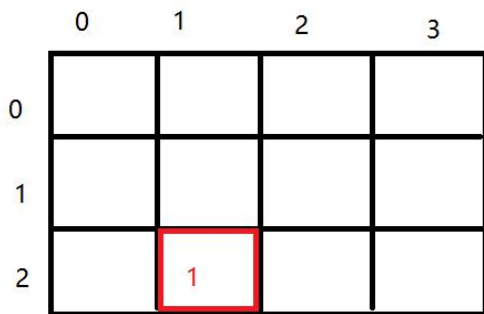
例如 `int mapp[10][10];`

(2) 一维数组与二维数组的区别

一维数组：



二维数组：



(3) 二维数组的初始化

二维数组的初始化可以按行分段赋值，也可按行连续赋值。

➤ 例如，对于数组 `a[5][3]`，按行分段赋值：

```
int mapp[5][3]={ {80,75,92}, {61,65,71}, {59,63,70}, {85,87,90}, {76,77,85} };
```

➤ 按行连续赋值：

```
int mapp[5][3]={80, 75, 92, 61, 65, 71, 59, 63, 70, 85, 87, 90, 76, 77, 85};
```


2、二维数组的数据读入

```
for(int i = 1; i <= n; i++) {  
    for(int j = 1; j <= m; j++) {  
        cin >> mapp[i][j]  
    }  
}
```

3、课堂练习

练习 1:

题目描述：输入整数 N ，输出相应方阵。

输入：一个整数 N 。（ $0 < n < 10$ ）

输出：一个方阵，每个数字的场宽为 3。

样例输入：

5

样例输出：

1 2 3 4 5

6 7 8 9 10

11 12 13 14 15

16 17 18 19 20

21 22 23 24 25

练习 2:

题目描述：输入整数 N ，输出相应方阵。

输入：一个整数 N 。（ $0 < n < 10$ ）

输出：一个方阵，每个数字的场宽为 3。

样例输入：

5

样例输出：

1 0 0 0 0

0 1 0 0 0

0 0 1 0 0

0 0 0 1 0

0 0 0 0 1

练习 3:

题目描述：输入整数 N ，输出相应方阵。

输入：一个整数 N 。（ $0 < n < 10$ ）

输出：一个方阵，每个数字的场宽为 3。

样例输入：

5

样例输出：

1 2 3 4 5

2 3 4 5 6

3 4 5 6 7

4 5 6 7 8

5 6 7 8 9

练习 4:

扫雷游戏是一款十分经典的单机小游戏。在 n 行 m 列的雷区中有一些格子含有地雷（称之为地雷格），其他格子不含地雷（称之为非地雷格）。玩家翻开一个非地雷格时，该格将会出现一个数字——提示周围格子中有多少个是地雷格。游戏的目标是在不翻出任何地雷格的条件下，找出所有的非地雷格。

现在给出 n 行 m 列的雷区中的地雷分布，要求计算出每个非地雷格周围的地雷格数。

注：一个格子的周围格子包括其上、下、左、右、左上、右上、左下、右下八个方向上与之直接相邻的格子。

输入格式：

第一行是用一个空格隔开的两个整数 n 和 m ，分别表示雷区的行数和列数。

接下来 n 行，每行 m 个字符，描述了雷区中的地雷分布情况。字符 '*' 表示相应格子是地雷格，字符 '?' 表示相应格子是非地雷格。相邻字符之间无分隔符。

输出格式：

输出文件包含 n 行，每行 m 个字符，描述整个雷区。用 '*' 表示地雷格，用周围的地雷个数表示非地雷格。相邻字符之间无分隔符。

（对于 100% 的数据， $1 \leq n \leq 100, 1 \leq m \leq 100$ 。）

输入样例：

3 3

*??

???

?*?

输出样例：

*10

221

1*1

练习 5:

明明有一天走到了一片苹果林，里面每颗树上都结有不同数目的苹果，明明身上只能拿同一棵树上的苹果，他每到一棵果树前都会把自己身上的苹果扔掉并摘下他所在树上的苹果并带走（假设明明会走过每一棵苹果树），问在明明摘苹果的整个过程中，他身上携带的最多苹果数与最小苹果数的差是多少？

输入：

m, n（即苹果林中有果树的行数和列数， $0 < n, m \leq 10$ ） m 行 n 列数据（即每颗树上的苹果数）

输出：

1 个数字（明明摘苹果的整个过程中，他身上携带的最多苹果数与最小苹果数的差）

样例输入：

4 3

2 6 5

1 3 7

5 3 5

1 7 12

样例输出：

11

4、课后练习

练习 1：在一个数字方阵中查找“对称数”的个数。对称数：一个数的上和下两数相同或者左右两数相同， $n \leq 100$, 所有数据都 ≥ 0 。

样例输入：

4

4 0 6 2

9 7 6 1

6 2 1 5

2 3 6 3

样例输出：

2

练习 2:

给出两幅相同大小的黑白图像（用 0-1 矩阵，0 代表白色，1 代表黑色）表示，求它们的相似度。说明：若两幅图像在相同位置上的像素点颜色的值相同，则称它们在该位置具有相同的像素点。两幅图像的相似度定义为相同像素点数占总像素点数的百分比。数据解释：第一行的 2 2 表示图像的尺寸是 2 行，每行 2 个整

数；接下来的两行数据 1 0 和 0 1 表示第一幅图片的数值，再接下来两行数据 1 1 和 1 1 表示第二幅图片的数值；从数据上可以看出，两幅图片有 2 个数是相等的，因此两幅图片的相似度为 50%，实际输出不需要输出百分号，结果保留 2 位小数，因此实际输出 50.00。

输入：

第一行包含两个整数 n 和 m ，表示图像的行数和列数，中间用单个空格隔开。 $1 \leq n \leq 100$ ， $1 \leq m \leq 100$ 。

之后 n 行，每行 m 个整数 0 或 1，表示第一幅黑白图像上各像素点的颜色。相邻两个数之间用单个空格隔开。

之后 n 行，每行 m 个整数 0 或 1，表示第二幅黑白图像上各像素点的颜色。相邻两个数之间用单个空格隔开。

输出：

一个小数，表示相似度（以百分比的形式给出，但百分号不需要显式），精确到小数点后两位。

样例输入：

```
3 3
1 0 1
0 0 1
1 1 0
1 1 0
0 0 1
0 0 1
```

样例输出：

```
44.4
```

练习 3:

同学们在操场上排成了一个 n 行 m 列的队形，请将这个队形中，年龄最大的同学和年龄最小的同学交换位置，并输出交换的结果（本题数据保证年龄最大的同学和年龄最小的同学在矩阵中是唯一的）。比如：如下是一个 3 行 4 列的队形，这个队形中每个数字代表了每个同学的年龄。

```
8 10 18 9
15 12 10 6
17 3 12 15
```

这个队形中，年龄最大的同学在第 1 行第 3 列，年龄最小的同学在第 3 行第 2 列，将他们交换位置后输出结果为：

```
8 10 3 9
15 12 10 6
17 18 12 15
```

输入：

第 1 行有 2 个整数 n 和 m ，分别代表队形的行和列的值（ $2 \leq n, m \leq 200$ ）

接下来 n 行，每行有 m 个整数，代表每个同学的年龄（每个同学的年龄的值在 1~100 之间）。

输出：

输出 n 行 m 列，代表交换位置后的结果，每行的 m 个数之间用空格隔开。

样例输入：

3 4

8 10 18 9

15 12 10 6

17 3 12 15

样例输出：

8 10 3 9

15 12 10 6

17 18 12 15

五、指针

1、指针的定义

- 指针存储数据在**内存中的位置**，所以也叫指针变量
- 指针变量存储的是其指向**对象的地址**
- 指针指向的对象可以是变量、数组、函数等占据存储空间的实体。
- 先定义后使用

例如：

```
int num = 10;
```

```
int *p = &num;
```

```
5 int main () {  
6  
7     int num = 100;  
8     /*: 表示定义指针  
9     //p 是整数指针的变量 (int *), 存储  
10    //&: 获取变量的地址  
11    int *p = &num;  
12    //p是num的地址, 而*p是num的值  
13    cout << p << endl;  
14    cout << *p << endl;  
15  
16    //通过指针修改其指向的值  
17    *p = *p + 1;  
18    cout << *p << " " << num;  
19  
20    return 0;  
21 }
```

2、指针实现地址拷贝

- 值拷贝
- 地址拷贝

```

2  #include <iostream>
3  using namespace std;
4  int main () {
5      int num1 = 0;
6      int num2 = num1;
7      //值拷贝
8      num2++;
9      cout << num1 << endl;
10     int *p = &num1;
11     //地址拷贝
12     *p = *p + 1;
13     cout << num1;
14     return 0;
15 }

```

3、&、*、及它们的连用

(1) **&**: 是取地址运算符, 如有 `int a`; 即有一个小盒子里面存放的数据起名叫 `a`, `&a` 就是取 `a` 的地址, 即该盒子的编号。

(2) ***(地址)**: 是取值运算符, 这里 `*` 是解引用操作符, 可以理解成打开对应地址编号的盒子, 取出里面的数据。 `*(&a)` 就是打开 `a` 对应的小盒子, 取出里面的数据, 即 `*(&a)` 和 `a` 等价。

```

1  #include <iostream>
2
3  using namespace std;
4
5  int main () {
6      int num = 100;
7      int *p = &num;
8      cout << p << " " << *p << endl;
9
10     cout << &*p << endl; //取*p的地址
11     cout << *&*p << endl;
12     return 0;
13 }
14

```

4、优先级

一元运算符 `*` 和 `++` 具有相等的优先级别，但是在运算时它是从右向左顺序进行的。即，在 `*p++` 中，`++` 应用于 `p` 而不是应用于 `*p`，实现的是指针自增 1，而非数据自增 1。

```
2  #include <iostream>
3
4  using namespace std;
5
6  int main () {
7      int num = 100;
8      int *p = &num;
9      // *p = *p + 1;
10     //++的优先级比*的优先级高
11     *p++;
12     // (*p)++;
13     cout << num;
14     return 0;
15 }
```

5、指针的作用

- 有效地表示复杂的数据结构
- 动态分配内存
- 高效的使用数组和字符串
- 使得调用函数时得到多个返回值
- 其他作用
- 仅初赛中考查指针，复赛几乎避开对指针的使用
- `scanf` 和 `printf` 的使用


```

2 int main () {
3     int num1 = 100;
4     int num2 = 10;
5     // int _myList[10];
6     //scanf (格式, 地址);
7     // %d 代表整数, %f代表float, %lf 代表double %c 代表字符, %s代表字符串, %p代表指针
8     // scanf("%d", &num);
9     // printf("%d", num);
10    scanf("%d%d", &num1, &num2);
11    printf("%d\n", num1 + num2 );
12
13    int *p = &num1;
14    scanf(
15    //     for(int i = 0; i < 10; i++) {
16    //         scanf("%d", _myList + i);
17    //     }
18    //     for(int i = 0; i < 10; i++) {
19    //         printf("%d ", _myList[i]);
20    //     }
21    return 0;

```

6、函数多返回值

```

7 double fun(int a, int b, int *max, int *min) {
8     if(a > b) {
9         *max = a;
10        // max = &a;
11        *min = b;
12    }
13    else {
14        *max = b;
15        *min = a;
16    }
17
18    return (a + b) / 2.0;
19 }
20 int main () {
21     int a, b, max, min;
22     cin >> a >> b;
23     double r = fun(a, b, &max, &min);
24     cout << max << " " << min << " " << r << endl;
25     return 0;
26 }

```

8、数组的指针应用

```
1  #include <iostream>
2  #include <string.h>
3  using namespace std;
4  void show1 (char list[]) {
5      for(int i = 0; i < strlen(list); i++) {
6          cout << list[i] << " ";
7      }
8  }
9  void Show2 (char *s) {
10     while( *s != '\0') {
11         cout << *s << " ";
12         s++;
13     }
14 }
15 }
16 int main () {
17     char myList[] = "hello";
18     show1(myList);
19     show2(myList);
20     return 0;
21 }
```

7、函数中通过指针控制变量的值



划定一块区域，供程序存储变量使用

```

3 void fun1(int x) {
4     x++;
5 }
6 void fun2(int *p) {
7     (*p)++;
8 }
9 int main() {
10     int x = 1;
11     fun1(x);
12     cout << x << endl;
13
14     int p = 1;
15     fun2(&p);
16     cout << p << endl;
17
18     return 0;
19 }

```

9、练习

利用指针，编写一个用于交换两个整型变量值的函数。

输入：5 6

输出：6 5

六、进制转换

1、进位计数制

进位计数制：将数字符号按序排列成数位，并遵照某种由低位到高位进位的方法进行计数。

进位计数制的表示主要包含三个基本要素：

- 1、**数位：**指数码在一个数中所处的位置
- 2、**基数：**指在某种进位计数制中，每个数位上所能使用的数码的个数。
- 3、**位权：**指在某种进位计数制中，每个数位上的数码所代表的数值的大小。

2、常见进制

（1）二进制

二进制计数制简称二进制；有二个不同的数码符号：0、1。每个数码符号根据它在这个数中所处的位置（数位），按“逢二进一”来决定其实际数值，即各数位的位权是以 2 为底的幂次方。

（2）八进制

八进制计数制简称八进制；有八个不同的数码符号：0、1、2、3、4、5、6、7。每个数码符号根据它在这个数中所处的位置（数位），按“逢八进一”来决定其实际数值，即各数位的位权是以 8 为底的幂次方。

（3）十六进制

十六进制计数制简称十六进制；有十六个不同的数码符号：0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F。每个数码符号根据它在这个数中所处的位置（数位），按“逢十六进一”来决定其实际数值，即各数位的位权是以 16 为底的幂次方。

3、常见进制转换

(1) 10 进制转换

➤ R 转 10: 把 R 进制数按权展开、相加即得十进制数。

例如: $1011001 = 2^6 + 2^4 + 2^3 + 2^0$ 。

➤ 10 转 R: 除 R 求余法, 除以 R 取余, 逆序输出。

(2) 2 进制转换

➤ 二进制转 (八或十六) 进制:

以二转八进制为例: 左起每 3 位二进制数字转化成十进制数字, 不足 3 位前面补 0, 组合而成就是八进制。

➤ (八或十六) 转二进制:

以八转二进制为例: 每一位转化成 3 位二进制数字, 不足前面补充 0。

4、课堂练习

练习 1:

题目描述: 输入一个不大于 32767 的整数 n, 将它转换成一个二进制数

输入: 输入只有一行, 包括一个整数 n ($0 \leq n \leq 32767$)

输出: 只有一行

输入样例: 100

输出样例: 1100100

练习 2:

题目描述: 请将一个不超过 10 位的十六进制正整数转换为十进制整数

输入: 不超过 10 位的十六进制正整数

输出: 对应的十进制整数

输入样例: 2ECF

输出样例: 11983

练习 3:

小丽同学在编程中学到了回文数的概念，如果一个数正过来读和反过来读是同一个数，那么这个数就是回文数；比如：2、5、8、66、121、686、12321 都是回文数，小丽发现，这样的数不算多。于是小丽有个想法，如果这个数不是回文数，但这个数在 2 进制或者 16 进制下是回文数，就算这个整数是半个回文数，比如 417 并不是回文，但 417 对应的 16 进制数是 1A1 是回文数，因此 417 算半个回文数。请你编程帮助小丽找符合条件的半个回文数。

输入：

第一行是一个整数 n ($10 \leq n \leq 100$)

第二行是 n 个整数（这些整数都是 0~999999999 之间的整数）

输出：

所有符合条件的半个回文数，每行一个。

样例输入：

5

121 417 27 100 21

样例输出：

417

27

21

5、课后练习

练习 1:

题目描述：请将一个不超过 100 位的十六进制数转换成二进制数

输入：一个不超过 100 位的十六进制数

输出：该数对应的二进制数

输入样例：123456789ABCDEF

输出样例：100100011010001010110011110001001101010111100110111101111

练习 2:

题目描述：请从键盘读入一个非负整数 n (n 是一个不超过 18 位的正整数)，将 n 转换为 16 进制。

输入：一个不超过 18 位的非负整数 n

输出：该数的 16 进制

输入样例：100000000000

输出样例：174876E800

七、高精度计算

1、高精度计算的概念

我们都知道，int 类型变量最大值是 $2^{31}-1$ ，long long 类型变量最大值是 $2^{63}-1$ ，如果我们需要更大的整数计算呢？

高精度计算就是通过**数位**对大整数进行拆分计算。

2、加法

先写一个加法竖式：

$$\begin{array}{r} 9734 \\ + 597 \\ \hline 10331 \end{array}$$

整理下加法竖式的计算过程：

高精度加法就是用程序模拟加法竖式的计算过程
算法流程：

- (1) 字符串读入高精度整数
- (2) 将每一位逆序存入两个整数数组
- (3) 从左向右逐位计算
- (4) 逆序输出结果

例：

下标: 0 1 2 3 4

4	3	7	9	
7	9	5	0	
11	12	12	9	
1	3	3	0	1

最终结果: 1 0 3 3 1

3、减法

在高精度加法的基础上，需要考虑结果为负数的情况。
在此基础上思考减法竖式的计算过程：

$$\begin{array}{r} 734 \\ - 597 \\ \hline 137 \end{array}$$

算法流程:

- 判断正负，若 s1 比 s2 对应整数小，结果为负，交换两字符串，并提前输出 '-'。
- 将两个字符串，逆序存入两个整数数组。
- 从左到右，逐位相减，不够借位。
- 逆序输出处理，要考虑无效 0。

例：

下标 : 0	1	2	3
4	3	7	
7	9	5	
7	3	1	

4、乘法

类似加法，可以用竖式求乘法。在做乘法运算时，同样也有进位，但对每一位进行乘法运算时，必须执行错位相加。

例：

			5	6	7
	*			7	1
			5	6	7
+	3	9	6	9	
	4	0	2	5	7

算法流程：

- (1) 字符串读入高精度整数。
- (2) 将每一位逆序存入两个整数数组。
- (3) 从左向右错位相乘并计算。
- (4) 从左向右依次进位。
- (5) 逆序输出处理，要考虑无效 0。

例：

下标：0 1 2 3 4					
7	6	5			j
1	7				i
7	6	5			
	4	4	3		
	9	2	5		
7	5	4	3		
	5	7	5		
7	5	2	0	4	
4	0	2	5	7	

5、除法（高精度大整数除以 int 型整数）

例题：

一个大整数（不超过 300 位且大于 0）除以一个正整数（不超过 9 位），求商和余数

除法竖式部分：

$$\begin{array}{r} 0 2 1 0 \\ 36 \overline{) 7562} \\ \underline{0} \\ 75 \\ \underline{72} \\ 36 \\ \underline{36} \\ 2 \\ \underline{0} \\ 2 \end{array}$$

算法流程：

- （1）字符串转数组。
- （2）逐位整除、求余、退位。
- （3）正序输出结果数组并输出余数，需要考虑前面的无效 0。

6、课堂练习

求 2^n 的高精度值。

八、贪心算法

1、贪心算法的概念

- 贪心算法是一种对某些求最优解问题的更简单、更快的算法。
- 贪心算法的特点是一步一步地进行，**每一步只考虑当前的最优策略**，而不考虑各种可能的整体情况。
- 贪心算法采用自顶向下，以迭代的方法做出相继的贪心选择，每做一次贪心选择，就将所求问题简化为一个规模更小的子问题。
- 普及组中最常用的算法之一，一般会**先排序**。

2、贪心算法过程

贪心算法思路：

- (1) 建立数学模型来描述问题；
- (2) 把求解的问题分成若干个子问题；
- (3) 对每个子问题求解，得到子问题的局部最优解；
- (4) 把子问题的解局部最优解合成原来解问题的一个解。

3、贪心算法不一定是最优解

贪心算法产生的全局解有时不一定是最优的。



4、课堂练习

练习 1:

某工厂有 n 个零件加工的师傅，每位师傅每天能够加工出不同数量的零件。现有 m 个零件要求一天加工完，请问该工厂最少需要派几个师傅来完成这次零件加工任务，如果安排所有的师傅都参与加工也不能在一天内完成任务，请输出“NO”。

输入：

第一行有两个整数，用空格隔开；第一个整数代表要加工的总零件个数 m ($m \leq 10^6$)，第二个整数代表工厂的零件加工师傅的数量 n ($n \leq 100$)。

第二行有 n 个整数，分别代表每个师傅每天能够加工出来的零件数量（每个师傅每天加工的零件数量 $\leq 10^4$ ）。

输出：

工厂在 1 天时间内加工所有零件需要的师傅数量或者输出 NO。

样例输入：

10 5

1 3 2 4 2

样例输出：

4

练习 2:

有 n 个人排队到 r 个水龙头去打水，他们装满水桶的时间 t_1, t_2, \dots, t_n 为整数且各不相等，应如何安排他们的打水顺序才能使他们花费的总时间最少？每个人打水的时间 = 排队的时间 + 实际打水的时间，本题假设一个人打好水，排在他后面的人接着打水的这个切换过程不消耗时间。比如，有 2 个人 A 和 B，他们打水的时间分别是 3 和 2，只有 1 个水龙头，这时，如果 A 先打水，B 后打水，那么 A 和 B 打水的时间分别为 3、3+2（B 排队 3 分钟）。因此，所有人打水的总时间就是每个人的打水时间及每个人的排队时间的总和。

输入：

第 1 行，两个整数 n ($1 \leq n \leq 500$) 和 r ($1 \leq r \leq 100$)。

第 2 行， n 个正整数 t_1, t_2, \dots, t_n ， ($1 \leq t_i \leq 1000$) 表示每个人装满水桶的时间。

输出：

1 行，一个正整数，表示他们花费的最少总时间。

样例输入：

4 2

2 6 4 5

样例输出：

23

练习 3:

某国为了防御敌国的导弹袭击，发展出一种导弹拦截系统。但是这种导弹拦截系统有一个缺陷：虽然它的第一发炮弹能够到达任意的高度，但是以后每一发炮弹都不能高于前一发的高度。输入 n 个导弹依次飞来的高度（给出的高度数据是不大于 30000 的正整数），计算如果要拦截所有导弹最少要配备多少套这种导弹拦截系统。 比如：有 8 颗导弹，飞来的高度分别为

389 207 175 300 299 170 158 165

那么需要 2 个系统来拦截，他们能够拦截的导弹最优解分别是：

系统 1：拦截 389 207 175 170 158

系统 2：拦截 300 299 165

输入：

两行，第一行表示飞来导弹的数量 n ($n \leq 1000$)

第二行表示 n 颗依次飞来的导弹高度

输出：

要拦截所有导弹最小配备的系统数 k

样例输入：

8

389 207 175 300 299 170 158 165

样例输出：

2

5、课后练习

练习 1:

在漆黑的夜里， N 位旅行者来到了一座狭窄而且没有护栏的桥边。如果不借助手电筒的话，大家是无论如何也不敢过桥去的。不幸的是， N 个人一共只带了一只手电筒，而桥窄得只够让两个人同时过。如果各自单独过桥的话， N 人所需的时间已知；而如果两人同时过桥，所需要的时间就是走得比较慢的那个人单独行动时所需的时间。问题是，如何设计一个方案，让这 N 人尽快过桥，计算出这 N 个人的最短过桥时间。

输入：

第一行是一个整数 N ($1 \leq N \leq 1000$) 表示共有 N 个人要过河

第二行是 N 个整数 S_i , 表示这 N 个人过河所需要花时间。 ($0 < S_i \leq 100$)

输出：

所有人过河的最短时间

样例输入：

4

1 2 5 10

样例输出：

17

九、递推算法

1、递推介绍

递推：通过已知条件,利用特定关系得出中间推论,直至得到结果的算法。

(1) **顺推法**：从已知条件出发，逐步推算出要解决的方法。例如，我们在入门第十二讲斐波那契数列递推解法中就使用了顺推法。

(2) **逆推法**：从已知的结果出发，用迭代表达式逐步推算出问题开始的条件，即顺推法的逆过程。

(3) 无论是顺推还是逆推，其关键是找到**递推式**。

2、数塔问题

写一个程序来查找从最高点到底部任意处结束的路径，使路径经过数字的和最大。每一步可以走到左下方的点也可以到达右下方的点。

```
      7
     3  8
    8  1  0
   2  7  4  4
  4  5  2  6  5
```

在上面的样例中,从 $7 \rightarrow 3 \rightarrow 8 \rightarrow 7 \rightarrow 5$ 的路径产生了最大

输入格式

第一个行一个正整数 r ,表示行的数目。

后面每行为这个数字金字塔特定行包含的整数。

输出格式

单独的一行,包含那个可能得到的最大的和。

输入样例：

5


7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

输出样例：
30

分析：

(1) 顺推：如果从塔顶出发寻找最大路径是否能找到？向下走的时候每次选择最大值（贪心）？

(2) 逆推：如果从每个塔底出发向上更新最大路径呢？



30				
23	21			
20	13	10		
9	12	10	10	
4	5	2	6	5

使用逆推法后，数塔的最大路径值被更新为上图。

递推式为：

$$\text{num}[i][j] = \max(\text{num}[i+1][j], \text{num}[i+1][j+1]) + \text{num}[i][j]$$

3、课堂练习

练习 1:

在数塔问题的基础上要求最大路径的行走路线。

样例输入:

```
5
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
```

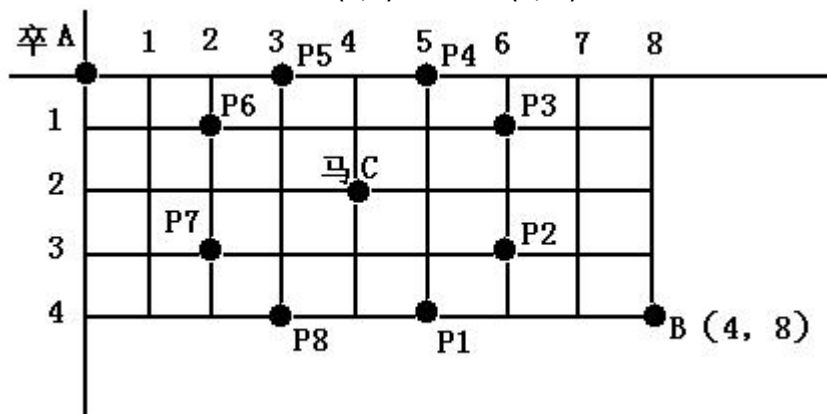
样例输出:

```
5, 2->4, 2->3, 1->2, 1->1, 1
30
```

练习 2:

棋盘上 A 点有一个过河卒，需要走到目标 B 点。卒行走的规则：可以向下、或者向右。同时在棋盘上 C 点有一个对方的马，该马所在的点和所有跳跃一步可达的点称为对方马的控制点。因此称之为“马拦过河卒”。

棋盘用坐标表示，A 点 (0,0)、B 点 (n,m)，同样马的位置坐标是需要给出的。



现在要求你计算出卒从 A 点能够到达 B 点的路径的条数，假设马的位置是固定不动的，并不是卒走一步马走一步。

输入格式

一行四个正整数，分别表示 B 点坐标和马的坐标。

输出格式

一个整数，表示所有的路径条数。

输入 #1

6 6 3 3

输出 #1

6

说明/提示

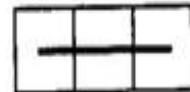
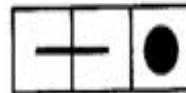
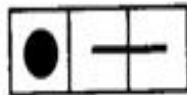
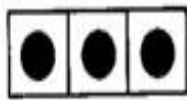
对于 100% 的数据， $1 \leq n, m \leq 20$ ， $0 \leq \text{马的坐标} \leq 20$ 。

4、课后练习

练习 1:

有 $1 \times n$ ($n \leq 50$) 的一个长方形，用一个 1×1 、 1×2 和 1×3 的骨牌铺满方格，请问有多少种铺法？

例如当 $n=3$ 时为 1×3 的方格。此时用 1×1 、 1×2 和 1×3 的骨牌铺满方格，共有四种铺法。如下图：



输入

一个整数 n ($n \leq 50$)

输出

骨牌的铺法

样例输入

3

样例输出

4

练习 2:

Pell 数列的前 10 项值分别是：1 2 5 12 29 70 169 408 985 2378，求该数列的第 n 项值 ($1 \leq n \leq 1000$)。

十、递归算法

1、递归介绍：

我们在入门第十二讲中已经简单介绍了递归算法、递归和递推的关系，在这一讲中，我们将进一步学习如何使用递归算法解决问题。

- 递归：函数中调用自身。
- 递归中的递：将问题拆解成子问题来解决，子问题再拆解成子子问题，...，直到被拆解的子问题无需再拆分成更细的子问题。
- 递归中的归：最小的子问题解决了，那么它的上一层子问题也就解决了，上一层的子问题解决了，上上层子问题自然也就解决了。



2、递归算法通用解决思路

- 明确函数功能
- 分析问题与子问题的关系，求递推式
- 递归边界

3、递归算法一般用于解决三类问题：

- 问题的定义是按递归定义的（Fibonacci 函数，阶乘，...）
- 问题的解法是递归的（有些问题只能使用递归方法来解决，例如，汉诺塔问题）
- 数据结构是递归的（树的遍历，树的深度，...）

4、课堂练习

练习 1：：

螺旋矩阵

输入：

一个整数 n

输出：

一个 n 行方阵

输入样例：

3

输出样例：

1 2 3

8 9 4

7 6 5

	1	2	3	4	5	6
1	1	2	3	4	5	6
2	20	21	22	23	24	7
3	19	32	33	34	25	8
4	18	31	36	35	26	9
5	17	30	29	28	27	10
6	16	15	14	13	12	11

练习 2：

在某个字符串（长度不超过 100）中有左括号、右括号和大小写字母。与常见的算数式一样，任何一个左括号都由内至外的与它右侧且距离最近的右括号匹配。写一个程序，找到无法匹配的左括号和右括号，输出原来的字符串，并在下一行

标出不能匹配的括号。不能匹配的左括号用“\$”标注，不能匹配的右括号用“?”标注。

输入：包括多组数据，每组数据一行，包含一个字符串，只包含左右括号和大小写字母，字符串长度不超过 100。

输出：对每组输出数据，输出有两行，第一行为原始字符串，第二行由\$和?组成，表示不能匹配的左括号和右括号

样例输入：

```
((ABCD(x)
)(rttyy())sss)(
```

样例输出：

```
((ABCD(x)
$$
)(rttyy())sss)(
?      ?$
```

5、课后练习

练习 1:

拐角

输入整数 N，输出相应方阵

输入：

一个整数 N ($0 < n < 10$)

输出：

数字组成的一个方阵

输入样例：

5

输出样例：

```
1 1 1 1 1
1 2 2 2 2
1 2 3 3 3
1 2 3 4 4
1 2 3 4 5
```

练习 2:

汉诺塔（又称河内塔）问题

三根柱子分别表示为 A，B，C。A 杆上有若干金片。

每次只能移动一个金片，小的只能放到大 的上方。

把所有金片由 A 杆挪到 C 杆上。

输入:

一个整数 N，表示 A 柱上有 N 个碟子。（ $0 < n \leq 20$ ）

输出:

若干行，依次输出每一步移动的情况

输入样例:

3

输出样例:

A -> C

A -> B

C -> B

A -> C

B -> A

B -> C

A -> C

十一、深度优先搜索

1、深度优先搜索的特点

- 深度优先搜索 DFS（Depth First Search）属于图算法的一种。
- 对每一个可能的分支路径深入到不能再深入为止，而且每个节点只能访问一次。
- 搜索一条路，直到不能走为止，换另一条路。

2、深搜常见题型

- 求最少步数
- 求第一条路径
- 求所有路径

3、回溯算法

- 回溯算法实际上一个类似枚举的搜索尝试过程。
- 在搜索尝试过程中寻找问题的解，当发现已不满足求解条件时，就“回溯”返回，尝试别的路径。
- 许多复杂的，规模较大的问题都可以使用回溯法，有“通用解题方法”的美称。

4、回溯与深搜的区别

```
1 void Dfs(cc) {  
2  
3     if(找到终点) {  
4         输出/停止  
5     }  
6     else {  
7         for(int i = 0; i < 4; i++) {  
8             curX = x + _changeX[i];  
9             curY = y + _changeY[i];  
10            if(满足条件) {  
11                保存当前状态  
12                Dfs(cc + 1); // 在当前状态下进行搜索  
13                回退到上一步的状态  
14            }  
15        }  
16    }  
17 }
```

5、回溯法算法的两个框架

```
1 int Dfs(int k) {  
2     for(int i = 1; i <= 算符种数; i++) {  
3         if(满足条件) {  
4             保存结果  
5             if(到达目的地) {  
6                 输出解  
7             }  
8             else {  
9                 Dfs(k + 1);  
10            }  
11            恢复状态 (回溯)  
12        }  
13    }  
14 }
```

```

1  int Dfs(int k) {
2      if(到达目的地) {
3          输出解
4      }
5      else {
6          for(int i = 1; i <= 算符种数; i++) {
7              if(满足条件) {
8                  保存结果
9                  Dfs(k + 1);
10                 恢复状态 (回溯)
11             }
12         }
13     }
14 }

```

6、课堂练习

练习 1:

Mike 同学在为扫地机器人设计一个在矩形区域中行走的算法，Mike 是这样设计的：先把机器人放在出发点(1,1)点上，机器人在每个点上都会沿用如下的规则来判断下一个该去的点是哪里。规则：优先向右，如果向右不能走（比如：右侧出了矩形或者右侧扫过了）则尝试向下，向下不能走则尝试向左，向左不能走则尝试向上；直到所有的点都扫过。

Mike 为了验证自己设计的算法是否正确，打算先模拟一下这个算法，每当机器人走过一个单元格时，会在单元格内标记一个数字，这个数字从 1 开始，每经过一个单元格数字会递增 1，直到所有的单元格都扫一遍，也就是所有的单元格都标记过数字，机器人会自动停止。

比如：如果机器人按照上面的规则，清扫一个 3 * 4 大小的矩形区域，那么标记数字的结果如下图所示。

	1	2	3	4
1	1	2	3	4
2	10	11	12	5
3	9	8	7	6

请你帮助 Mike 设计一个程序，按照上面的规则，将一个 $n * m$ 大小的矩形，标记一下数字，输出最终标记的结果。

输入：

一行内有 2 个两个整数 n 和 m ，用空格隔开，分别代表矩形区域的行数（高）和

列数（宽）（ n 和 m 都是 2~9 之间的整数）

输出：

输出按题意机器人走过每个点之后，标记数字的结果，每个数字输出时场宽设置为 3。

样例输入：

3 4

样例输出：

```
1  2  3  4
10 11 12  5
 9  8  7  6
```

练习 2：

一天 Extense 在森林里探险的时候不小心走入了一个迷宫，迷宫可以看成是由 $n * n$ 的格点组成，每个格点只有 2 种状态，0 和 1，前者表示可以通行后者表示不能通行。同时当 Extense 处在某个格点时，他只能移动到上下左右四个方向之一的相邻格点上，Extense 想要从点 A 走到点 B，问在不走出迷宫的情况下能不能办到。如果起点或者终点有一个不能通行(为 1)，则看成无法办到。

输入

第 1 行是一个正整数 n ($1 \leq n \leq 100$)，表示迷宫的规模是 $n * n$ 的。接下来是一个 $n * n$ 的矩阵，矩阵中的元素为 0 或者 1。再接下来一行是 4 个整数 $x1, y1, x2, y2$ ，描述 A 处在第 $x1$ 行 第 $y1$ 列，B 处在第 $x2$ 行 第 $y2$ 列。

输出

能办到则输出“YES”，否则输出“NO”。

输入样例：

```
3
0 1 1
0 0 1
1 0 0
1 1 3 3
```

输出样例：

YES

练习 3：

已知一 $N \times N$ 的迷宫，允许往上、下、左、右四个方向行走，现请你按照左、上、右、下顺序进行搜索，找出第一条从左上角到右下角的路径。

输入数据有若干行，第一行有一个自然数 N ($N \leq 20$)，表示迷宫的大小，其后

有 N 行数据，每行有 N 个 0 或 1（数字之间没有空格，0 表示可以通过，1 表示不能通过），用以描述迷宫地图。入口在左上角（1，1）处，出口在右下角（ N ， N ）处。所有迷宫保证存在从入口到出口的可行路径。

输出

输出数据仅一行，为按照要求的搜索顺序找到的从入口到出口的第一条路径（搜索顺序：左、上、右、下）。

输入样例：

```
4
0001
0010
0010
0000
```

输出样例：

```
(1,1)->(1,2)->(2,2)->(2,1)->(3,1)->(3,2)->(4,2)->(4,3)->(4,4)
```

练习 4：

从键盘读入一个整数 n （ $n \leq 6$ ），请输出 $1 \sim n$ 中所有整数的全排列，按照由小到大输出结果，每组的 n 个数之间用空格隔开。

全排列的含义：从 n 个不同元素中任取 m （ $m \leq n$ ）个元素，按照一定的顺序排列起来，叫做从 n 个不同元素中取出 m 个元素的一个排列。当 $m=n$ 时所有的排列情况叫全排列。

如当 $n=3$ 时，全排列的结果为：

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

输入

一个整数 n （ $n \geq 1 \ \&\& \ n \leq 6$ ）

输出

$1 \sim n$ 中所有数的全排列的结果，按照由小到大输出，每行 n 个数

7、课后练习

练习 1:

农夫约翰的农场可以表示成 $N \times M$ ($1 \leq N, M \leq 100$) 个方格组成的矩形。由于近日的降雨，在约翰农场上的不同地方形成了池塘。每一个方格或者有积水 ('W') 或者没有积水 ('.')。农夫约翰打算数出他的农场上共形成了多少池塘。一个池塘是一系列相连的有积水的方格，每一个方格周围的四个方格都被认为是与这个方格相连的。现给出约翰农场的图样，要求输出农场上的池塘数。

输入:

第 1 行: 由空格隔开的两个整数: N 和 M

第 2.. $N+1$ 行: 每行 M 个字符代表约翰农场的一排方格的状态。每个字符或者是 'W' 或者是 '.', 字符之间没有空格。

输出:

输出只有 1 行, 输出约翰农场上的池塘数

输入样例:


```
10 12
W.....WW.
.WWW.....WWW
....WW...WW.
.....WW.
.....W..
..W.....W..
.WW.....WW.
W.W.W.....W.
.WW.....W.
..W.....W.
```

输出样例:

```
13
```

练习 2:

在一张 $n \times m$ 的棋盘上 (如 6 行 7 列) 的最左上角 (1,1) 的位置有一个卒。该卒只能向下或者向右走, 且卒采取的策略是先向下, 请问从 (1,1) 点走到 (n,m) 点可以怎样走, 输出这些走法

	1	2	3	4	5	6	7
1							
2							
3							
4							
5							
6							

输入

两个整数 n, m 代表棋盘大小($3 \leq n \leq 8, 3 \leq m \leq 8$)

输出

卒的行走路线

输入样例：

3 3

输出样例：

1:1,1->2,1->3,1->3,2->3,3

2:1,1->2,1->2,2->3,2->3,3

3:1,1->2,1->2,2->2,3->3,3

4:1,1->1,2->2,2->3,2->3,3

5:1,1->1,2->2,2->2,3->3,3

6:1,1->1,2->1,3->2,3->3,3

十二、广度优先搜索

1、广度优先搜索的特点

- 广度优先搜索算法（Breadth First Search）BFS
- 从一个节点出发，先访问其直接相连的所有子节点，再访问其子节点相连的子节点
- 按层次顺序访问，直到访问到目标节点。

2、深搜和广搜的区别

- BFS 关注：解决最短或最少问题
- DFS 关注：解决路径能否到达或所有路径有效

3、线性表-队列

- 队列是一种操作受限制的线性表。
- 进行插入操作的端称为队尾。
- 进行删除操作的端称为队头。



4、广搜模板

```
while(_head <= _tail) {
    for(int k = 0; k < 4; k++) {
        curX = _changeX[k] + _que[_head][1];
        curY = _changeY[k] + _que[_head][2];
        if(_map[curX][curY] == -1) {
            tail++; // 队尾插入元素
            // 当前元素入队
        }
    }
    _head++; // 队首删除元素
}
}
```

5、课堂练习

练习 1:

Leyni 是一个地址调查员，有一天在他调查的地方突然出现个泉眼。由于当地的地势不均匀，有高有低，他觉得如果这个泉眼不断的向外溶出水来，这意味着这里在不久的将来将会一个小湖。水往低处流，凡是比泉眼地势低或者等的地方都会被水淹没，地势高的地方水不会越过。而且又因为泉水比较弱，当所有地势低的地方被淹没后，水位将不会上涨，一直定在跟泉眼一样的水位上。

由于 Leyni 已经调查过当地很久了，所以他手中有这里地势的详细数据。所有的地图都是一个矩形，并按照坐标系分成了一个个小方格，Leyni 知道每个方格的具体高度。我们假定当水留到地图边界时，不会留出地图外，现在他想通过这些数据分析出，将来这里将会出现一个多大面积的湖。

输入

有若干组数据，每组数据的第一行有四个整数 $n, m, p1, p2$ ($0 \leq 1000$)， n 和 m 表示当前地图的长和宽， $p1$ 和 $p2$ 表示当前地图的泉眼位置，即第 $p1$ 行第 $p2$ 列，随后的 n 行中，每行有 m 个数据。表示这每一个对应坐标的高度。

输出

输出对应地图中会有多少个格子被水充满。

输入样例：

3 5 2 3

3 4 1 5 1

2 3 3 4 7

4 1 4 1 1

输出样例：

6

练习 2:

有 $n*m$ 的迷宫，该迷宫有一个入口，一个出口。编写一程序打印一条从迷宫入口到出口的最短路径，黑色方块的单元表示走不通（用 1 表示），白色方块的内容表示走的通（用 0 表示）

只能往上下左右四个方向走，如果有最短路径，保证最短路径一定是唯一的，如果没有路径可以到达，则输出 “no way”。

输入

第一行输入 2 个整数 n 和 m (n 和 m 都是 10~150 之间的整数)，代表迷宫的行数和列数

接下来 n 行，每行有 m 个整数，1 代表不可走的点，0 代表可走的点

接下来一行，有 2 个整数 $s1$ 和 $s2$ 代表入口的坐标

接下来一行，有 2 个整数 $e1$ 和 $e2$ 代表出口的坐标，且入门和出口一定不同位置

输出：输出从入口到出口的最短路径，如果没有路径可达输出 “no way”

输入样例：

8 5

1 1 1 1 1

0 0 0 0 1

1 1 1 0 1

1 0 0 0 1

1 0 0 1 1

1 0 0 0 1

1 1 1 0 1

1 0 0 0 1

2 1

8 4

输出样例：

(2,1)->(2,2)->(2,3)->(2,4)->(3,4)->(4,4)->(4,3)->(5,3)->(6,3)->(6,4)->(7,4)->(8,4)

6、课后练习

练习 1:

John 用他的一头母牛和 Don 先生交换了一头“骑士牛”。这头牛有一个独特的能力——在牧场中能像中国象棋中的马一样跑跳（会中国象棋吗？不会？）。当

然，这头牛不能跳到岩石或树上，不过能跳到有牧草的地方。这儿有一个宽为 X ，高为 Y 的矩形牧场($1 \leq X \leq 150$; $1 \leq Y \leq 150$)。“骑士牛”和其它牛一样喜欢干草。给你一张包含“骑士牛”出发地和树、岩石、灌木或其它障碍物及大包干草等位置信息的地图，确定“骑士牛”得到干草最少要跳几“跳”。地图中“骑士牛”出发地用'K'表示；障碍物用'*'表示，牧草用'!'表示，干草所在地用'H'表示。

输入

第 1 行: 两个空格隔开的整数: X 和 Y

第 2.. $Y+1$ 行: 第 $Y-i+2$ 行包含 X 个没有空格的字符（就像上面的地图一样）：表示第 i 行的地图。

输出

第 1 行: 一个单独的整数表示最少的得到干草的“跳”数。所有的数据都能得到干草。

输入样例:

```
10 11
.....
...*....
.....
...*. *...
...*. *...
.....*..
..*..*...H
*.....
...*...*..
..K.....
...*...*
..*...*..
```

输出样例:

```
5
```

练习 2:

一个迷宫由 R 行 C 列格子组成，有的格子里有障碍物，不能走；有的格子是空地，可以走。

给定一个迷宫，求从左上角走到右下角最少需要走多少步(数据保证一定能走到)。只能在水平方向或垂直方向走，不能斜着走。

输入

第一行是两个整数， R 和 C ，代表迷宫的长和宽。（ $1 \leq R, C \leq 40$ ）

接下来是 R 行，每行 C 个字符，代表整个迷宫。空地格子用'.'表示，有障碍物的格子用'#'表示。迷宫左上角和右下角都是'.'。

输出

输出从左上角走到右下角至少要经过多少步（即至少要经过多少个空地格子）。计算步数要包括起点和终点

输入样例：

5 5

..###

#....

###.

###.

###.

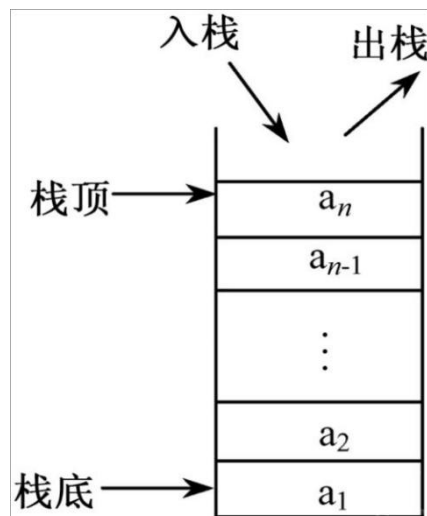
输出样例：

9

十三、栈

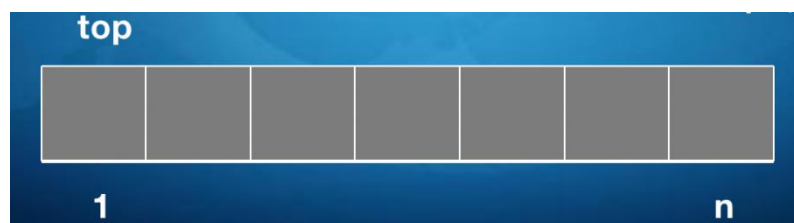
1、栈是什么

- 栈是只能在某一端插入和删除的特殊线性表。
- 栈的特点是 LIFO (Last In First Out, 后进先出), 栈也称为后进先出 (LIFO) 表。
- 栈是类似桶堆积物品的数据结构, 进行删除和插入的一端称为栈顶, 另一端称为底。
- 插入一般称为进栈 (PUSH), 删除则称为出栈 (POP)。



2、栈的操作

- 一个栈可以用定长为 $n+1$ 的数组来表示, 用一个栈指针 top 指向栈顶。栈指针在运算中永远指向栈顶。



- 若 $top=0$, 表示栈空。若 $top=n$, 表示栈满。
- 进栈时 $top+1$, 当 $top \geq n$ 时为上溢。
- 出栈时 $top-1$, 当 $top \leq 0$ 时为下溢。

- 进栈算法:
top<n 时:
top++;
s[top] = x;
- 出栈算法:
top > 1 时:
top--;

3、STL 容器适配器-stack

(1)、stack 在顺序容器的基础上实现，屏蔽了一些顺序容器的功能，并增加了一些特有的功能。其他的容器适配器还有：queue, priority queue。

(2)、容器适配器均具有以下三个成员函数：

- push:添加一个元素
- top:返回顶部或队首
- pop:删除一个元素

(3)、容器适配器没有迭代器，因此 STL 中的查找、排序等算法均无法使用

函数/方法	功能
push(value)	入栈
pop()	出栈
top()	返回栈顶元素
size()	返回元素的个数
empty()	判断栈是否为空

4、练习

练习 1:

假设一个表达式有英文字母（小写）、运算符（+，-，*，/）和左右小（圆）括号构成，以“@”作为表达式的结束符。请编写一个程序检查表达式中的左右圆括号是否匹配，若匹配，则返回“YES”；否则返回“NO”。表达式长度小于 255，左圆括号少于 20 个。

输入样例 1:

2*(x+y)/(1-x)@

输出样例 1:

YES

输入样例 2:

(25+x)*(a*(a+b+b)@

输出样例 2:

NO

练习 2:

输入一个由()[]四种符号构成的字符串。判断其中的括号是否匹配，是，就输出 yes，否则输出 no。

比如：输入“(())”、“([()])”、“([(([]))]”、“()[][]()[]”这几个字符串（双引号内部的内容），我们都算是匹配的。

再比如：输入“(())”、“([()])”、“([([()]))”这几个字符串，我们都认为是不匹配的。

输入

一个由()[]四种符号构成的字符串

输出

如果匹配，请输出 yes，如果不匹配，请输出 no

输入样例:

([])

输出样例:

yes

练习 3:

小明在你的帮助下，破密了 Ferrari 设的密码门，正要往前走，突然又出现了一个密码门，门上有一个算式，其中只有“(", ")", "0-9", "+", "-", "*", "/", "^”求出的值就是密码。小明数学学得不好，还需你帮他的忙。("/"用整数除法)

100%的数据满足：算式长度 ≤ 30 其中所有数据在 $2^{31}-1$ 的范围内。

输入

输入文件 calc.in 共 1 行，为一个算式。

输出

输出文件 calc.out 共 1 行，就是密码。

输入样例

$1+(3+2)*(7^2+6*9)/(2)$

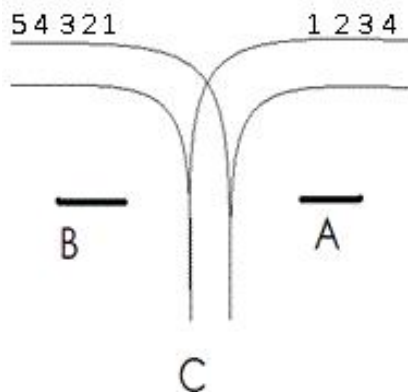
输出样例

258

练习 4:

有一个火车站，铁路如图所示，每辆火车从 A 驶入，再从 B 方向驶出，同时它的车厢可以重新组合。假设从 A 方向驶来的火车有 n 节 ($n \leq 1000$)，分别按照顺序编号为 1, 2, 3, ..., n 。假定在进入车站前，每节车厢之间都不是连着的，并且它们可以自行移动到 B 处的铁轨上。另外假定车站 C 可以停放任意多节车厢。但是一旦进入车站 C，它就不能再回到 A 方向的铁轨上了，并且一旦当它进入 B 方向的铁轨，它就不能再回到车站 C。

负责车厢调度的工作人员需要知道能否使它以 a_1, a_2, \dots, a_n 的顺序从 B 方向驶出，请来判断能否得到指定的车厢顺序。



输入

第一行为一个整数 n ，其中 $n \leq 1000$ ，表示有 n 节车厢，第二行为 n 个数字，表示指定的车厢顺序。

输出

如果可以得到指定的车厢顺序，则输出一个字符串“YES”，否则输出“NO”（注意要大写，不包含引号）。

输入样例

5

5 4 3 2 1

输出样例

YES