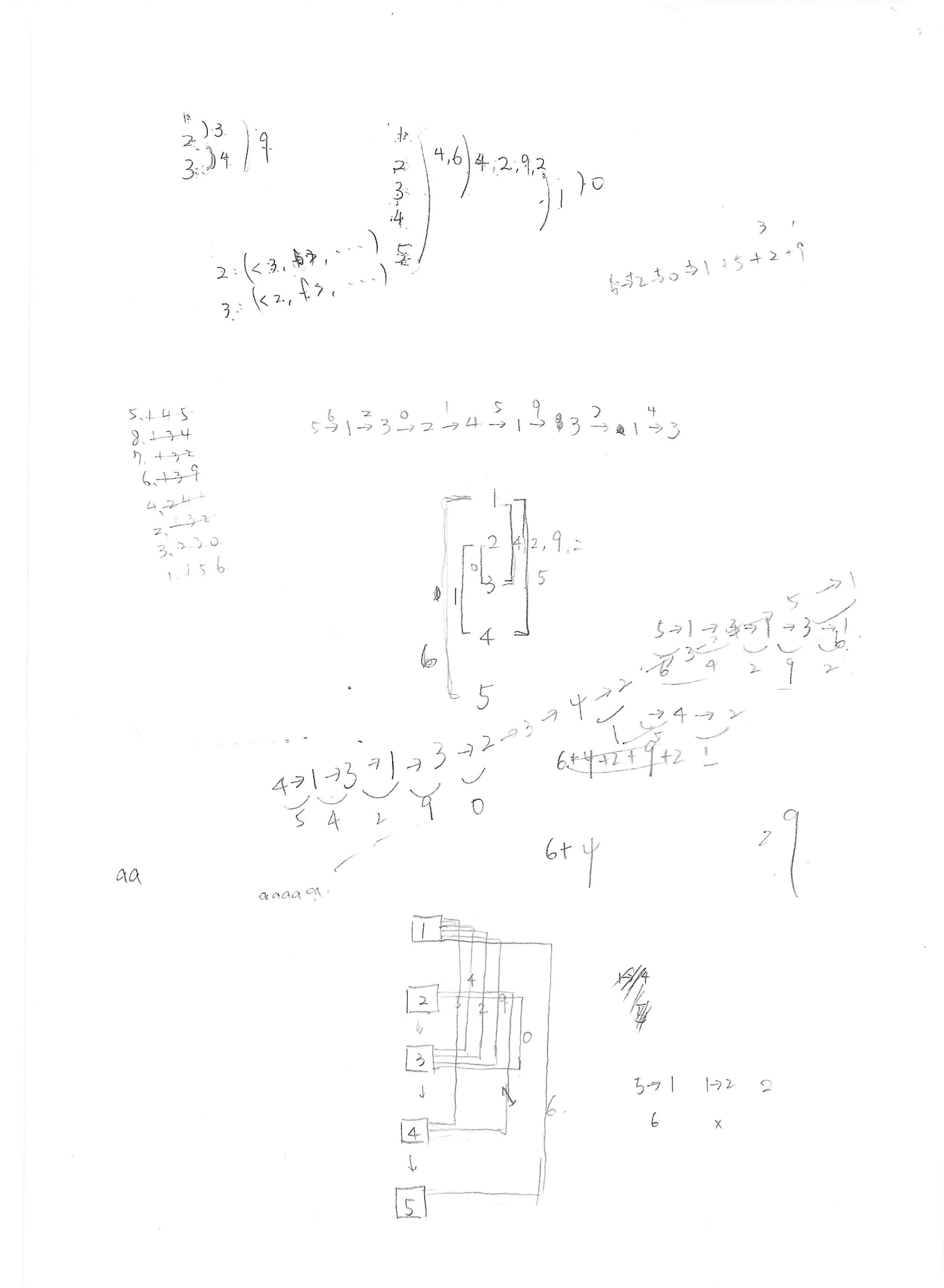
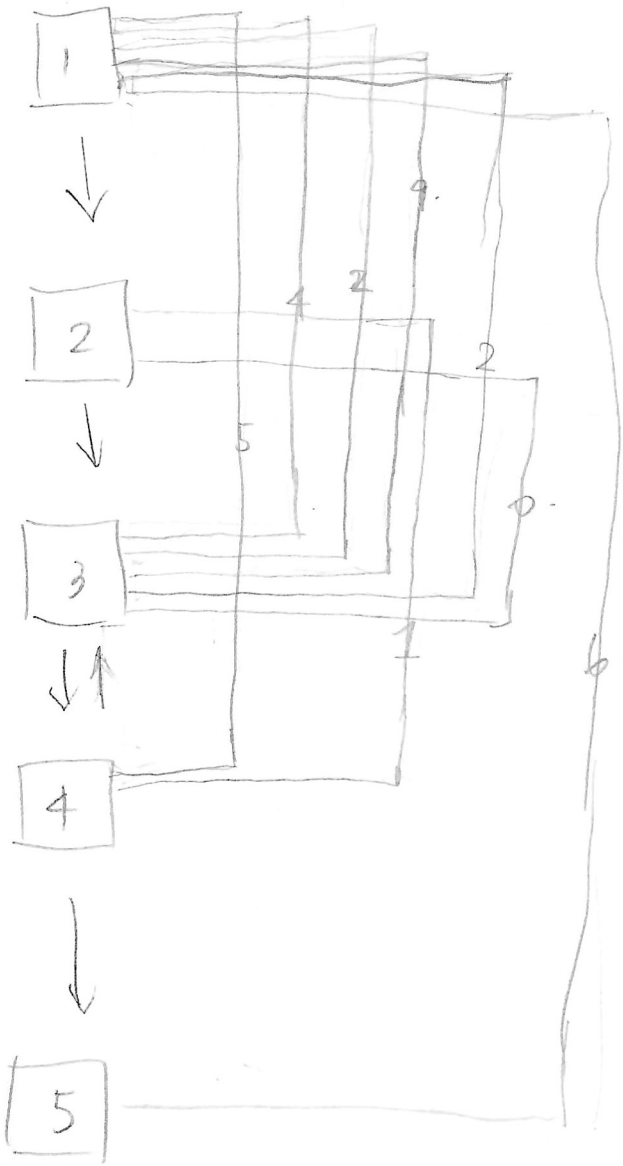


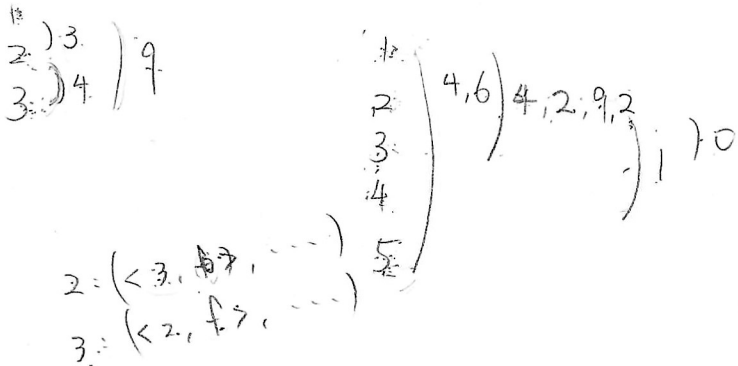


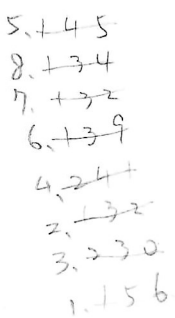
$5+4+2+9+2=16$

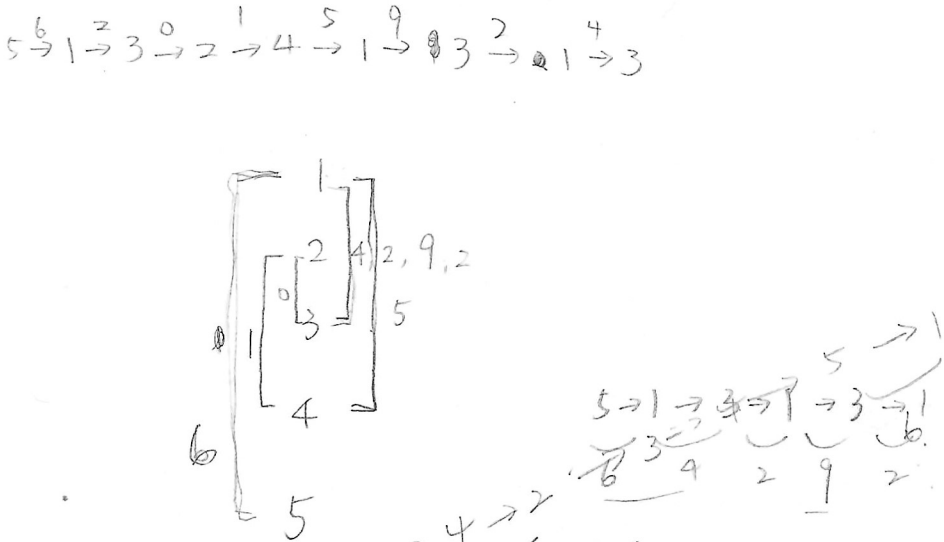
$10+9+9=28$

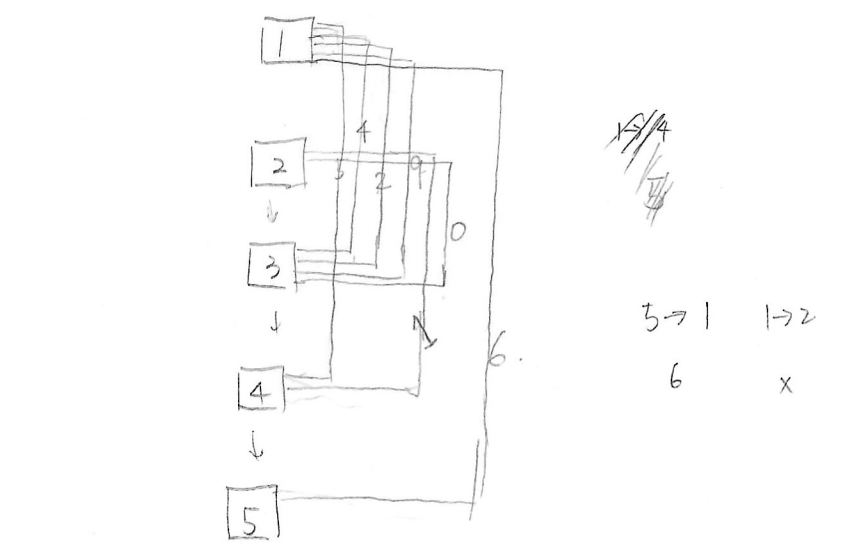
19

$0 \to 1$









```
list_b = list(words)
list_b.sort(reverse=True) # 降序排序
print(list_b) # ['kiwi', 'cherry', 'banana', 'apple']

list_c = list(words)
list_c.sort(key=len) # 按字符串長度升序排序
print(list_c) # ['kiwi', 'apple', 'banana', 'cherry']

list_d = list(data_tuples)
list_d.sort(key=lambda item: item[1]) # 按元組的第二個元素升序排序
print(list_d) # [(20, 2), (15, 2), (10, 5), (5, 8)]

# sorted() 函數 (返回一個新的排序列表，不修改原列表)
original_list = [3, 1, 4, 1]
new_sorted_list = sorted(original_list) # 升序排序
print(new_sorted_list) # [1, 1, 3, 4]
print(original_list) # [3, 1, 4, 1] (原列表不變)

new_sorted_desc = sorted(words, reverse=True) # 降序排序
print(new_sorted_desc) # ['kiwi', 'cherry', 'banana', 'apple']
```

9.8 四捨五入

```
import math
def myRound(x, d):
    mul = 1
    for i in range(d): mul *= 10
    if x >= 0: return math.floor(x * mul + 0.5) / mul
    else: return math.ceil(x * mul - 0.5) / mul
```

9.9 math

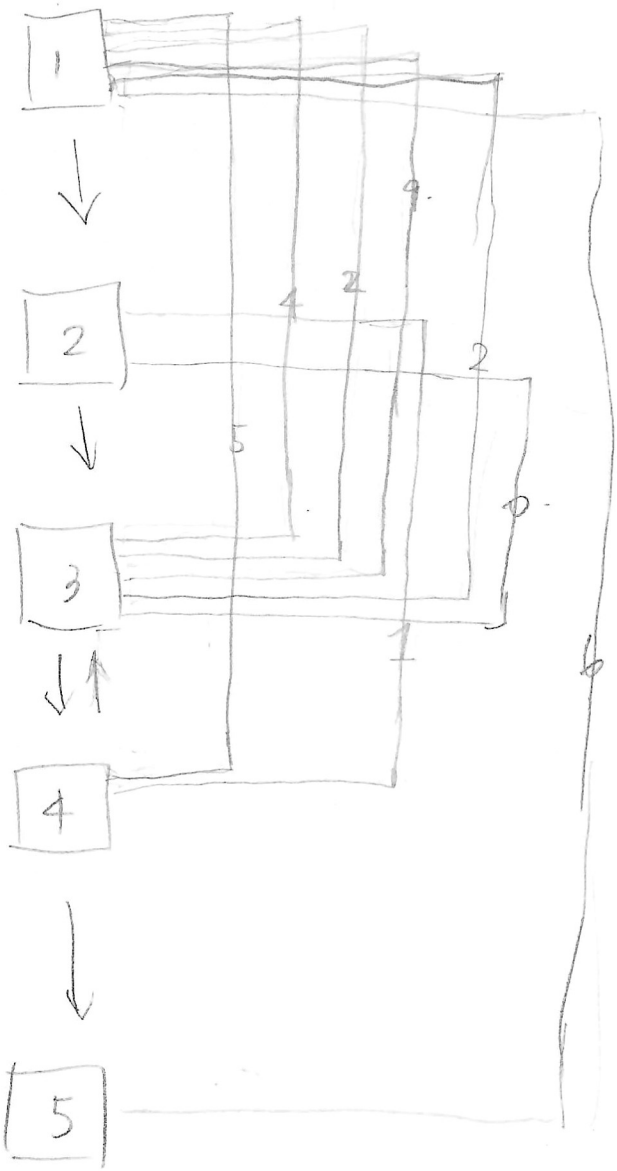
```
import math
b = 49
print(math.sqrt(b)) # 7.0 (計算平方根)
print(math.pi) # 3.141592653589793 (圓周率常數)
print(math.sin(math.pi/2)) # sin90 度= 1.0 (計算正弦值，參數為弧度)
print(math.factorial(5)) # 階乘 5! = 120 (計算階乘)
print(math.gcd(56, 42)) # 最大公因數 14 (計算最大公因數)
math.comb(n, k) # n! / (k! * (n - k)!) (計算組合數)
math.fmod(x, y) # 在實用浮點數運算通常是首選的取餘數方法
math.log(x, base) # 計算 x 底 base 為底的對數，預設 base 為 e (自然對數)
math.ceil(x) # 傳回大於或等於 x 的最小整數 (roundup)
math.floor(x) # 傳回小於或等於 x 的最大整數 (rounddown)
math.pow(x, y) # 計算 x 的 y 次方 (x**y)
math.degrees(x) # 將弧度轉換為角度
math.radians(x) # 將角度轉換為弧度
math.dist(p, q) # 計算點 p 和 q 之間的歐氏距離 (p 和 q 為座標序列)
```

9.10 output formatting

```
int_val = 42
float_val = 3.14159265
another_float_val = 123.456789
str_val = "Hello TOPC"

# 格式化整數
print("整數: %d" % int_val) # 整數: 42

# 格式化浮點數 (預設顯示6位小數，會四捨五入)
print("浮點數: %f" % float_val) # 浮點數: 3.141593

# 格式化浮點數並指定小數位數 (%.nf 表示保留 n 位小數，會四捨五入)
print("保留2位小數: %.2f" % another_float_val) # 保留2位小數: 123.46
print("保留4位小數: %.4f" % float_val) # 保留4位小數: 3.1416

# 格式化字符串
print("字符串: %s" % str_val) # 字符串: Hello TOPC

# 同時格式化多個值 (將所有要格式化的變數放入一個元組)
name = "Alice"
age = 30
height = 1.75
print("姓名: %s, 年齡: %d, 身高: %.2f米" % (name, age, height))
      # 姓名: Alice, 年齡: 30, 身高: 1.75米
```

9.11 structure list

```
# --- 建立列表 ---
# 列表是有序、可變、可包含重複元素的序列。

empty_list = [] # 建立一個空列表
print(f"空列表: {empty_list}") # []

# 直接定義列表元素
my_list = [1, 2, 3, 'apple', 'banana', True]
print(f"一般列表: {my_list}") # [1, 2, 3, 'apple', 'banana', True]

# 使用 list() 函數從其他可迭代對象建立
str_to_list = list("hello")
print(f"從字符串建立: {str_to_list}") # ['h', 'e', 'l', 'l', 'o']

# 列表推導式 (List Comprehension) - 簡潔建立或轉換列表
squares = [i**2 for i in range(5)] # 建立 0 到 4 的平方
print(f"列表推導式範例: {squares}") # [0, 1, 4, 9, 16]

# --- 訪問列表元素 (索引) ---
# 索引從 0 開始
fruits = ['apple', 'banana', 'cherry', 'date']
print(f"第一個元素: {fruits[0]}") # apple
print(f"第三個元素: {fruits[2]}") # cherry

# 負數索引 (從列表末尾開始計算，-1 表示最後一個元素)
print(f"最後一個元素: {fruits[-1]}") # date
print(f"倒數第二個元素: {fruits[-2]}") # cherry

# --- 列表切片 (Slicing) ---
# [start:end:step]
# start (包含), end (不包含), step (步長，默認為 1)
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(f"從索引 2 到 5 (不含): {numbers[2:5]}") # [2, 3, 4]
print(f"從開頭到索引 3 (不含): {numbers[:3]}") # [0, 1, 2]
print(f"從索引 5 到結尾: {numbers[5:]}") # [5, 6, 7, 8, 9]
print(f"整個複製列表: {numbers[:]}") # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(f"每隔一個取值: {numbers[::2]}") # [0, 2, 4, 6, 8]
print(f"反轉列表: {numbers[::-1]}") # [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

# --- 修改列表元素 ---
my_nums = [10, 20, 30, 40]
my_nums[1] = 25 # 修改索引 1 的元素
print(f"修改元素後: {my_nums}") # [10, 25, 30, 40]

my_nums[2:4] = [35, 45, 55] # 修改切片範圍內的元素 (可以改變列表長度)
print(f"修改切片後: {my_nums}") # [10, 25, 35, 45, 55]

# --- 增加元素 ---
add_list = [1, 2]
add_list.append(3) # 在列表末尾添加一個元素
print(f"append 後: {add_list}") # [1, 2, 3]

add_list.extend([4, 5]) # 在列表末尾添加多個元素 (使用另一個可迭代對象)
print(f"extend 後: {add_list}") # [1, 2, 3, 4, 5]

add_list.insert(1, 1.5) # 在指定索引位置插入一個元素
print(f"insert 後: {add_list}") # [1, 1.5, 2, 3, 4, 5]

# --- 刪除元素 ---
remove_list = ['a', 'b', 'c', 'd', 'e']
removed_item = remove_list.pop() # 移除並返回列表末尾的元素
print(f"pop 末尾元素: {removed_item}, 列表: {remove_list}") # e, ['a', 'b', 'c', 'd']

removed_at_index = remove_list.pop(1) # 移除並返回指定索引的元素
print(f"pop 索引 1 元素: {removed_at_index}, 列表: {remove_list}") # b, ['a', 'c', 'd']

remove_list.remove('c') # 移除列表中第一個匹配的元素 (若不存在會報 ValueError)
print(f"remove 'c' 後: {remove_list}") # ['a', 'd']

del remove_list[0] # 刪除指定索引的元素
print(f"del 索引 0 後: {remove_list}") # ['d']

del remove_list[0:0] # 刪除切片範圍內的元素 (若範圍為空則不刪除)
```



5+4+2+9+2=16

10+9+9=28

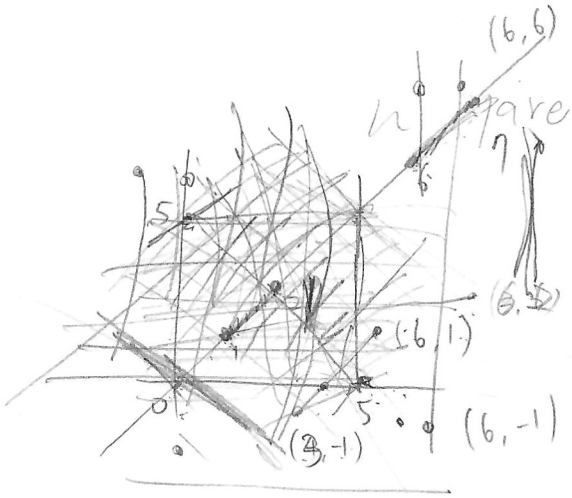
18

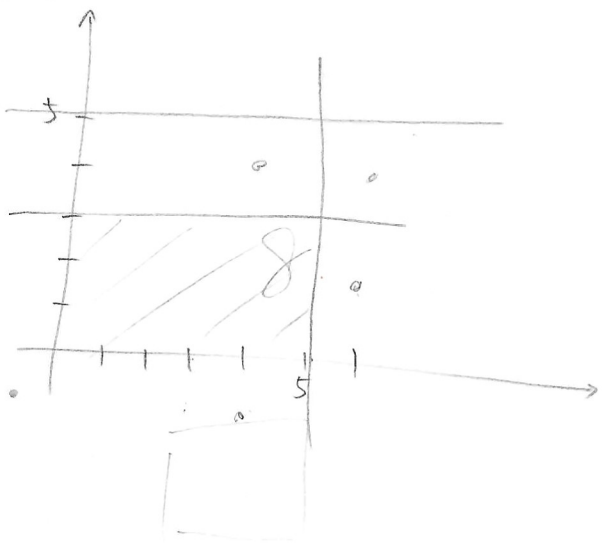
0 → 1

ACM ICPC Judge Test - woowoowoo

C++ Resource Test

```cpp
#include <bits/stdc++.h>
using namespace std;

namespace system_test {

const size_t KB = 1024;
const size_t MB = KB * 1024;
const size_t GB = MB * 1024;

size_t block_size, bound;
void stack_size_dfs(size_t depth = 1) {
  if (depth >= bound)
    return;
  int8_t ptr[block_size]; // 若無法編譯將 block_size 改成常數
  memset(ptr, 'a', block_size);
  cout << depth << endl;
  stack_size_dfs(depth + 1);
}

void stack_size_and_runtime_error(size_t block_size, size_t
    bound = 1024) {
  system_test::block_size = block_size;
  system_test::bound = bound;
  stack_size_dfs();
}

double speed(int iter_num) {
  const int block_size = 1024;
  volatile int A[block_size];
  auto begin = chrono::high_resolution_clock::now();
  while (iter_num--)
    for (int j = 0; j < block_size; ++j)
      A[j] += j;
  auto end = chrono::high_resolution_clock::now();
  chrono::duration<double> diff = end - begin;
  return diff.count();
}

void runtime_error_1() {
  // Segmentation fault
  int *ptr = nullptr;
  *(ptr + 7122) = 7122;
}

void runtime_error_2() {
  // Segmentation fault
  int *ptr = (int *)memset;
  *ptr = 7122;
}

void runtime_error_3() {
  // munmap_chunk(): invalid pointer
  int *ptr = (int *)memset;
  delete ptr;
}

void runtime_error_4() {
  // free(): invalid pointer
  int *ptr = new int[7122];
  ptr += 1;
  delete[] ptr;
}

void runtime_error_5() {
  // maybe illegal instruction
  int a = 7122, b = 0;
  cout << (a / b) << endl;
}

void runtime_error_6() {
  // floating point exception
  volatile int a = 7122, b = 0;
  cout << (a / b) << endl;
}

void runtime_error_7() {
  // call to abort.
  assert(false);
}

} // namespace system_test

#include <sys/resource.h>
void print_stack_limit() { // only work in Linux
  struct rlimit l;
  getrlimit(RLIMIT_STACK, &l);
  cout << "stack_size = " << l.rlim_cur << " byte" << endl;
}
```

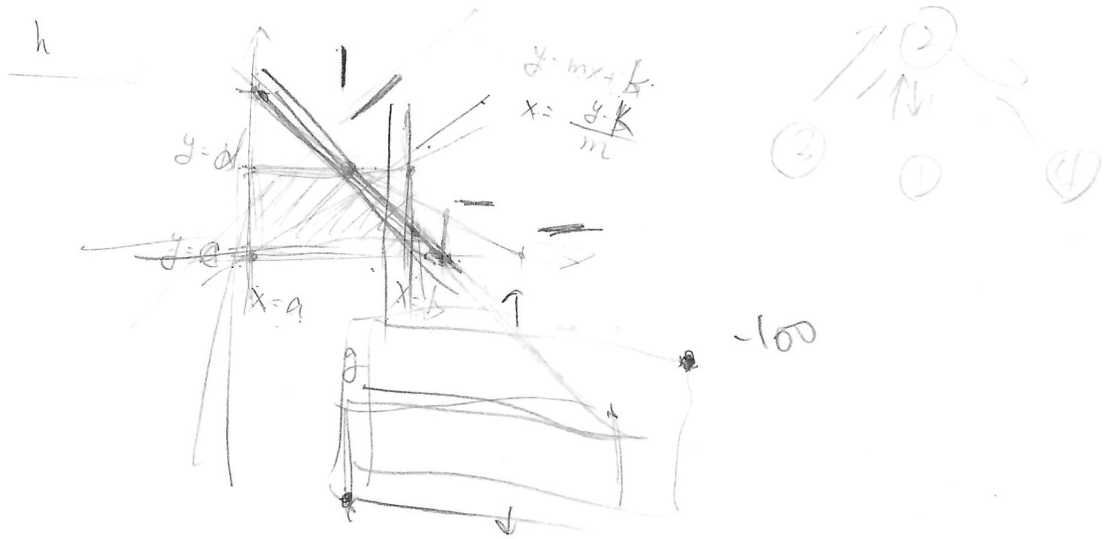




	8		1	2
	4	3		7
		5	6	

$y = +1x + 10$

$y = x$



```
print(f"del 空切片後: {remove_list}") # ['d']

# del remove_list # 刪除整個列表變數
# remove_list.clear() # 清空列表所有元素

# --- 遍歷列表 ---
iterate_list = ['one', 'two', 'three']

print("\n遍歷列表 (for 迴圈):")
for item in iterate_list:
    print(item) # one, two, three

print("遍歷列表 (for 迴圈，帶索引):")
for i, item in enumerate(iterate_list):
    print(f"索引: {i}, 元素: {item}") # 索引: 0, 元素: one / 索
        引: 1, 元素: two / ...

# --- 其他常用列表操作 ---
check_list = [1, 2, 3, 2, 4]

# 列表長度
print(f"列表長度: {len(check_list)}") # 5

# 檢查元素是否存在
print(f"3 在列表中嗎? {3 in check_list}") # True
print(f"5 不在列表中嗎? {5 not in check_list}") # True

# 元素計數
print(f"元素 2 出現次數: {check_list.count(2)}") # 2

# 查找元素索引 (返回第一個匹配的索引，若不存在會報 ValueError)
print(f"元素 4 的索引: {check_list.index(4)}") # 4

# 列表排序 (請參考 sorting.py 檔案)
# check_list.sort() # 就地排序
# sorted_list = sorted(check_list) # 返回新列表

# 反轉列表
check_list.reverse() # 就地反轉列表
print(f"反轉後: {check_list}") # [4, 2, 3, 2, 1]

# 複製列表
# copy_of_list = check_list[:] # 切片方式複製
# copy_of_list = list(check_list) # list() 函數複製
# copy_of_list = check_list.copy() # .copy() 方法複製
```

9.12 structure dictionary

```
# --- 建立字典 ---
# 字典是由鍵值對的無序集合，鍵必須是唯一的且不可變 (如數字、字符
    串、元組)。

empty_dict = {} # 建立一個空字典
print(f"空字典: {empty_dict}") # {}

# 直接指定鍵值對初始化
my_dict = {'apple': 1, 'banana': 2, 'cherry': 3}
print(f"初始字典: {my_dict}") # {'apple': 1, 'banana': 2, '
    cherry': 3}

# 從包含鍵值對元組的列表建立字典
dict_from_pairs = dict([('a', 10), ('b', 20)])
print(f"從鍵值對列表建立: {dict_from_pairs}") # {'a': 10, 'b':
    20}

# --- 訪問字典元素 ---
# 1. 直接使用鍵訪問 (若鍵不存在會報 KeyError)
print(f"apple 的值: {my_dict['apple']}") # 1

# 2. 使用 .get() 訪問 (若鍵不存在返回 None 或指定預設值)
print(f"banana 的值 (get): {my_dict.get('banana')}") # 2
print(f"grape 的值 (get, 不存在): {my_dict.get('grape')}") #
    None
print(f"grape 的值 (get, 預設為 0): {my_dict.get('grape', 0)}")
    # 0

# --- 新增與修改元素 ---
my_dict['date'] = 4 # 新增一個鍵值對
print(f"新增 'date': {my_dict}") # {'apple': 1, 'banana': 2, '
    cherry': 3, 'date': 4}

my_dict['apple'] = 10 # 修改 'apple' 的值
print(f"修改 'apple': {my_dict}") # {'apple': 10, 'banana': 2,
    'cherry': 3, 'date': 4}

# --- 移除元素 ---
del my_dict['cherry'] # 刪除指定鍵值對 (若鍵不存在會報 KeyError
    )
print(f"刪除 'cherry' 後: {my_dict}") # {'apple': 10, 'banana':
    2, 'date': 4}

popped_value = my_dict.pop('date') # 移除並返回指定鍵的值 (若鍵
    不存在會報 KeyError)
print(f"pop 'date' 的值: {popped_value}, 字典: {my_dict}") # 4,
    {'apple': 10, 'banana': 2}

# .pop() 也可以帶預設值，避免 KeyError
popped_non_existent = my_dict.pop('grape', -1) # 如果 'grape'
    不存在，返回 -1
print(f"pop 'grape' (不存在): {popped_non_existent}, 字典: {
    my_dict}") # -1, {'apple': 10, 'banana': 2}

my_dict.clear() # 清空所有元素
print(f"清空後: {my_dict}") # {}

# --- 遍歷字典 ---
iter_dict = {'a': 1, 'b': 2, 'c': 3}

print("\n遍歷鍵 (默認):")
for key in iter_dict:
    print(key) # a, b, c

print("遍歷值 (.values()):")
for value in iter_dict.values():
    print(value) # 1, 2, 3

print("遍歷鍵值對 (.items()):")
for key, value in iter_dict.items():
    print(f"鍵: {key}, 值: {value}") # 鍵: a, 值: 1 / 鍵: b, 值
        : 2 / 鍵: c, 值: 3

# --- 其他常見操作 ---
check_dict = {'x': 10, 'y': 20, 'z': 30}

# 檢查鍵是否存在
print(f"'x' 在字典中嗎? {'x' in check_dict}") # True
print(f"'w' 在字典中嗎? {'w' in check_dict}") # False

# 取得字典長度 (鍵的數量)
print(f"字典長度: {len(check_dict)}") # 3

# 複製字典 (淺複製)
copied_dict = check_dict.copy()
print(f"複製的字典: {copied_dict}") # {'x': 10, 'y': 20, 'z':
    30}
```

9.13 math functions

```
import math # 導入 math 模組

# 對數運算
num_log = 100
print(math.log(num_log)) # 以自然常數e為底的對數 (約 4.605)
print(math.log10(num_log)) # 以10為底的對數 (2.0)
print(math.log(num_log, 2)) # 以2為底的對數 (約 6.643)

# 大數對數 (Python 的整數支持任意精度)
large_number = 10**500000 # 產生一個非常大的整數 (1後面有50萬個
    0)
print(math.log(large_number, 6)) # 計算這個大數以6為底的對數 (
    約 [illegible])

# 乘方運算
print(2 ** 10) # 2的10次方 (1024, 返回整數)
print(math.pow(2, 10)) # 2的10次方 (1024.0, 返回浮點數)
```

10 Shortest-Path

10.1 Floyd-Warshall

```
// Floyd-Warshall Algorithm
// 多點最短路徑 時間O(n^3) 空間O(n^2)
// 邏輯:a to c加個中間點b 如果比較短就改a to b to c
// path為距離 to為移動路徑(都是二維vector) to[i][j] = j  cin >>
    path[i][j]
FOR(mid,0,n){
    FOR(start,0,n){
        FOR(end,0,n){
```

a b c d.

e f g h

0 0 5 5

$x = a$.

$x = c$

$y = b$

$y = d$.

$y = mx + k$.

$f = em + k$

$-) \; h = gm + k$

$=$

$$y = \frac{h-f}{g-e}x + k$$

(e, f)

(g, h).

$$\frac{gh-eh}{g-e} = \frac{gh-gf}{g-e} + k$$

$$y = \frac{h-f}{g-e} + \left(\frac{gf-he}{g-e}\right)$$

$g-e$

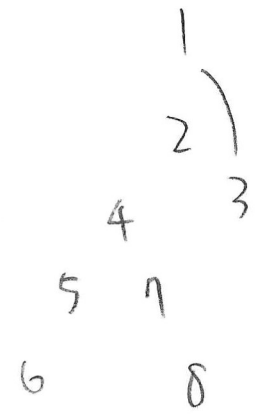
$$h = \frac{h-f}{g-e}g + k$$

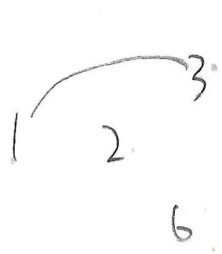
$$\frac{hg-he}{g-e} - \left(\frac{gh-gf}{g-e}\right) = k$$

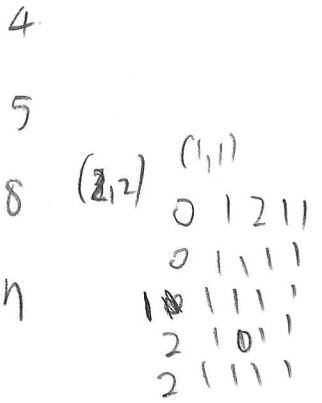
$$\frac{gf-he}{g-e} = k$$

ACM ICPC Team Reference - woowoowoo

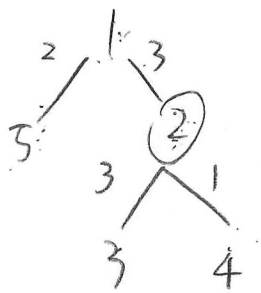
Contents







adcde



palindrome

eatind atindrome eplaindrome

lindrome

indrome

hcaab

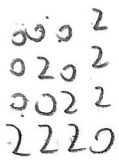
bcdeeh

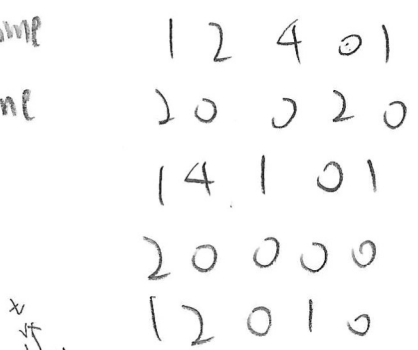
bcddcbo

① change

② delete

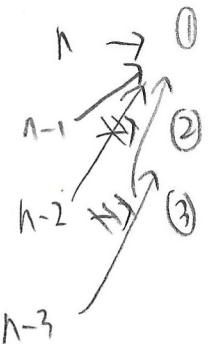
③ add



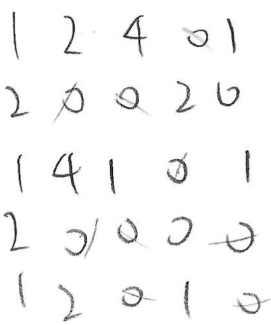


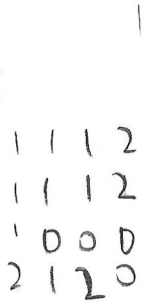
dc

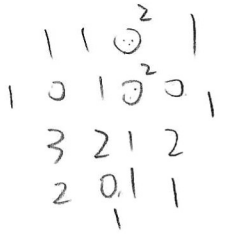
ahkcdeeabba



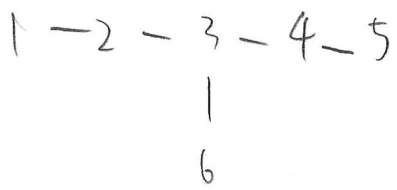
250000







wally walter



```
8               if(path[start][mid] + path[mid][end] < path[start][
                    end]){
9                   path[start][end] = path[start][mid] + path[mid
                        ][end];
10                  to[start][end] = to[start][mid];
11              }
12          }
13      }
14  }
```

Problem A
In-n-Out

Number of Test Cases: 100
Excution Time Limit: 1 second

The Autumn Festival has started again, and those who stand inside the white zone can participate in the Festival Lottery and win a chance to stand on the podium!

The T-shaped white zone consists of a horizontal strip and a vertical half-strip. As a staff of this event, you're given the coordinates of the strips, and your task is to demonstrate a point inside the T-shaped white zone and another point outside the zone.

Note that, points on the boundary of the strips are considered inside the white zone.

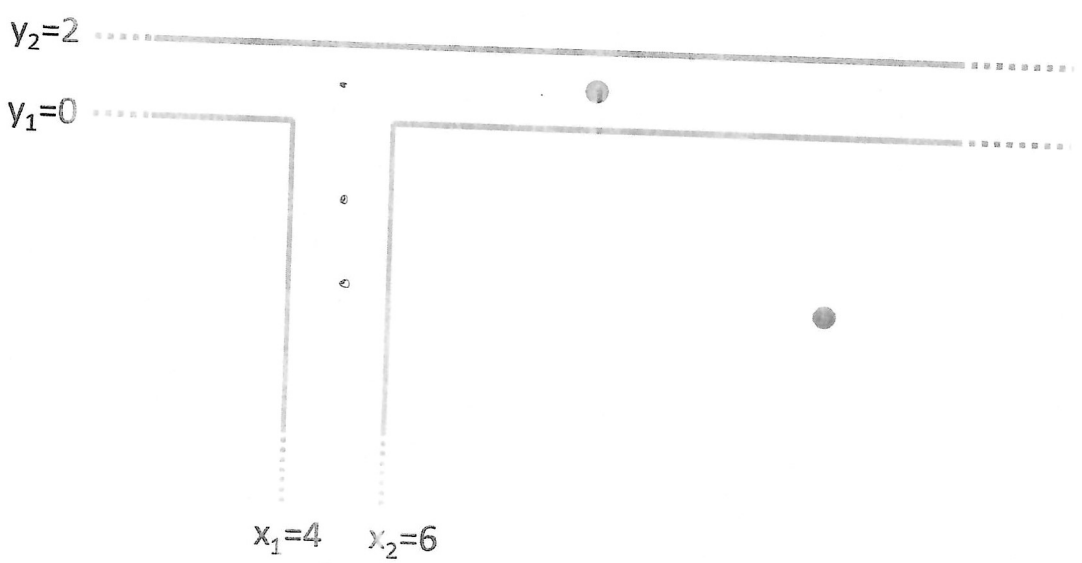


Figure 1: The above figure shows an example with a hozontal strip that spans from $y_1 = 0$ to $y_2 = 2$ and a vertical half-strip that anchors at $y = 0$ and spans from $x_1 = 4$ to $x_2 = 6$. This figures demonstrate two points. The point at $(7, 1)$ is inside the T-shaped white zone, and the point $(8, -6)$ is outside.

Input Format

There will be multiple test cases in the input file, one at a line.

Each test case consists of four integers y_1, y_2, x_1, x_2, which are the y-coordinates of the lower-boundary and the upper-boundary of the horizontal strip and the x-coordinates of the vertical strip, respectively.

A test case with $y_1 = y_2 = x_1 = x_2 = 0$ indicates the end of the input.

You may additionally assume the following.

- $y_1 \leq y_2$, $x_1 \leq x_2$.
- The absolute value of all coordinates does not exceed 10^6.

Output Format

For each test case, print four integers x_i, y_i, x_o, y_o, separated by spaces, where (x_i, y_i) is a point inside the zone and (x_o, y_o) is a point outside the zone.

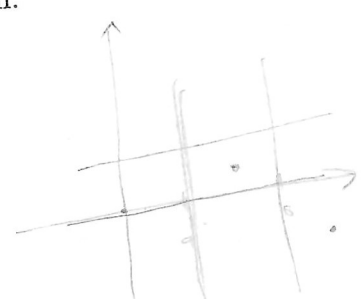
Your output has to satisfy the following constraint that

- $-10^9 \leq x_i, y_i, x_o, y_o \leq 10^9$.

If there are multiple solutions, print any of them.

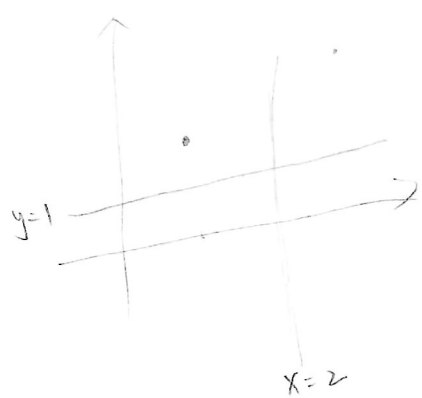
Sample Input

```
0 2 4 6
1 1 2 2
0 0 0 0
```



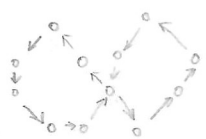
Output for the Sample Input

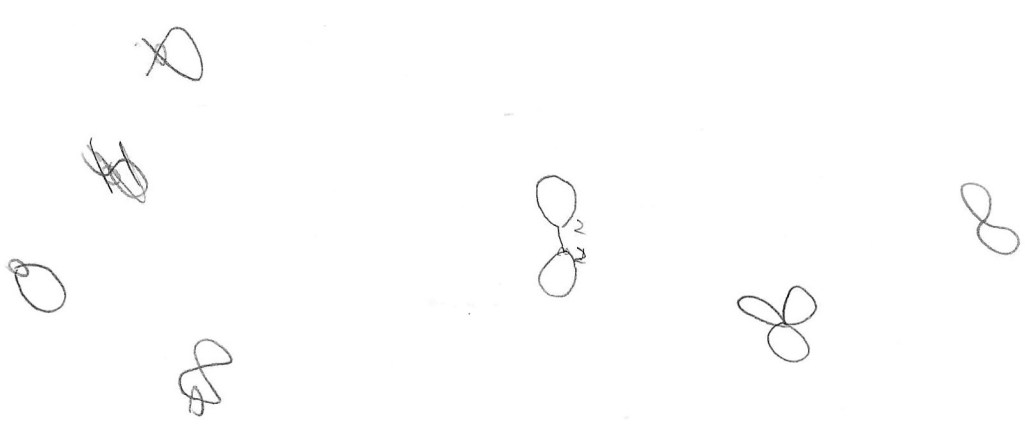
```
7 1 8 -6
1 2 3 4
```



44678 12|8 22 4660

4 3 2
5 < * 1
6 7 8





Problem E
Solving Linear Systems of Equations

Number of Test Cases: 10
Execution Time Limit: 1 second

Write a program to solve a system of n linear equations of n variables $x_1, x_2, \ldots, x_n$:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

All coefficients a_{ij} and b_i ($1 \le i, j \le n$) are integers. The solutions x_i are rational numbers, and the solution should not be expressed in real numbers. For example, $x_i = 1/7$ should not be written as $x_i = 0.5$.

Input Format

The input data consists of multiple test cases.

Each test case starts with a positive integer n ($1 \le n \le 50$) on a line by itself. This is the number of unknowns and n equations in this test case.

Following the number n are the n equations. Each of the equations contains $n+1$ rational numbers:
$a_{i1}, a_{i2}, \ldots, a_{in}, b_i$ ($i = 1, 2, \ldots, n$).
These numbers are presented in the following form:
$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n = b_i$ ($i = 1, 2, \ldots, n$).
Every rational number is represented as an integer p or a fraction p/q.

The last test case is followed by a 0, indicating the end of all test cases.

Output Format

The output for each system of equations consists of two parts. The first part is the system of equations itself, and the second part is the solution.

If the answer is an integer, it must be printed as an integer; otherwise, it must be printed as a simplified fraction of the form p/q. See the sample output for details.

If the system has no solution, print: "no solutions"; and if the system has infinitely many solutions, print: "infinite many solutions".

x, y coordinates of the 4 red candles. The second line contains 8 integers, which are x, y coordinates of the 4 green candles. The coordinates are integers between 0 and 9, $0 \leq x, y \leq 9$. Obviously, no two candles will be placed at the same position.

Output Format

Output "G" if there is a way for girls to draw the first line to win the game. If not, output "B", if boys can always win the game no matter how the girls draw the first line.

Sample Input

```
3
0 0 7 0 0 5 7 5
6 3 9 3 6 8 9 8
0 0 7 0 0 5 7 5
3 1 9 1 3 8 9 8
0 0 0 2 2 0 2 2
4 4 4 6 6 4 6 6
```

Output for the Sample Input

```
B
G
B
```

Problem A
Cake Game

Max no. of test cases: 10
Time limit: 1 second

On a birthday cake, four red candles and four green candles are placed. The boys and the girls in the party come up with a game to play. The girls play first and draw a line by connecting two red candles of their choice. The boys play next and must draw two non-intersecting lines, each connecting two green candles. Each candle can only be used once. If the boys can successfully draw two non-intersecting lines without crossing the line drawn by the girls, the boys win the game, if not the girls win.

In the example given in Fig. 1, the circles are the red candles and the triangles are the green candles. In this example, no matter how the girls connect the red candles, the boys can always find a way to draw two lines to connect the green candles to win the game.

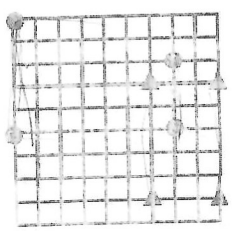


Figure 1: Boys can always win the game regardless of how girls play the game.

In Fig. 2, if the girls draw the dash line first, then the boys will have no mean to draw two lines without crossing the dash line. Therefore, girls win this game.

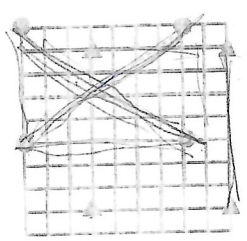


Figure 2: Boys cannot always win the game.

Given the positions of the 4 red candles and 4 green candles, please determine whether these is a way for girls to draw the first line so that boys cannot win the game.

Input File Format

The first line of the input contains one integer denoting the number of test cases to follow. For each test case, there are 2 lines. First line contains 8 integers, which are

Problem E
Twinkle, Twinkle, Little Triangle

Max no. of test cases: 30
Time limit: 1 second

Zeus plans to create a new star cluster. He establishes a two-dimensional coordinate system on a plane. The origin (0,0), has a star numbered as no.1. He extends three stars outward at intervals of 120°, positioned at $(L, 0)$, $(-\frac{1}{2}L, \frac{\sqrt{3}}{2}L)$, and $(-\frac{1}{2}L, -\frac{\sqrt{3}}{2}L)$, and are labeled as stars no.2, no. 3, and no. 4, respectively.

Furthermore, originating from each of the above three stars, no.2, no.3, and no.4, Zeus projects three additional stars outward with half of the previous length, namely $\frac{L}{2}$, and at intervals of 120°. These new stars are then rotated counterclockwise by 30° around the corresponding central star to determine their final positions. Following the same procedure, more new stars are generated iteratively. Note that from star i, three stars, namely stars no.$3i-1$, $3i$, and $3i+1$ are created. Consequently, stars originate from no.3 are named as no.8, no.9, and no.10; stars originate from no.4 are identified as no.11, no.12, and no.13; and stars originate from no. 5 are identified as no. 14, no. 15, and no. 16. An illustration of the above explanations can be found in Figure 1.

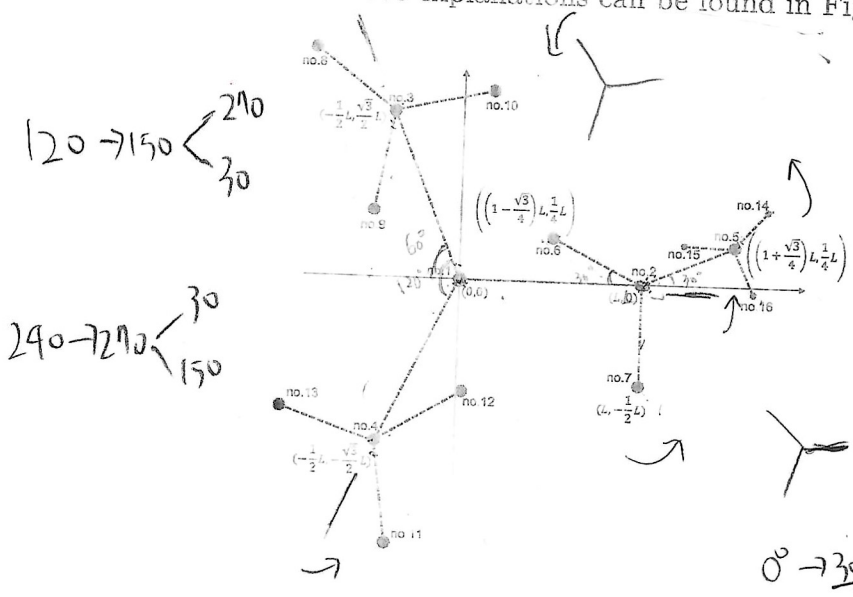


Figure 1: An illustration of forming the star cluster. Note placement of stars (black dots) in this figure may not be true to scale.

As shown in Fig. 1, stars no.5, no.6 and no.7 originate from star no.2, and are $\frac{L}{2}$ from star no.2. These 3 stars extend from star no.2 along the axis defined by star 2 and its central star (namely star 1), resulting in the points $(\frac{3}{2}L, 0)$, $(\frac{3}{4}L, \frac{\sqrt{3}}{4}L)$, and $(\frac{3}{4}L, -\frac{\sqrt{3}}{4}L)$. After a 30° counterclockwise rotation around star no.2, their positions become: $((1+\frac{\sqrt{3}}{4})L, \frac{1}{4}L)$, $((1-\frac{\sqrt{3}}{4})L, \frac{1}{4}L)$, and $(L, -\frac{1}{2}L)$. These three new stars are subsequently

designated as no.5, no.6, and no.7 in a counterclockwise sequence. Furthermore, stars no.14, no.15, and no.16, all originate from star no.5 along the axis defined by star 5 and its central star (namely star no.2). Their distance to star no.5 are all $\frac{L}{2 \times 2}$.

Given the star cluster formed from above procedure, please determine the type of triangles that is formed from any three stars.

Input File Format

The first line has an integer n ($n \leq 30$), denoting the number of test cases. The second line contains an integer L. The next n lines each has 3 integers, which are stars' numbers selected to form a triangle. There are at most 100 stars.

Output Format

For each test case, output whether the triangle formed is "Equilateral" or "Isosceles". If it is neither, then output whether the triangle is "Right", "Acute" or "Obtuse". If the three points cannot form a triangle, then output "Null" on a single line. For this task, length or degree accuracy is set to 10^{-5}.

Type	Definition
Equilateral	all three sides have the same length
Isosceles	only two sides have the same length
Right	one angle is right angle (equals to 90°)
Acute	angles are all acute (less than 90°)
Obtuse	one angle is obtuse (greater than 90°)

Sample Input

```
3
100
2 3 4
3 4 1
7 3 2
```

Output for the Sample Input

```
Equilateral
Isosceles
Obtuse
```

Problem F
Drones

Max no. of test cases: 30
Time limit: 2 seconds

StarLine company established N outpost stations along a long road on the moon. A drone patrols all N outpost stations every day, and the cost to operate the drone increases with the total distance. In fact, if the distance between the two farthest stations is d, then the cost is $10 + d^2$ (i.e. d squared, plus a fixed cost of 10). To cut the cost of drone operation, StarLine company decides to add more drones to patrol the outpost stations. For example, if 3 stations are located at position 0, 1, and 10, the cost to patrol with just one drone is $10 + 10^2$. With two drones (one drone patrols positions 0, 1, and one drone patrols position 10), the total cost will be reduced to $(10 + 1^2) + (10 + 0^2) = 21$. However the cost of patrol with 3 drones will be at least 30 which is higher than the cost with two drones.

Given the positions of N outpost stations, please determine the minimum cost to patrol all stations.

Input File Format

The first line of the input contains one integer denoting the number of test cases to follow. For each test case, the first line of the input contains one integer N, $1 \leq N \leq 1000$, which is the number of the outpost stations. The next line contains N integers in increasing order, which represent the positions of the N outpost stations. The positions are between 0 and 10,000, inclusive.

Output Format

For each test case, output an integer on a single line, which is the optimal (minimum) cost for patrolling all stations.

Sample Input

```
1
4
0 2 8 10
```

Output for the Sample Input

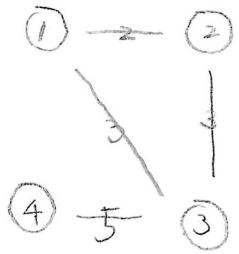
```
28
```

1 → n

Sample Input

5白天 3晚上

```
2
4 5 3
1 2 2
2 3 3
1 3 3
3 4 5
6 4 5 3
1 3 2
3 4 3
4 5 2
5 6 1
```

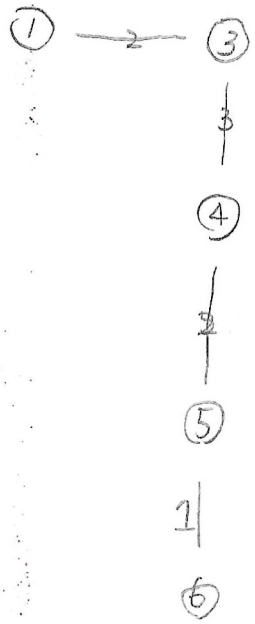


5+3+5 =13.

Output for the Sample Input

```
13
11
```

6格 4條路 5白天 3晚上



5+3+~~5~~

5+3+3 = 11.

Problem D
Little Red Riding Hood

Max no. of test cases: 20
Time limit: 2 seconds

Little Red Riding Hood is a beloved girl that everyone adores. One day, her grandmother fell ill, and she hoped to visit her grandmother as quickly as possible.

In Little Red Riding Hood's world, there are a total of a hours during the day and b hours at night. She can only travel through the villages during the day. If she's not in any village by nightfall, she will be eaten by the big bad wolf.

Little Red Riding Hood lives in Village 1, and her grandmother lives in Village n. The villages are connected by a network of roads, each requiring a certain amount of time to travel.

Since Little Red Riding Hood can only travel one kilometer per hour, starting from her village (Village 1), how long does it take her at the shortest to reach her grandmother's village without being endangered by the big bad wolf?

Input File Format

The first line of input contains an integer denoting the number of test cases to follow. For each test case, the first line contains four integers, n, m, a, and b, denoting that there are n villages, m roads connecting villages, a hours in the day and b hours in the night. The next m lines, each contains 3 integers, u, v, and w, denoting that there is a road connecting villages u and v with a distance of w.

- $2 \leq n \leq 2 \cdot 10^5$
- $0 \leq m \leq 2 \cdot 10^5$
- $1 \leq a, b, w \leq 10^9$
- $1 \leq u, v \leq n$, $u \neq v$

Output Format

For each test case, output a single integer on a line, denoting the shortest among of time needed for Little Red Riding Hood to reach grandma's village safely. If it is not possible to do so, output -1.

Sample Input

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

Output for the Sample Input

```
2
```

Problem B

Subarray Sums

Max no. of test cases: 10
Time limit: 2 seconds

Given an array X that stores a set of positive integers, please write a program to find the maximum value among the remainders obtained by dividing the sums of the subarrays in X by a given divisor. For example, if we have the array $X = [2, 1, 3, 4]$, then there are 10 subarrays, namely [2], [2, 1], [2, 1, 3], [2, 1, 3, 4], [1], [1, 3], [1, 3, 4], [3], [3, 4], [4]. And the subarray sums are: $2, 2+1=3, 2+1+3=6, 2+1+3+4=10, 1, 1+3=4, 1+3+4=8, 3, 3+4=7$, and 4. If the divisor is set to 5, the remainders are: 2, 3, 1, 0, 1, 4, 3, 3, 2, and 4, respectively. The maximum remainder value is 4 which would be from subarrays [1, 3] and [4].

Input File Format

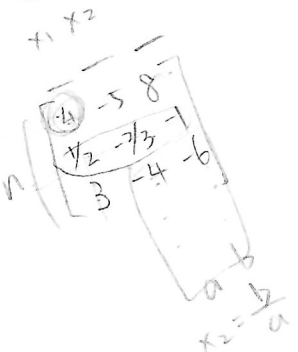
The first line of input contains an integer n, indicating the number of test cases. For each test case, the first line contains an integer s, $1 \leq s \leq 5000$, which is the size of the array X. The second line contains s positive integers (≤ 99999), indicating the s elements in the array X. The last line contains an integer d, representing the divisor.

Output Format

For each test case, output an integer on a single line, indicating the number of subarrays with the maximum remainder value.

$1+2+3+4=10 \equiv 0 \pmod 5 \rightarrow 4$

Sample Input



```
2
4 -5 8
3 2 12
2
1 2 3
2 4 6
2
1 2 3
1 2 4
2
1/2 -2/3 -1
1/4 1/9 3/2
0
```

input: str

check 第0項 → no change
else → + " "

solve: $\frac{p_1}{q_1} \quad \frac{p_2}{q_2} = \frac{p_3}{q_3}$

$p_1q_2q_3 \quad p_2q_1q_3 \quad p_3q_1q_2$

Sample Output

check: we know $x_1 = \frac{p_1}{q_1}$ $x_2 = \frac{p_2}{q_2}$

$A, B, C \in \mathbb{N}\mathbb{Z}$ $AX_1 + BX_2 = C$

$a_1x + b_1y = c_1$ $a_1a_2x + b_1a_2y = c_1a_2$ p_y $Ap_1q_2 + Bp_2q_1 = Cq_1q_2$

$a_2x + b_2y = c_2$ $a_1a_2x + b_2a_1y = c_2a_1$ q_y

$8\times3 - 12\times4 = -24$

$(-5)\times3 - 2\times4 = -23$

$y = \frac{c_1a_2 - c_2a_1}{b_1a_2 - b_2a_1}$

$x = \frac{c_1 - b_1y}{a_1}$

$= \frac{q_yc_1 - b_1p_y}{a_1q_y}$

check: $a_1 = 0$, $b_1 = 0$, $a_2 = 0$, $b_2 = 0$

```
n = 2
 +4 X1 -5 X2 = 8
 +3 X1 +2 X2 = 12

X1 = 76/23
X2 = 24/23

n = 2
 +1 X1 +2 X2 = 3
 +2 X1 +4 X2 = 6

infinite many solutions

n = 2
 +1 X1 +2 X2 = 3
 +1 X1 +2 X2 = 4

no solutions

n = 2
 +1/2 X1 -2/3 X2 = -1
 +1/4 X1 +1/9 X2 = 3/2

X1 = 4
X2 = 9/2
```

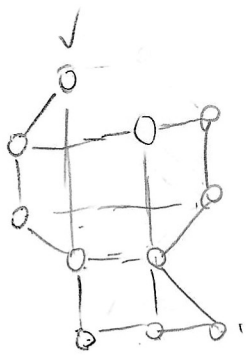
if n=1

check $a_1 \neq 0$ and $b_1 \neq 0$ → Y → infinite

or $a = b = c = 0$ → N → no solution

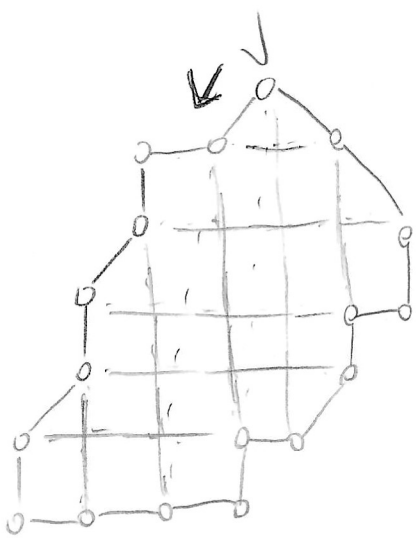
6 7 8 η 11 4 2 3 5 0

4 3 2
5
6η 8



6 5 η 6 η 6 η 1 1 1 3 1 2 3 1 3 4 4 0

4 3 2
5
6η 8



Problem C
Ladder Game

Max no. of test cases: 30
Time limit: 2 seconds

A ladder diagram consists of n vertical lines with some random horizontal line segments connecting two adjacent vertical lines at different positions along the vertical lines. There are distinct lottery numbers at the bottom of the vertical lines, corresponding to the prizes won. The game is played by choosing to start at the top of a vertical line and move down the lines until a horizontal line segment is reached. In which case, the player followed that horizontal line segment to the adjacent vertical line and continue to move downward. The process is continued until all players have reached the bottom of the vertical lines and have received their prizes. Note that all players choose and start at the top of distinct vertical line and that all players move at the same pace.

For example, in Fig. 1 the sample input shows 4 vertical lines. There is a horizontal line segment connecting vertical lines 1 and 2 at position 1 along the vertical lines.. There is also a horizontal line segment connecting vertical lines 3 and 4 at position 3 along the vertical lines. If four players play the game, each starting at different vertical line, players choosing vertical lines 1 and 2 will cross each other at the first horizontal line segment, while players choosing vertical lines 3 and 4 will cross each other at the second horizontal line segment.

Figure 1: A valid ladder game with 4 vertical lines and 2 horizontal line segments.

For this ladder game, It is guaranteed that horizontal line segments will not appear in the last line. Furthermore, there will not be consecutive horizontal line segments that joins 3 or more vertical lines at the same horizontal position. The horizontal line segments shown in Fig. 2 are not allowed.

Figure 2: Invalid ladder game.

During the Lunar New Year, Jack and his family are playing the ladder game to draw red envelops. However, to ensure fairness, Jack want to make sure no two player cross path (meet up somewhere) during the game. Given a ladder game diagram, please help Jack determine what's the maximum number of people that can play the game without possibly crossing each other.

Input File Format

The first of input contains a single integer denoting the number test cases to follow. For each test case, the first line has two integers, n and m, where $1 \leq n, m \leq 1000$, denoting n vertical lines of length m. The next m lines define the ladder game to be played. For the next m lines, each line contains $2 \times n - 1$ characters, the characters are "|" (at the odd positions, denoting vertical lines) and "_" or space (at the even positions, denoting whether there is a horizontal line segment connecting two adjacent vertical lines). All input ladder games are guaranteed to be valid games.

Output Format

For each test case, output a single integer denoting the maximum number of players can play the given ladder game without having the possibility to cross path.

Sample Input

```
1
4 4
|_|_| |
| | | |
| | |_|
| | | |
 1 3 5
```

n: 1 → 1 2n-3, 2n-1

n: 2 → 1, 3

n: 3 → 3, 5

n: 4 → 5

Output for the Sample Input

```
2
```

1 1 Template

1.1 template

```cpp
#include <bits/stdc++.h>
using namespace std;

#define F first
#define S second
#define ALL(v) v.begin(),v.end()
#define PB push_back
#define endl '\n'
//#define int long long
//#define float double
//#define float long double
#define FOR(i, a, b) for(int i = a; i < b; i++)
using uint = unsigned;
using ull = unsigned long long;
using ll = long long;
using ld = long double;
template<typename T> using Prior = priority_queue<T>;
template<typename T> using prior = priority_queue<T, vector<T>,
    greater<T>>;
const ll MOD = 1e9 + 7;
const double EPS = 1e-9;
void addmod(ll &a, ll b) {a = (a+b)%MOD;}
void submod(ll &a, ll b) {a = (a-b+MOD)%MOD;}
void timesmod(ll &a, ll b) {a = (a*b)%MOD;}
ll POW(ll a, ll b) {ll res=1; do{if(b%2)timesmod(res,a);
    timesmod(a,a);}while(b>>=1); return res;}
template<typename T>
void print(T a){
    for(auto u: a){
        cout << u << ' ';
    }
    cout << endl;
}

void solve() {
    ifstream fcin("input.txt");
    ofstream fcout("output.txt");

}

int main() {
    ios::sync_with_stdio(false),cin.tie(0);
    int t = 1;
    //cin >> t;
    while (t--) solve();
    return 0;
}
```

1.2 vscode io

```cpp
#include <bits/stdc++.h>
using namespace std;
int main(){
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    // 程式寫在這裡，上面兩行上傳前記得註解掉
}
```

1.3 vscode io

```python
import sys
sys.stdin = open("input.txt", "r")
sys.stdout = open("output.txt", "w")
# 程式寫在下面，上面三行上傳前記得註解掉
```

1.4 template python

```python
# --- 常用模組導入 ---
import sys      # 系統相關，用於快速輸入、設置遞歸限制
import math     # 數學函數 (例如 log, sqrt, pi, ceil, floor,
    gcd, factorial)

# --- 快速輸入設置 ---
# 在處理大量輸入時，sys.stdin.readline 比 input() 更快。
# 注意：sys.stdin.readline() 讀取會包含換行符，需要使用 .strip
    () 移除。
input = sys.stdin.readline

# --- 遞歸深度限制 (若題目涉及深層遞歸，例如 DFS) ---
# 默認遞歸深度通常是 1000，競賽中可能不夠。
# 題目通常不會需要這麼深，但以防萬一可取消註釋。
# sys.setrecursionlimit(10**6)

# --- 主解題邏輯函數 ---
def solve():
    # --- 輸入範例 (使用快速輸入) ---
    # 讀取一個整數
    n = int(input()) # n = 5

    # 讀取一行多個整數 (以空格分隔)
    arr = list(map(int, input().split())) # arr = [10, 20, 30,
        40, 50]

    # 讀取一個字符串
    s = input().strip() # s = "hello world" (記得 .strip() 移除
        換行符)

    # 讀取多行輸入直到 EOF (參考 eof_input.py)
    # while True:
    #     try:
    #         line = input().strip()
    #         if not line: # 處理空行情況
    #             continue
    #         # ... 處理 line ...
    #     except EOFError:
    #         break

    # --- 解題邏輯 ---
    # 這裡是你應用演算法和數據結構的地方。
    # 例如：
    # arr.sort() # 排序 (參考 sorting.py)
    # max_val = max(arr) # 求最大值
    # sum_val = sum(arr) # 求和
    # pi_val = math.pi # 使用 math 模組 (參考 math_simple.py,
        math_functions.py)

    # 範例：計算所有元素的和
    result = sum(arr)

    # --- 輸出範例 ---
    # 輸出單個值
    print(result) # 輸出 150

    # 輸出多個值 (自動以空格分隔)
    print(n, s) # 輸出 5 hello world

    # 輸出列表元素 (以空格分隔)
    print(*arr) # 輸出 10 20 30 40 50

    # 格式化輸出 (使用 % 運算符，參考 output_formatting.py)
    float_res = result / n # 150 / 5 = 30.0
    print("結果: %.2f" % float_res) # 輸出 結果: 30.00
    print("結果: {:.2f}".format(float_res)) # 輸出 結果: 30.00

    # 對於浮點數比較/精度 (參考 float_error.py)
    # if math.isclose(float_res, 30.0, rel_tol=1e-9):
    #     print("結果約等於 30.0")

# --- 主程序執行區塊 ---
if __name__ == "__main__":
    # TOPC 題目通常是單一測試案例，直接調用 solve() 即可。
    solve()

    # 如果題目有多個測試案例，通常會是這樣處理 (請根據題目說明
        決定是否使用)：
    # t = int(input()) # 讀取測試案例數量
    # for _ in range(t): # 循環 t 次
    #     solve() # 對每個測試案例調用 solve()
```

2 2 Datastruct

2.1 SCC

```cpp
struct SCC {
    int n, scc{}, _t{};
    vector<vector<int>> G;
    vector<int> dfn, low, id, stk;

    SCC(int _n) : n{_n}, G(_n + 1) {}

    void add_edge(int u, int v) {
        G[u].PB(v);
    }

    void dfs(int u) {
```

```
        dfn[u] = low[u] = _t++;
        stk.PB(u);

        for (int v : G[u]) {
            if (dfn[v] == -1) {
                dfs(v);
                low[u] = min(low[u], low[v]);
            } else if (id[v] == -1) {
                low[u] = min(low[u], dfn[v]);
            }
        }

        if (dfn[u] == low[u]) {
            int t;
            do {
                t = stk.back();
                stk.pop_back();
                id[t] = scc;
            } while (t != u);
            scc++;
        }
    }

    void work() {
        dfn.assign(n + 1, -1);
        low.assign(n + 1, -1);
        id.assign(n + 1, -1);

        for (int i = 1; i <= n; ++i) {
            if (dfn[i] == -1) {
                dfs(i);
            }
        }
    }

    vector<int> get_sizes() {
        vector<int> sz(scc, 0);
        for (int i = 1; i <= n; ++i) {
            if (id[i] != -1) {
                sz[id[i]]++;
            }
        }
        return sz;
    }
};
```

2.2 SegmentTree (Normal)

```
// 要有結合率
template<class Info>
struct SegmentTree {
    int n;
    std::vector<Info> info;
    SegmentTree() : n(0) {}
    SegmentTree(int n_, Info v_ = Info()) {
        init(n_, v_);
    }
    template<class T>
    SegmentTree(std::vector<T> init_) {
        init(init_);
    }
    void init(int n_, Info v_ = Info()) {
        init(std::vector(n_, v_));
    }
    template<class T>
    void init(std::vector<T> init_) {
        n = init_.size();
        info.assign(4 << std::__lg(n), Info());
        std::function<void(int, int, int)> build = [&](int p,
            int l, int r) {
            if (r - l == 1) {
                info[p] = init_[l];
                return;
            }
            int m = (l + r) / 2;
            build(2 * p, l, m);
            build(2 * p + 1, m, r);
            pull(p);
        };
        build(1, 0, n);
    }
    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }
    void modify(int p, int l, int r, int x, const Info &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        if (x < m) {
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }
    void modify(int p, const Info &v) {
        modify(1, 0, n, p, v);
    }

    void rangeAdd(int p, int l, int r, int x, int y, int v) {
        if (l >= y or r <= x) {
            return;
        }
        if (l >= x and r <= y) {
            info[p].x += v * (r - l);
            return;
        }
        int m = (l + r) / 2;
        rangeAdd(2 * p, l, m, x, y, v);
        rangeAdd(2 * p + 1, m, r, x, y, v);
        pull(p);
    }

    void rangeAdd(int x, int y, int v) {
        rangeAdd(1, 0, n, x, y, v);
    }

    Info rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y || r <= x) {
            return Info();
        }
        if (l >= x && r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p
            + 1, m, r, x, y);
    }
    Info rangeQuery(int l, int r) {
        return rangeQuery(1, 0, n, l, r);
    }

    // auto pred = [&](const Info &info) { return info.x > 1;};
    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F &&pred)
        {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findFirst(int l, int r, F &&pred) {
        return findFirst(1, 0, n, l, r, pred);
    }
    template<class F>
    int findLast(int p, int l, int r, int x, int y, F &&pred) {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findLast(int l, int r, F &&pred) {
        return findLast(1, 0, n, l, r, pred);
    }
};

struct Info {
    int x = 0;
};

Info operator+(const Info &a, const Info &b) {
    return {a.x + b.x};
}
```

2.3 DSU

```
// DSU 找是否在同一個集合 可以回溯
// 使用方式 DSU dsu_example(n) 就n個 應該沒有變的空間
// 功能有 find(x) 找x在哪個集合  same(x,y)確認x和y是不是在一樣的集合
// merge(x,y) 將x跟y在的集合合併 數量比較少小的放大數量比較多的
// t=time() 紀錄現在的時間  revert(t) 回朔到t的時候
// 功能皆為在後面+. 如 dsu_example.find(x)
struct DSU {
    vector<int> siz;
    vector<int> f;
    vector<array<int, 2>> his;

    DSU(int n) : siz(n + 1, 1), f(n + 1) {
        iota(f.begin(), f.end(), 0);
    }

    int find(int x) {
        while (f[x] != x) {
            x = f[x];
        }
        return x;
    }

    bool same(int x, int y) {
        return find(x) == find(y);
    }

    bool merge(int x, int y) {
        x = find(x);
        y = find(y);
        if (x == y) {
            return false;
        }
        if (siz[x] < siz[y]) {
            swap(x, y);
        }
        his.push_back({x, y});
        siz[x] += siz[y];
        f[y] = x;
        return true;
    }

    int time() {
        return his.size();
    }

    void revert(int tm) {
        while (his.size() > tm) {
            auto [x, y] = his.back();
            his.pop_back();
            f[y] = y;
            siz[x] -= siz[y];
        }
    }
};
```

2.4 BIT

```
// 要有結合率+可逆運算
// 2進位分解數列 適合只有線性改變and搜尋 修改/搜尋複雜度為O(Logn)
// 使用方法為 BIT<int> bit_example(n+1)  最少要n+1 超過好像沒影響(開)
// 功能有 add(x,value) 在第x個值 + value
// sum(x) [0,x)的總和 rangeSum(l,r) [l,r)的總和
// select(k) 找到 sum(x)>=k 的最小x
// 功能皆用方法皆為加. 如 bit_example.add(1,2)
template <class T>
struct BIT {
    int n;
    vector<T> a;
    BIT(int n_ = 0) {
        init(n_);
    }

    void init(int n_) {
        n = n_;
        a.assign(n, T{});
    }

    void add(int x, const T &v) {
        for (int i = x + 1; i <= n; i += i & -i) {
            a[i - 1] += v;
        }
    }

    T sum(int x) {
        T ans{};
        for (int i = x; i > 0; i -= i & -i) {
            ans += a[i - 1];
        }
        return ans;
    }

    T rangeSum(int l, int r) { // [l, r)
        return sum(r) - sum(l);
    }


    int select(const T &k) { // 前綴區間和 >= k 的位置
        int x = 0;
        T cur{};
        for (int i = 1 << __lg(n); i; i /= 2) {
            if (x + i <= n and cur + a[x + i - 1] <= k) {
                x += i;
                cur = cur + a[x - 1];
            }
        }
        return x;
    }
};
```

2.5 SegmentTree (Lazy)

```
template<class T>
struct SegmentTree{
    int n;
    vector<int> tag;
    vector<T> info;
    SegmentTree(int n_) : n(n_), tag(4 * n), info(4 * n) {}

    void pull(int p) {
        info[p] = info[2 * p] + info[2 * p + 1];
    }

    void add(int p, int v, int l, int r) {
        tag[p] += v;
        info[p].x += v * (r - l);
    }

    void push(int p, int l, int r) {
        if (tag[p] != 0) {
            int m = (l + r) / 2;
            add(2 * p + 1, tag[p], l, m);
            add(2 * p, tag[p], m, r);
            tag[p] = 0;
        }
    }

    T rangeQuery(int p, int l, int r, int x, int y) {
        if (l >= y or r <= x) {
            return T{};
        }
        if (l >= x and r <= y) {
            return info[p];
        }
        int m = (l + r) / 2;
        push(p, l, r);
        return rangeQuery(2 * p, l, m, x, y) + rangeQuery(2 * p + 1, m, r, x, y);
    }

    T rangeQuery(int x, int y) {
        return rangeQuery(1, 0, n, x, y);
    }

    void rangeAdd(int p, int l, int r, int x, int y, int v) {
        if (l >= y or r <= x) {
            return;
        }
        if (l >= x and r <= y) {
            add(p, v, l, r);
            return;
        }
        int m = (l + r) / 2;
        push(p, l, r);
        rangeAdd(2 * p, l, m, x, y, v);
        rangeAdd(2 * p + 1, m, r, x, y, v);
        pull(p);
    }

    void rangeAdd(int x, int y, int v) {
        rangeAdd(1, 0, n, x, y, v);
    }

    void modify(int p, int l, int r, int x, const T &v) {
        if (r - l == 1) {
            info[p] = v;
            return;
        }
        int m = (l + r) / 2;
        push(p, l, r);
        if (x < m) {
```

```
            modify(2 * p, l, m, x, v);
        } else {
            modify(2 * p + 1, m, r, x, v);
        }
        pull(p);
    }

    void modify(int x, const T &v) {
        modify(1, 0, n, x, v);
    }

    // auto pred = [&](const Info &info) { return info.x > 1;};
    template<class F>
    int findFirst(int p, int l, int r, int x, int y, F &&pred)
        {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findFirst(2 * p, l, m, x, y, pred);
        if (res == -1) {
            res = findFirst(2 * p + 1, m, r, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findFirst(int l, int r, F &&pred) {
        return findFirst(1, 0, n, l, r, pred);
    }
    template<class F>
    int findLast(int p, int l, int r, int x, int y, F &&pred) {
        if (l >= y || r <= x) {
            return -1;
        }
        if (l >= x && r <= y && !pred(info[p])) {
            return -1;
        }
        if (r - l == 1) {
            return l;
        }
        int m = (l + r) / 2;
        int res = findLast(2 * p + 1, m, r, x, y, pred);
        if (res == -1) {
            res = findLast(2 * p, l, m, x, y, pred);
        }
        return res;
    }
    template<class F>
    int findLast(int l, int r, F &&pred) {
        return findLast(1, 0, n, l, r, pred);
    }
};

struct Info{
    int x = 0;
};

Info operator+(const Info &a, const Info &b) {
    return {a.x + b.x};
}
```

3 3 Datastruct Default

3.1 queue

```
int main() {
    // 競賽時建議加上這兩行，加速 C++ 輸入輸出
    ios::sync_with_stdio(false);
    cin.tie(0);

    cout << "--- std::queue (佇列: 先進先出 FIFO) ---" << endl;
    queue<string> q; // 建立一個存儲 string 的佇列

    q.push("apple"); // push(value): 將元素加入佇列尾部
    q.push("banana");
    q.push("cherry");
    cout << "佇列前端元素 (q.front()): " << q.front() << endl;
        // 取得佇列前端元素 (不移除)，結果 apple
    cout << "佇列尾端元素 (q.back()): " << q.back() << endl;
        // 取得佇列尾端元素 (不移除)，結果 cherry
    cout << "佇列大小 (q.size()): " << q.size() << endl;
        // 取得佇列中元素數量，結果 3

    q.pop(); // pop(): 移除佇列前端元素 (不返回任何值)
    cout << "pop 後佇列前端: " << q.front() << endl;
        // 移除 apple 後，結果 banana
    cout << "佇列是否為空 (q.empty()): " << (q.empty() ? "是" :
        "否") << endl; // 判斷佇列是否為空，結果 否

    q.pop();
    q.pop();
    cout << "pop 完後佇列是否為空: " << (q.empty() ? "是" : "否
        ") << endl; // 結果 是

    return 0;
}
```

3.2 stack

```
int main() {
    // 競賽時建議加上這兩行，加速 C++ 輸入輸出
    ios::sync_with_stdio(false);
    cin.tie(0);

    cout << "--- std::stack (堆疊: 後進先出 LIFO) ---" << endl;
    stack<int> s; // 建立一個存儲 int 的堆疊

    s.push(10); // push(value): 將元素加入堆疊頂部
    s.push(20);
    s.push(30);
    cout << "堆疊頂部元素 (s.top()): " << s.top() << endl; //
        取得堆疊頂部元素 (不移除)，結果 30
    cout << "堆疊大小 (s.size()): " << s.size() << endl;   //
        取得堆疊中元素數量，結果 3

    s.pop(); // pop(): 移除堆疊頂部元素 (不返回任何值)
    cout << "pop 後堆疊頂部: " << s.top() << endl;          //
        移除 30 後，結果 20
    cout << "堆疊是否為空 (s.empty()): " << (s.empty() ? "是" :
        "否") << endl; // 判斷堆疊是否為空，結果 否

    s.pop();
    s.pop();
    cout << "pop 完後堆疊是否為空: " << (s.empty() ? "是" : "否
        ") << endl; // 結果 是

    return 0;
}
```

3.3 priority queue

```
int main() {
    // 競賽時建議加上這兩行，加速 C++ 輸入輸出
    ios::sync_with_stdio(false);
    cin.tie(0);

    cout << "--- std::priority_queue (優先級佇列: 默認最大堆)
        ---" << endl;
    // 默認是最大堆 (max-heap)，即 top() 總是返回最大的元素。
    priority_queue<int> pq_max; // 建立一個存儲 int 的最大堆

    pq_max.push(10); // push(value): 將元素加入堆中，自動保持堆
        屬性
    pq_max.push(30);
    pq_max.push(20);
    cout << "最大堆頂部元素 (pq_max.top()): " << pq_max.top()
        << endl; // 取得堆中最大元素 (不移除)，結果 30
    cout << "最大堆大小 (pq_max.size()): " << pq_max.size() <<
        endl;   // 取得堆中元素數量，結果 3

    pq_max.pop(); // pop(): 移除堆頂元素 (最大的，不返回任何值)
    cout << "pop 後最大堆頂部: " << pq_max.top() << endl;
                // 移除 30 後，結果 20
    cout << "最大堆是否為空 (pq_max.empty()): " << (pq_max.
        empty() ? "是" : "否") << endl; // 判斷是否為空

    cout << endl;

    cout << "--- std::priority_queue (最小堆) ---" << endl;
    // 若要最小堆 (min-heap)，即 top() 總是返回最小的元素，
    // 需要指定比較器 `greater<int>` 和底層容器 `vector<int>`。
    priority_queue<int, vector<int>, greater<int>> pq_min; //
        建立一個存儲 int 的最小堆

    pq_min.push(10);
    pq_min.push(30);
    pq_min.push(20);
    cout << "最小堆頂部元素 (pq_min.top()): " << pq_min.top()
        << endl; // 取得堆中最小元素 (不移除)，結果 10
```

```

# 浮點數輸出精度控制 (使用 % 格式化)
value = 1.0 / 3.0 # 0.3333333333333333
print("原始值: %f" % value) # 原始值: 0.333333
print("保留7位小數: %.7f" % value) # 保留7位小數: 0.3333333
print("保留2位小數: %.2f" % value) # 保留2位小數: 0.33
# 這邊的輸出格式化會自動進行四捨五入。
```

9.7 sorting

```
my_list = [3, 1, 4, 1, 5, 9, 2, 6]
words = ["banana", "apple", "cherry", "kiwi"]
data_tuples = [(10, 5), (20, 2), (5, 8), (15, 2)]

# list.sort() 方法 (就地排序，直接修改原列表)
list_a = list(my_list) # 複製一份，避免影響後續範例
list_a.sort() # 升序排序
print(list_a) # [1, 1, 2, 3, 4, 5, 6, 9]

list_b = list(words)
list_b.sort(reverse=True) # 降序排序
print(list_b) # ['kiwi', 'cherry', 'banana', 'apple']

list_c = list(words)
list_c.sort(key=len) # 按字符串長度升序排序
print(list_c) # ['kiwi', 'apple', 'banana', 'cherry']

list_d = list(data_tuples)
list_d.sort(key=lambda item: item[1]) # 按元組的第二個元素升序排序
print(list_d) # [(20, 2), (15, 2), (10, 5), (5, 8)]

# sorted() 函數 (返回一個新的排序列表，不修改原列表)
original_list = [3, 1, 4, 1]
new_sorted_list = sorted(original_list) # 升序排序
print(new_sorted_list) # [1, 1, 3, 4]
print(original_list) # [3, 1, 4, 1] (原列表不變)

new_sorted_desc = sorted(words, reverse=True) # 降序排序
print(new_sorted_desc) # ['kiwi', 'cherry', 'banana', 'apple']
```

9.8 四捨五入

```
import math
def myRound(x, d):
    mul = 1
    for i in range(d): mul *= 10
    if x >= 0: return math.floor(x * mul + 0.5) / mul
    else: return math.ceil(x * mul - 0.5) / mul
```

9.9 math

```
import math
b = 49
print(math.sqrt(b)) # 7.0 (計算平方根)
print(math.pi) # 3.141592653589793 (圓周率常數)
print(math.sin(math.pi/2)) # sin90 度= 1.0 (計算正弦值，參數為弧度)
print(math.factorial(5)) # 階層 5! = 120 (計算階乘)
print(math.gcd(56, 42)) # 最大公因數 14 (計算最大公因數)
math.comb(n, k) # n! / (k! * (n - k)!) (計算組合數)
math.fmod(x, y) # 在使用浮點數時通常是首選的取餘數方法
math.log(x, base) # 計算以 base 為底的對數，預設 base 為 e (自然對數)
math.ceil(x) # 傳回大於或等於 x 的最小整數 (roundup)
math.floor(x) # 傳回小於或等於 x 的最大整數 (rounddown)
math.pow(x, y) # 計算 x 的 y 次方 (x**y)
math.degrees(x) # 將弧度轉換為角度
math.radians(x) # 將角度轉換為弧度
math.dist(p, q) # 計算兩點 p 和 q 之間的歐氏距離 (p 和 q 為座標序列)
```

9.10 output formatting

```
int_val = 42
float_val = 3.14159265
another_float_val = 123.456789
str_val = "Hello TOPC"

# 格式化整數
print("整數: %d" % int_val) # 整數: 42

# 格式化浮點數 (默認顯示6位小數，會四捨五入)
print("浮點數: %f" % float_val) # 浮點數: 3.141593

# 格式化浮點數並指定小數位數 (%.nf 表示保留 n 位小數，會四捨五入)
print("保留2位小數: %.2f" % another_float_val) # 保留2位小數: 123.46
print("保留4位小數: %.4f" % float_val) # 保留4位小數: 3.1416

# 格式化字符串
print("字符串: %s" % str_val) # 字符串: Hello TOPC

# 同時格式化多個值 (將所有要格式化的變數放入一個元組)
name = "Alice"
age = 30
height = 1.75
print("姓名: %s, 年齡: %d, 身高: %.2f米" % (name, age, height))
      # 姓名: Alice, 年齡: 30, 身高: 1.75米
```

9.11 structure list

```
# --- 建立列表 ---
# 列表是有序、可變、可包含重複元素的序列。

empty_list = [] # 建立一個空列表
print(f"空列表: {empty_list}") # []

# 直接定義列表元素
my_list = [1, 2, 3, 'apple', 'banana', True]
print(f"一般列表: {my_list}") # [1, 2, 3, 'apple', 'banana', True]

# 使用 list() 函數從其他可迭代對象建立
str_to_list = list("hello")
print(f"從字符串建立: {str_to_list}") # ['h', 'e', 'l', 'l', 'o']

# 列表推導式 (List Comprehension) - 簡潔建立或轉換列表
squares = [i**2 for i in range(5)] # 建立 0 到 4 的平方
print(f"列表推導式範例: {squares}") # [0, 1, 4, 9, 16]

# --- 訪問列表元素 (索引) ---
# 索引從 0 開始
fruits = ['apple', 'banana', 'cherry', 'date']
print(f"第一個元素: {fruits[0]}") # apple
print(f"第三個元素: {fruits[2]}") # cherry

# 負數索引 (從列表末尾開始計算，-1 表示最後一個元素)
print(f"最後一個元素: {fruits[-1]}") # date
print(f"倒數第二個元素: {fruits[-2]}") # cherry

# --- 列表切片 (Slicing) ---
# [start:end:step]
# start (包含), end (不包含), step (步長，默認為 1)
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(f"從索引 2 到 5 (不含): {numbers[2:5]}") # [2, 3, 4]
print(f"從開頭到索引 3 (不含): {numbers[:3]}") # [0, 1, 2]
print(f"從索引 5 到結尾: {numbers[5:]}") # [5, 6, 7, 8, 9]
print(f"整個列表: {numbers[:]}") # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(f"每隔一個取值: {numbers[::2]}") # [0, 2, 4, 6, 8]
print(f"反轉列表: {numbers[::-1]}") # [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

# --- 修改列表元素 ---
my_nums = [10, 20, 30, 40]
my_nums[1] = 25 # 修改索引 1 的元素
print(f"修改元素後: {my_nums}") # [10, 25, 30, 40]

my_nums[2:4] = [35, 45, 55] # 修改切片範圍內的元素 (可以改變列表長度)
print(f"修改切片後: {my_nums}") # [10, 25, 35, 45, 55]

# --- 增加元素 ---
add_list = [1, 2]
add_list.append(3) # 在列表末尾添加一個元素
print(f"append 後: {add_list}") # [1, 2, 3]

add_list.extend([4, 5]) # 在列表末尾添加多個元素 (使用另一個可迭代對象)
print(f"extend 後: {add_list}") # [1, 2, 3, 4, 5]
```

```python
add_list.insert(1, 1.5) # 在指定索引位置插入一個元素
print(f"insert 後: {add_list}") # [1, 1.5, 2, 3, 4, 5]

# --- 刪除元素 ---
remove_list = ['a', 'b', 'c', 'd', 'e']
removed_item = remove_list.pop() # 移除並返回列表末尾的元素
print(f"pop 末尾元素: {removed_item}, 列表: {remove_list}") # e
    , ['a', 'b', 'c', 'd']

removed_at_index = remove_list.pop(1) # 移除並返回指定索引的元
    素
print(f"pop 索引 1 元素: {removed_at_index}, 列表: {remove_list
    }") # b, ['a', 'c', 'd']

remove_list.remove('c') # 移除列表中第一個匹配的元素 (若不存在
    會報 ValueError)
print(f"remove 'c' 後: {remove_list}") # ['a', 'd']

del remove_list[0] # 刪除指定索引的元素
print(f"del 索引 0 後: {remove_list}") # ['d']

del remove_list[0:0] # 刪除切片範圍內的元素 (若範圍為空則不刪除
    )
print(f"del 空切片後: {remove_list}") # ['d']

# del remove_list # 刪除整個列表變數
# remove_list.clear() # 清空列表所有元素

# --- 遍歷列表 ---
iterate_list = ['one', 'two', 'three']

print("\n遍歷列表 (for 迴圈):")
for item in iterate_list:
    print(item) # one, two, three

print("遍歷列表 (for 迴圈, 帶索引):")
for i, item in enumerate(iterate_list):
    print(f"索引: {i}, 元素: {item}") # 索引: 0, 元素: one / 索
        引: 1, 元素: two / ...

# --- 其他常用列表操作 ---
check_list = [1, 2, 3, 2, 4]

# 列表長度
print(f"列表長度: {len(check_list)}") # 5

# 檢查元素是否存在
print(f"3 在列表中嗎? {3 in check_list}") # True
print(f"5 不在列表中嗎? {5 not in check_list}") # True

# 元素計數
print(f"元素 2 出現次數: {check_list.count(2)}") # 2

# 查找元素索引 (返回第一個匹配的索引, 若不存在會報 ValueError)
print(f"元素 4 的索引: {check_list.index(4)}") # 4

# 列表排序 (請參考 sorting.py 檔案)
# check_list.sort() # 就地排序
# sorted_list = sorted(check_list) # 返回新列表

# 反轉列表
check_list.reverse() # 就地反轉列表
print(f"反轉後: {check_list}") # [4, 2, 3, 2, 1]

# 複製列表
# copy_of_list = check_list[:] # 切片方式複製
# copy_of_list = list(check_list) # list() 函數複製
# copy_of_list = check_list.copy() # .copy() 方法複製
```

9.12 structure dictionary

```python
# --- 建立字典 ---
# 字典是鍵值對的無序集合, 鍵必須是唯一的且不可變 (如數字、字符
    串、元組)。

empty_dict = {} # 建立一個空字典
print(f"空字典: {empty_dict}") # {}

# 直接定義鍵值對
my_dict = {'apple': 1, 'banana': 2, 'cherry': 3}
print(f"一般字典: {my_dict}") # {'apple': 1, 'banana': 2, '
    cherry': 3}

# 從鍵值對的列表/元組建立字典
dict_from_pairs = dict([('a', 10), ('b', 20)])
print(f"從鍵值對列表建立: {dict_from_pairs}") # {'a': 10, 'b':
    20}

# --- 訪問字典元素 ---
# 1. 直接使用鍵訪問 (若鍵不存在會報 KeyError)
print(f"apple 的值: {my_dict['apple']}") # 1

# 2. 使用 .get() 訪問 (若鍵不存在返回 None 或指定預設值)
print(f"banana 的值 (get): {my_dict.get('banana')}") # 2
print(f"grape 的值 (get, 不存在): {my_dict.get('grape')}") #
    None
print(f"grape 的值 (get, 預設為 0): {my_dict.get('grape', 0)}")
    # 0

# --- 新增與修改元素 ---
my_dict['date'] = 4 # 新增一個鍵值對
print(f"新增 'date': {my_dict}") # {'apple': 1, 'banana': 2, '
    cherry': 3, 'date': 4}

my_dict['apple'] = 10 # 修改 'apple' 的值
print(f"修改 'apple': {my_dict}") # {'apple': 10, 'banana': 2,
    'cherry': 3, 'date': 4}

# --- 移除元素 ---
del my_dict['cherry'] # 刪除指定鍵值對 (若鍵不存在會報 KeyError
    )
print(f"刪除 'cherry' 後: {my_dict}") # {'apple': 10, 'banana':
    2, 'date': 4}

popped_value = my_dict.pop('date') # 移除並返回指定鍵的值 (若鍵
    不存在會報 KeyError)
print(f"pop 'date' 的值: {popped_value}, 字典: {my_dict}") # 4,
    {'apple': 10, 'banana': 2}

# .pop() 也可以帶預設值，避免 KeyError
popped_non_existent = my_dict.pop('grape', -1) # 如果 'grape'
    不存在，返回 -1
print(f"pop 'grape' (不存在): {popped_non_existent}, 字典: {
    my_dict}") # -1, {'apple': 10, 'banana': 2}

my_dict.clear() # 清空所有元素
print(f"清空後: {my_dict}") # {}

# --- 遍歷字典 ---
iter_dict = {'a': 1, 'b': 2, 'c': 3}

print("\n遍歷鍵 (默認):")
for key in iter_dict:
    print(key) # a, b, c

print("遍歷值 (.values()):")
for value in iter_dict.values():
    print(value) # 1, 2, 3

print("遍歷鍵值對 (.items()):")
for key, value in iter_dict.items():
    print(f"鍵: {key}, 值: {value}") # 鍵: a, 值: 1 / 鍵: b, 值
        : 2 / 鍵: c, 值: 3

# --- 其他常用操作 ---
check_dict = {'x': 10, 'y': 20, 'z': 30}

# 檢查鍵是否存在
print(f"'x' 在字典中嗎? {'x' in check_dict}") # True
print(f"'w' 在字典中嗎? {'w' in check_dict}") # False

# 取得字典長度 (鍵的數量)
print(f"字典長度: {len(check_dict)}") # 3

# 複製字典 (淺複製)
copied_dict = check_dict.copy()
print(f"複製的字典: {copied_dict}") # {'x': 10, 'y': 20, 'z':
    30}
```

9.13 math functions

```python
import math # 導入 math 模組

# 對數運算
num_log = 100
print(math.log(num_log)) # 以自然常數e為底的對數 (約 4.605)
print(math.log10(num_log)) # 以10為底的對數 (2.0)
print(math.log(num_log, 2)) # 以2為底的對數 (約 6.643)

# 大數對數 (Python 的整數支持任意精度)
large_number = 10**500000 # 產生一個非常大的整數 (1後面有50萬個
    0)
print(math.log(large_number, 6)) # 計算這個大數以6為底的對數 (
    約 643243.684)
```

```

# 乘方運算
print(2 ** 10) # 2的10次方 (1024, 返回整數)
print(math.pow(2, 10)) # 2的10次方 (1024.0, 返回浮點數)
```

10 Shortest-Path

10.1 Floyd-Warshall

```
// Floyd-Warshall Algorithm
// 多點最短路徑 時間O(n^3) 空間O(n^2)
// 邏輯:a to c加個中間點b 如果比較短就改a to b to c
// path為距離 to為移動路徑(都是二維vector) to[i][j] = j  cin >>
    path[i][j]
FOR(mid,0,n){
    FOR(start,0,n){
        FOR(end,0,n){
            if(path[start][mid] + path[mid][end] < path[start][
                end]){
                path[start][end] = path[start][mid] + path[mid
                    ][end];
                to[start][end] = to[start][mid];
            }
        }
    }
}
```

11 geometry

11.1 PointInPolygon

```
// 判斷一個點是在多邊形內部、外部還是邊界上。
int inPoly(Pt p, const vector <Pt> &P) {
    const int n = P.size();
    int cnt = 0;
    for (int i = 0; i < n; i++) {
        Pt a = P[i], b = P[(i + 1) % n];
        if (PtOnSeg(p, {a, b})) return 1; // on edge
        if ((sgn(a.y - p.y) == 1) ^ (sgn(b.y - p.y) == 1))
            cnt += sgn(ori(a, b, p));
    }
    return cnt == 0 ? 0 : 2; // out, in
}
```

11.2 IntersectionOfCircles

```
// 計算兩個圓的所有交點。
vector <Pt> CircleInter(Cir a, Cir b) {
    double d2 = abs2(a.o - b.o), d = sqrt(d2);
    if (d < max(a.r, b.r) - min(a.r, b.r) || d > a.r +
b.r) return {};
    Pt u = (a.o + b.o) / 2 + (a.o - b.o) * ((b.r * b.r - a.r *
        a.r) / (2 * d2));
    double A = sqrt((a.r + b.r + d) * (a.r - b.r + d) *
(a.r + b.r - d) * (-a.r + b.r + d));
    Pt v = rotate(b.o - a.o) * A / (2 * d2);
    if (sgn(v.x) == 0 and sgn(v.y) == 0) return {u};
    return {u - v, u + v}; // counter clockwise of a
b);
}
```

11.3 TangentLinesOfCircles

```
// 計算兩個圓之間的所有公切線。
vector <Line> CircleTangent(Cir c1, Cir c2, int sign1) {
    // sign1 = 1 for outer tang, -1 for inter tang
    vector <Line> ret;
    double d_sq = abs2(c1.o - c2.o);
    if (sgn(d_sq) == 0) return ret;
    double d = sqrt(d_sq);
    Pt v = (c2.o - c1.o) / d;
    double c = (c1.r - sign1 * c2.r) / d;
    if (c * c > 1) return ret;
    double h = sqrt(max(0.0, 1.0 - c * c));
    for (int sign2 = 1; sign2 >= -1; sign2 -= 2) {
        Pt n = Pt(v.x * c - sign2 * h * v.y, v.y * c + sign2 *
            h * v.x);
        Pt p1 = c1.o + n * c1.r;
        Pt p2 = c2.o + n * (c2.r * sign1);
        if (sgn(p1.x - p2.x) == 0 && sgn(p1.y - p2.y)
== 0)
            p2 = p1 + rotate(c2.o - c1.o);
        ret.push_back({p1, p2});
    }
    return ret;
}
```

11.4 IntersectionOfLines

```
// 判斷兩線段是否相交及計算兩直線交點。
bool isInter(Line l, Line m) {
    if (PtOnSeg(m.a, l) or PtOnSeg(m.b, l) or
        PtOnSeg(l.a, m) or PtOnSeg(l.b, m))
        return true;
    return PtSide(m.a, l) * PtSide(m.b, l) < 0 and
           PtSide(l.a, m) * PtSide(l.b, m) < 0;
}
Pt LineInter(Line l, Line m) {
    double s = ori(m.a, m.b, l.a), t = ori(m.a, m.b, l.b);
    return (l.b * s - l.a * t) / (s - t);
}
bool strictInter(Line l, Line m) {
    int la = PtSide(m.a, l);
    int lb = PtSide(m.b, l);
    int ma = PtSide(l.a, m);
    int mb = PtSide(l.b, m);
    if (la == 0 and lb == 0) return false;
    return la * lb < 0 and ma * mb < 0;
}
```

11.5 DynamicConvexHull

```
// 動態維護凸包，支援點的插入與點內查詢。
template <class T, class Comp = less<T>>
struct DynamicHull {
    set<T, Comp> H;
    void insert(T p) {
        if (inside(p)) return;
        auto it = H.insert(p).ff;
        while (it != H.begin() and prev(it) != H.begin() \
               and ori(*prev(it, 2), _prev(it),_ it) <= 0) {
            it = H.erase(--it);
        }
        while (it != --H.end() and next(it) != --H.end() \
               and ori(*it, _next(it),_ next(it, 2)) <= 0) {
            it = --H.erase(++it);
        }
    }
    int inside(T p) { // 0: out, 1: on, 2: in
        auto it = H.lower_bound(p);
        if (it == H.end()) return 0;
        if (it == H.begin()) return p *it;
        return 1 - sgn(ori(*prev(it), p, *it));
    }
};
// DynamicHull <Pt> D;
// DynamicHull <Pt, greater <>> U;
// D.inside(p) and U.inside(p)
```

11.6 AreaOfCirclePolygon

```
// 計算圓與多邊形重疊的面積。
double CirclePoly(Cir C, const vector <Pt> &P) {
    auto arg = [&](Pt p, Pt q) { return atan2(p ^ q, p * q); };
    double r2 = C.r * C.r / 2;
    auto tri = [&](Pt p, Pt q) {
        Pt d = q - p;
        auto a = (d * p) / abs2(d), b = (abs2(p) - C.r * C.r)/
            abs2(d);
        auto det = a * a - b;
        if (det <= 0) return arg(p, q) * r2;
        auto s = max(0., -a - sqrt(det)), t = min(1., -a + sqrt
            (det));
        if (t < 0 or 1 <= s) return arg(p, q) * r2;
        Pt u = p + d * s, v = p + d * t;
        return arg(p, u) * r2 + (u ^ v) / 2 + arg(v, q) * r2;
    };
}
```

11.7 SweepLine

```
// 掃描線算法框架，用於處理矩形面積並等問題。
struct node{
    node *lch,*rch;
    int val, sum;

    node () {
        lch = rch = nullptr;
        val = sum = 0;
    }

    void pull(){
        sum = 0;
        if(lch) sum += lch -> sum;
        if(rch) sum += rch -> sum;
    }

    void modify(int l, int r, int lb, int rb, int v){
        if(l <= lb && rb <= r){
            val += v;
            if(val) sum = rb - lb + 1;
            else pull();
            return ;
        }
        int mid = lb + rb >> 1;
        if(!lch) lch = new node();
        if(!rch) rch = new node();
        if(l <= mid) lch -> modify(l, r, lb, mid, v);
        if(mid < r) rch -> modify(l, r, mid+1, rb, v);
        if(val) sum = rb - lb + 1;
        else pull();
    }
};
struct segment{
    int L, R, H, V;
};
void solve(){
    int n;
    cin >> n;
    vec<segment > v;
    rep(i, n){
        int l, r, u, d;
        cin >> l >> r >> d >> u;
        v.EB(segment{l+1, r, d, 1});
        v.EB(segment{l+1, r, u, -1});
    }
    sort(ALL(v), [](segment a, segment b){
        return a.H < b.H;
    });
    node *root = new node();
    int cur = v.front().H;
    ll ans = 0;
    for(auto i:v){
        if(i.H != cur){
            ans += (ll)root -> sum * (i.H - cur);
            cur = i.H;
        }
        root -> modify(i.L, i.R, 1, 1e6, i.V);
    }
    ans += (ll)root -> sum * (v.back().H - cur);
    cout << ans << endl;
}
```

11.8 HalfPlaneIntersection

```
// 計算多個半平面的交集區域（通常為凸多邊形）。
bool cover(Line L, Line P, Line Q) {
    // return PtSide(LineInter(P, Q), L) <= 0; for
    i128 u = (Q.a - P.a) ^ Q.dir();
    i128 v = P.dir() ^ Q.dir();
    i128 x = P.dir().x * u + (P.a - L.a).x * v;
    i128 y = P.dir().y * u + (P.a - L.a).y * v;
    return sgn(x * L.dir().y - y * L.dir().x) * sgn(v) >= 0;
}
vector <Line> HPI(vector <Line> P) {
    sort(all(P), [&](Line l, Line m) {
        if (argcmp(l.dir(), m.dir())) return true;
        if (argcmp(m.dir(), l.dir())) return false;
        return ori(m.a, m.b, l.a) > 0;
    });
    int n = P.size(), l = 0, r = -1;
    for (int i = 0; i < n; i++) {
        if (i and !argcmp(P[i - 1].dir(), P[i].dir()))
            continue;
        while (l < r and cover(P[i], P[r - 1], P[r])) r--;
        while (l < r and cover(P[i], P[l], P[l + 1])) l++;
        P[++r] = P[i];
    }
    while (l < r and cover(P[l], P[r - 1], P[r])) r--;
    while (l < r and cover(P[r], P[l], P[l + 1])) l++;
    if (r - l <= 1 or !argcmp(P[l].dir(), P[r].dir()))
        return {}; // empty
    if (cover(P[l + 1], P[l], P[r]))
        return {}; // infinity
    return vector(P.begin() + l, P.begin() + r + 1);
}
```

11.9 SortPoint

```
// 對點集進行排序（通常用於極角排序或掃描線算法）。
void sortPoint(vec<Point > &v) {
    // counterclockwise from x <= 0, y = 0
    vec<Point > top, bot;
    int cnt = 0;
    for(auto i:v) {
        if(i.F == 0 && i.S == 0)
            cnt++;
        else if(i.S < 0 || (i.S == 0 && i.F > 0))
            bot.EB(i);
        else
            top.EB(i);
    }
    sort(ALL(bot), [](Point a, Point b) {
        return ori({0,0}, a, b) > 0;
    });
    for(auto i:bot) v.EB(i);
    while(cnt--) v.EB(0,0);
    sort(ALL(top), [](Point a, Point b) {
        return ori({0, 0}, a, b) > 0;
    });
    for(auto i:top) v.EB(i);
}
```

11.10 Circle

```
// 定義圓結構，並判斷兩個圓之間是否分離或包含。
struct Cir {
    Pt o;
    double r;
};
bool disjunct(const Cir &a, const Cir &b) {
    return sgn(abs(a.o - b.o) - a.r - b.r) >= 0;
}
bool contain(const Cir &a, const Cir &b) {
    return sgn(a.r - b.r - abs(a.o - b.o)) >= 0;
}
```

11.11 MinimalEnclosingCircle

```
// 計算給定點集的最小覆蓋圓。
Pt Center(Pt a, Pt b, Pt c) {
    Pt x = (a + b) / 2;
    Pt y = (b + c) / 2;
    return LineInter({x, x + rotate(b - a)}, {y, y + rotate(c -
        b)});
}
Cir MEC(vector <Pt> P) {
    mt19937 rng(time(0));
    shuffle(all(P), rng);
    Cir C;
    for (int i = 0; i < P.size(); i++) {
        if (C.inside(P[i])) continue;
        C = {P[i], 0};
        for (int j = 0; j < i; j++) {
            if (C.inside(P[j])) continue;
            C = {(P[i] + P[j]) / 2, abs(P[i] - P[j]) /
2};
            for (int k = 0; k < j; k++) {
                if (C.inside(P[k])) continue;
                C.o = Center(P[i], P[j], P[k]);
                C.r = abs(C.o - P[i]);
            }
        }
    }
    return C;
}
```

11.12 IntersectionOfCircleLine

```
// 計算圓與直線的所有交點。
vector <Pt> CircleLineInter(Cir c, Line l) {
    Pt H = proj(c.o, l);
    Pt dir = unit(l.b - l.a);
    double h = abs(H - c.o);
    if (sgn(h - c.r) > 0) return {};
    double d = sqrt(max((double)0., c.r * c.r - h * h));
```

```
    if (sgn(d) == 0) return {H};
    return {H - dir * d, H + dir * d};
    // Counterclockwise
}
```

11.13 ConvexHullAndrew

```
// 計算點集的凸包 (Andrew's Monotone Chain 變體)。
vector <Pt> Hull(vector <Pt> P) {
    sort(all(P));
    P.erase(unique(all(P)), P.end());
    P.insert(P.end(), P.rbegin() + 1, P.rend());
    vector <Pt> stk;
    for (auto p : P) {
        auto it = stk.rbegin();
        while (stk.rend() - it >= 2 and ori(*next(it), *it, p)
             <= 0 and ((*next(it) < _it) == (_it < p))) {
            it++;
            stk.resize(stk.rend() - it);
        }
        stk.push_back(p);
    }
    stk.pop_back();
    return stk;
}
```

11.14 DelaunayTriangulation

```
// 德洛內三角剖分 (確保沒有點落在任何三角形的外接圓內)。
bool inCC(const array <Pt, 3> &p, Pt a) {
    i128 det = 0;
    for (int i = 0; i < 3; i++)
        det += i128(abs2(p[i]) - abs2(a)) * ori(a, p[(i + 1) %
            3], p[(i + 2) % 3]);
    return det > 0;
}
struct Edge {
    int id;
    list<Edge >::iterator rit;
};
vector <list<Edge>> Delaunay(const vector <Pt> &P) {
    assert(is_sorted(all(P))); // need sorted before!
    const int n = P.size();
    vector <list<Edge>> E(n);
    auto addEdge = [&](int u, int v, auto a, auto b) {
        a = E[u].insert(a, {v});
        b = E[v].insert(b, {u});
        return array{b->rit = a, a->rit = b};
    };
    auto divide = [&](auto &&self, int l, int r) -> int
    {
        if (r - l <= 1) return l;
        int m = (l + r) / 2;
        array <int, 2> t{self(self, l, m), self(self, m, r)};
        int w = t[P[t[1]].y < P[t[0]].y];
        auto low = [&](int s) {
            for (Edge e : E[t[s]]) {
                if (ori(P[t[2]], P[t[0]], P[e.id]) > 0
or
                    PtOnSeg(P[e.id], {P[t[0]], P[t[1]]})) {
                    t[s] = e.id;
                    return true;
                }
            }
            return false;
        };
        while (low(0) or low(1));
        array its = {addEdge(t[0], t[1], E[t[0]].begin(), E[t
            [1]].end())};
        while (true) {
            Line L{P[t[0]], P[t[1]]};
            auto cand = [&](int s) -> optional <list<
Edge >::iterator> {
                auto nxt = [&](auto it) {
                    if (s == 0) return (++it == E[t[0]].end() ?
                        E[t[0]].begin() :
it);
                    return --(it == E[t[1]].begin() ? E[t[1]].
                        end() : it);
                };
                if (E[t[s]].empty()) return {};
                auto lst = nxt(its[s]), it = nxt(lst);
                while (PtSide(P[it->id], L) > 0 and
                    inCC({P[a], P[b], P[lst->id]}, P[it->id]))
                    {
                    E[t[s ^ 1]].erase(lst->rit);
                    E[t[s]].erase(lst);
                    it = nxt(lst = it);
                }
                return PtSide(P[lst->id], L) > 0 ?
optional{lst} : nullopt;
            };
            auto lc = cand(0), rc = cand(1);
            if (!lc and !rc) break;
            int sd = !lc or (rc and inCC({L.a, L.b, P[(*lc)->id
                ]}, P[(*rc)->id]));
            auto lst = *(sd ? rc : lc);
            t[sd] = lst->id;
            its[sd] = lst->rit;
            its = addEdge(t[0], t[1], ++its[0], its[1]);
        }
        return w;
    };
    divide(divide , 0, n);
    return E;
}
;
```

11.15 PicksFormula

```
// 皮克定理：計算整數點多邊形的面積與內部/邊界點數關係。
// 給定頂點座標均是整點（或正方形格子點）的簡單多邊形（凸或
// 凹都可以）其面積 A 和內部格點數目 i、邊上格點數目 b 的關係：A=i+b/2-1
```

11.16 AreaOfCircleTriangle

```
// 計算圓與三角形重疊的面積。
double CircleTriangle(Pt a, Pt b, double r) {
    if (sgn(abs(a) - r) <= 0 and sgn(abs(b) - r) <= 0)
        return abs(a ^ b) / 2;
    if (abs(a) > abs(b)) swap(a, b);
    auto I = CircleLineInter({{}, r}, {a, b});
    erase_if(I, [&](Pt x) { return !PtOnSeg(x, {a, b}); });
    if (I.size() == 1) return abs(a ^ I[0]) / 2 +
    if (I.size() == 2) {
    }
}
```

11.17 ConvexHullTrick

```
// 凸多邊形的切線、點內判斷及交線查詢（或用於斜率優化）。
struct Convex {
    int n;
    vector <Pt> A, V, L, U;
    Convex(const vector <Pt> &_A) : A(_A), n(_A.size())
{ // n >= 3
        auto it = max_element(all(A));
        L.assign(A.begin(), it + 1);
        U.assign(it, A.end()), U.push_back(A[0]);
        for (int i = 0; i < n; i++) {
            V.push_back(A[(i + 1) % n] - A[i]);
        }
    }
    int inside(Pt p, const vector <Pt> &h, auto f) {
        auto it = lower_bound(all(h), p, f);
        if (it == h.end()) return 0;
        if (it == h.begin()) return p == it;
        return 1 - sgn(ori(*prev(it), p, *it));
    }
    // 0: out, 1: on, 2: in
    int inside(Pt p) {
        return min(inside(p, L, less{}), inside(p, U,
greater{}));
    }
    static bool cmp(Pt a, Pt b) { return sgn(a ^ b) > 0; }
    // A[i] is a far/closer tangent point
    int tangent(Pt v, bool close = true) {
        assert(v != Pt{});
        auto l = V.begin(), r = V.begin() + L.size() - 1;
        if (v < Pt{}) l = r, r = V.end();
        if (close) return (lower_bound(l, r, v, cmp) - V.begin
            ()) % n;
        return (upper_bound(l, r, v, cmp) - V.begin()) % n;
    }
    // closer tangent point
    array <int, 2> tangent2(Pt p) {
        array <int, 2> t{-1, -1};
        if (inside(p) == 2) return t;
        if (auto it = lower_bound(all(L), p); it != L.end() and
            p == *it) {
            int s = it - L.begin();
            return {(s + 1) % n, (s - 1 + n) % n};
        }
        if (auto it = lower_bound(all(U), p, greater{}); it !=
            U.end() and p == *it) {
```

```
            int s = it - U.begin() + L.size() - 1;
            return {(s + 1) % n, (s - 1 + n) % n};
        }
        for (int i = 0; i != t[0]; i = tangent((A[t[0] = i] - p
            ), 0));
        for (int i = 0; i != t[1]; i = tangent((p - A[t[1] = i
            ]), 1));
        return t;
    }
    int find(int l, int r, Line L) {
        if (r < l) r += n;
        int s = PtSide(A[l % n], L);
        return *ranges::partition_point(views::iota(l, r),
                                        [&](int m) {
                                            return PtSide(A[m %
                                                n], L) == s;
                                        }) - 1;
    }
    // Line A_x A_x+1 interset with L
    vector <int> intersect(Line L) {
        int l = tangent(L.a - L.b), r = tangent(L.b - L.a);
        if (PtSide(A[l], L) * PtSide(A[r], L) >= 0)
            return {};
        return {find(l, r, L) % n, find(r, l, L) % n};
    }
};
```

11.18 UnionOfCircles

```
// 計算多個圓的並集面積。
// Area[i] : area covered by at least i circle
vector <double > CircleUnion(const vector <Cir> &C) {
    const int n = C.size();
    vector <double > Area(n + 1);
    auto check = [&](int i, int j) {
        if (!contain(C[i], C[j]))
            return false;
        return sgn(C[i].r - C[j].r) > 0 or (sgn(C[i].r - C[j].r
            ) == 0 and i < j);
    };
    struct Teve {
        double ang; int add; Pt p;
        bool operator <(const Teve &b) { return ang < b.
ang; }
    };
    auto ang = [&](Pt p) { return atan2(p.y, p.x); };
    for (int i = 0; i < n; i++) {
        int cov = 1;
        vector <Teve> event;
        for (int j = 0; j < n; j++) if (i != j) {
                if (check(j, i)) cov++;
                else if (!check(i, j) and !disjunct(C[i], C
[j])) {
                    auto I = CircleInter(C[i], C[j]);
                    assert(I.size() == 2);
                    double a1 = ang(I[0] - C[i].o), a2 = ang(I
                        [1] - C[i].o);
                    event.push_back({a1, 1, I[0]});
                    event.push_back({a2, -1, I[1]});
                }
            }
        if (event.empty()) {
            Area[cov] += pi * C[i].r * C[i].r;
            continue;
        }
        sort(all(event));
        if (event[0].ang > event.back().ang) cov++;
        event.push_back(event[0]);
        for (int j = 0; j + 1 < event.size(); j++) {
            cov += event[j].add;
            Area[cov] += (event[j].p ^ event[j + 1].p) / 2.;
            double theta = event[j + 1].ang - event[j].ang;
            if (theta < 0) theta += 2 * pi;
            Area[cov] += (theta - sin(theta)) * C[i].r * C[i].r
                / 2.;
        }
    }
    for (int i = n - 1; i >= 0; i--) Area[i] += Area[i
+ 1];
    return Area;
}
```

11.19 TriangleCenter

```
// 計算三角形的外心、重心、垂心、內心。
Pt TriangleCircumCenter(Pt a, Pt b, Pt c) {
    Pt res;
    double a1 = atan2(b.y - a.y, b.x - a.x) + pi / 2;
    double a2 = atan2(c.y - b.y, c.x - b.x) + pi / 2;
    double ax = (a.x + b.x) / 2;
    double ay = (a.y + b.y) / 2;
    double bx = (c.x + b.x) / 2;
    double by = (c.y + b.y) / 2;
    double r1 = (sin(a2) * (ax - bx) - cos(a2) * (by - ay)) / (
        sin(a1) * cos(a2) - sin(a2) * cos(a1));
    return Pt(ax + r1 * cos(a1), ay + r1 * sin(a1));
}
Pt TriangleMassCenter(Pt a, Pt b, Pt c) {
    return (a + b + c) / 3.0;
}
Pt TriangleOrthoCenter(Pt a, Pt b, Pt c) {
    return TriangleMassCenter(a, b, c) * 3.0 -
TriangleCircumCenter(a, b, c) * 2.0;
}
Pt TriangleInnerCenter(Pt a, Pt b, Pt c) {
    Pt res;
    double la = abs(b - c);
    double lb = abs(a - c);
    double lc = abs(a - b);
    res.x = (la * a.x + lb * b.x + lc * c.x) / (la + lb + lc);
    res.y = (la * a.y + lb * b.y + lc * c.y) / (la + lb + lc);
    return res;
}
```

11.20 ConvexHull

```
// 計算點集的凸包 (Monotone Chain 算法)。
vector <Pt> convex(vector <Pt> v){
    vector <Pt> ret ;
    sort(v.begin(), v.end()) ;
    int k = 0 ;
    for(auto i:v){
        while(k >= 2 && ori(ret[k-1] - ret[k-2], i - ret[k-1])
            <= 0)
            ret.pop_back();
        ret.push_back(i) ;
        ++k ;
    }
    v.pop_back();
    int hf = --k ;
    reverse(v.begin(), v.end());
    for(auto i:v){
        while(k - hf >= 2 && ori(ret[k-1] - ret[k-2], i - ret[k
            -1]) <= 0)
            ret.pop_back();
        ret.push_back(i) ;
        ++k ;
    }
    return ret;
}
```

11.21 Line

```
// 定義線段結構，提供點線關係判斷及點投影。
struct Line {
    Pt a, b;
    Pt dir() const { return b - a; }
};
int PtSide(Pt p, Line L) {
    return sgn(ori(L.a, L.b, p)); // for int
    // return sgn(ori(L.a, L.b, p) / abs(L.a - L.b));
}
bool PtOnSeg(Pt p, Line L) {
    return PtSide(p, L) == 0 and sgn((p - L.a) * (p - L.b)) <=
        0;
}
Pt proj(Pt p, Line l) {
    Pt dir = unit(l.b - l.a);
    return l.a + dir * (dir * (p - l.a));
}
```

11.22 Point

```
// 點結構及基本幾何運算：加減乘除、向量長、旋轉、距離等。
using numbers::pi;
constexpr double eps = 1E-9L;
struct Pt {
    double x{}, y{};
};
Pt operator+(Pt a, Pt b) { return {a.x + b.x, a.y + b.y
}; }
Pt operator -(Pt a, Pt b) { return {a.x - b.x, a.y - b.y
}; }
Pt operator*(Pt a, double k) { return {a.x * k, a.y * k
}; }
Pt operator/(Pt a, double k) { return {a.x / k, a.y / k
}; }
```

ACM ICPC Team Reference - woowoowoo


```
double operator*(Pt a, Pt b) { return a.x * b.x + a.y *
b.y; }
double operator^(Pt a, Pt b) { return a.x * b.y - a.y *
b.x; }
auto operator <=>(Pt a, Pt b) { return pair{a.x, a.y}
<=> pair{b.x, b.y}; }
bool operator==(Pt a, Pt b) { return pair{a.x, a.y} ==
pair{b.x, b.y}; }
int sgn(double x) { return (x > -eps) - (x < eps); }
double abs(Pt a) { return sqrt(a * a); }
double abs2(Pt a) { return a * a; }
double ori(Pt a, Pt b, Pt c) { return (b - a) ^ (c - a)
; }
double arg(Pt x) { return atan2(x.y, x.x); }
bool argcmp(const Pt &a, const Pt &b) { // arg(a) < arg
(b)
    int f = (Pt{a.y, -a.x} > Pt{} ? 1 : -1) * (a != Pt
{});
    int g = (Pt{b.y, -b.x} > Pt{} ? 1 : -1) * (b != Pt
{});
    return f == g ? (a ^ b) > 0 : f < g;
}
Pt unit(Pt x) { return x / abs(x); }
Pt rotate(Pt u) { // pi / 2
    return {-u.y, u.x};
}
Pt rotate(Pt u, double a) {
    Pt v{sin(a), cos(a)};
    return {u ^ v, u * v};
}
```

11.23 dfcode

```
// 整數幾何基本運算：點、線定義，向量加減、內外積、點線段關係判
    斷。
using GType = ll;
using Point = pair<GType , GType >;
using Line = pair<Point , Point >;
Point operator+(const Point &a, const Point &b)
{
    return Point(a.F + b.F, a.S + b.S);
}
Point operator -(const Point &a, const Point &b)
{
    return Point(a.F - b.F, a.S - b.S);
}
Point operator*(const Point &a, const GType &b)
{
    return Point(a.F * b, a.S * b);
}
Point operator/(const Point &a, const GType &b)
{
    return Point(a.F / b, a.S / b);
}
GType dot(const Point &a, const Point &b)
{
    return a.F * b.F + a.S * b.S;
}
GType cross(const Point &a, const Point &b)
{
    return a.F * b.S - a.S * b.F;
}
GType ori(const Point &a, const Point &b, const Point &c)
{ // a->b x a->c
    GType res = cross(b - a, c - a);
    if (abs(res) < EPS)
        return 0;
    return res < 0 ? -1 : 1;
}
bool in_segment(Line a, Point b)
{
    return ori(a.F, a.S, b) == 0 && dot(a.F - b, a.S - b) <= 0;
}
bool banana(Line a, Line b)
{
    if (in_segment(a, b.F) || in_segment(a, b.S) ||
        in_segment(b, a.F) || in_segment(b, a.S))
        return true;
    return ori(a.F, a.S, b.F) * ori(a.F, a.S, b.S) < 0
&& ori(b.F, b.S, a.F) * ori(b.F, b.S, a.S) < 0;
}
```

11.24 MinkowskiSum

```
// 計算兩個凸多邊形的閔可夫斯基和。
// P, Q, R(return) are counterclockwise order convex
// polygon
vector <Pt> Minkowski(vector <Pt> P, vector <Pt> Q) {
    auto cmp = [&](Pt a, Pt b) {
        return Pt{a.y, a.x} < Pt{b.y, b.x};
    };
    auto reorder = [&](auto &R) {
        rotate(R.begin(), min_element(all(R), cmp), R.
end());
        R.push_back(R[0]), R.push_back(R[1]);
    };
    const int n = P.size(), m = Q.size();
    reorder(P), reorder(Q);
    vector <Pt> R;
    for (int i = 0, j = 0, s; i < n or j < m; ) {
        R.push_back(P[i] + Q[j]);
        s = sgn((P[i + 1] - P[i]) ^ (Q[j + 1] - Q[j]));
        if (s >= 0) i++;
        if (s <= 0) j++;
    }
    return R;
}
```

11.25 TangentLinesOfCirclePoint

```
// 計算從一個點到圓的所有切線。
vector <Line> CircleTangent(Cir c, Pt p) {
    vector <Line> z;
    double d = abs(p - c.o);
    if (sgn(d - c.r) == 0) {
        Pt i = rotate(p - c.o);
        z.push_back({p, p + i});
    } else if (d > c.r) {
        double o = acos(c.r / d);
        Pt i = unit(p - c.o);
        Pt j = rotate(i, o) * c.r;
        Pt k = rotate(i, -o) * c.r;
        z.push_back({c.o + j, p});
        z.push_back({c.o + k, p});
    }
    return z;
}
```

11.26 UnionOfPolygons

```
// 計算多個多邊形的並集面積。
// Area[i] : area covered by at least i polygon
vector <double > PolyUnion(const vector <vector <Pt>> &P) {
    const int n = P.size();
    vector <double > Area(n + 1);
    vector <Line> Ls;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < P[i].size(); j++)
            Ls.push_back({P[i][j], P[i][(j + 1) % P[i].size()
                ]});
    auto cmp = [&](Line &l, Line &r) {
        Pt u = l.b - l.a, v = r.b - r.a;
        if (argcmp(u, v)) return true;
        if (argcmp(v, u)) return false;
        return PtSide(l.a, r) < 0;
    };
    sort(all(Ls), cmp);
    for (int l = 0, r = 0; l < Ls.size(); l = r) {
        while (r < Ls.size() and !cmp(Ls[l], Ls[r])) r++;
        Line L = Ls[l];
        vector <pair<Pt, int>> event;
        for (auto [c, d] : Ls) {
            if (sgn((L.a - L.b) ^ (c - d)) != 0) {
                int s1 = PtSide(c, L) == 1;
                int s2 = PtSide(d, L) == 1;
                if (s1 ^ s2) event.emplace_back(
                    LineInter(L, {c, d}), s1 ? 1 : -1);
            } else if (PtSide(c, L) == 0 and sgn((L.a -
L.b) * (c - d)) > 0) {
                event.emplace_back(c, 2);
                event.emplace_back(d, -2);
            }
        }
        sort(all(event), [&](auto i, auto j) {
            return (L.a - i.ff) * (L.a - L.b) < (L.a - j.ff) *
                (L.a - L.b);
        });
        int cov = 0, tag = 0;
        Pt lst{0, 0};
        for (auto [p, s] : event) {
            if (cov >= tag) {
                Area[cov] += lst ^ p;
                Area[cov - tag] -= lst ^ p;
            }
            if (abs(s) == 1) cov += s;
            else tag += s / 2;
            lst = p;
        }
    }
    for (int i = n - 1; i >= 0; i--) Area[i] += Area[i
```

ACM ICPC Judge Test - WOOWOOWOO

C++ Resource Test

```cpp
#include <bits/stdc++.h>
using namespace std;

namespace system_test {

const size_t KB = 1024;
const size_t MB = KB * 1024;
const size_t GB = MB * 1024;

size_t block_size, bound;
void stack_size_dfs(size_t depth = 1) {
  if (depth >= bound)
    return;
  int8_t ptr[block_size]; // 若無法編譯將 block_size 改成常數
  memset(ptr, '@', block_size);
  cout << depth << endl;
  stack_size_dfs(depth + 1);
}

void stack_size_and_runtime_error(size_t block_size, size_t
    bound = 1024) {
  system_test::block_size = block_size;
  system_test::bound = bound;
  stack_size_dfs();
}

double speed(int iter_num) {
  const int block_size = 1024;
  volatile int A[block_size];
  auto begin = chrono::high_resolution_clock::now();
  while (iter_num--)
    for (int j = 0; j < block_size; ++j)
      A[j] += j;
  auto end = chrono::high_resolution_clock::now();
  chrono::duration<double> diff = end - begin;
  return diff.count();
}

void runtime_error_1() {
  // Segmentation fault
  int *ptr = nullptr;
  *(ptr + 7122) = 7122;
}

void runtime_error_2() {
  // Segmentation fault
  int *ptr = (int *)memset;
  *ptr = 7122;
}

void runtime_error_3() {
  // munmap_chunk(): invalid pointer
  int *ptr = (int *)memset;
  delete ptr;
}

void runtime_error_4() {
  // free(): invalid pointer
  int *ptr = new int[7122];
  ptr += 1;
  delete[] ptr;
}

void runtime_error_5() {
  // maybe illegal instruction
  int a = 7122, b = 0;
  cout << (a / b) << endl;
}

void runtime_error_6() {
  // floating point exception
  volatile int a = 7122, b = 0;
  cout << (a / b) << endl;
}

void runtime_error_7() {
  // call to abort.
  assert(false);
}

} // namespace system_test

#include <sys/resource.h>
void print_stack_limit() { // only work in Linux
  struct rlimit l;
  getrlimit(RLIMIT_STACK, &l);
  cout << "stack_size = " << l.rlim_cur << " byte" << endl;
}
```

```
    cout << "最小堆大小 (pq_min.size()): " << pq_min.size() <<
        endl;   // 取得堆中元素數量，結果 3

    pq_min.pop(); // pop(): 移除堆頂元素（最小的，不返回任何值）
    cout << "pop 後最小堆頂部: " << pq_min.top() << endl;
                // 移除 10 後，結果 20
    cout << "最小堆是否為空 (pq_min.empty()): " << (pq_min.
        empty() ? "是" : "否") << endl; // 判斷是否為空

    return 0;
}
```

4 BFS & DFS

4.1 DFS

```
// Looking for cycle in directed graph
void solve() {
    int n, m;
    cin >> n >> m;

    vector < vector < int > > G(n);
    int v, u;
    FOR(i, 0, m){
        cin >> v >> u;
        G[v-1].PB(u);
    }

    bool ans = false;
    int start = -1;
    int end = -1;
    vector < int > visit(n, 0);
    vector < int > parent(n, -1);
    auto dfs = [&](auto&& self, int x) -> bool {
        visit[x-1] = 1;

        for(int y: G[x-1]){
            if(visit[y-1]==1){
                start = y;
                end = x;
                return true;
            }
            if(visit[y-1]==0){
                parent[y-1] = x;
                if(self(self, y)){
                    return true;
                }
            }
        }

        visit[x-1] = 2;
        return false;
    };

    FOR(i, 1, n+1){
        if(visit[i-1]==0){
            if(dfs(dfs, i)){
                ans = true;
                break;
            }
        }
    }

    if(ans){
        cout << "YES" << endl;

        int x = end;
        vector < int > cycle;
        while(x!=start){
            cycle.PB(x);
            x = parent[x-1];
        }
        cycle.PB(x);
        reverse(ALL(cycle));
        print(cycle);
    }else{
        cout << "NO" << endl;
    }
}
```

4.2 BFS

```
// Bipartite graph check using BFS
void solve() {
    int n, m;
    cin >> n >> m;
    int u, v;
    vector < vector < int > > G(n+1);
    FOR(i, 0, m){
        cin >> u >> v;
        G[u].PB(v);
        G[v].PB(u);
    }

    bool ans = true;
    vector < int > color(n+1, 0);
    FOR(i, 1, n+1){
        if(color[i]!=0) continue;
        queue < int > q;
        color[i] = 1;
        q.push(i);
        while(!q.empty()){
            int t = q.front();
            q.pop();
            for(int y: G[t]){
                if(color[y]==0){
                    color[y] = 3 - color[t];
                    q.push(y);
                }else if(color[y]==color[t]){
                    ans = false;
                    break;
                }
            }
            if(!ans) break;
        }
        if(!ans) break;
    }

    if(ans){
        cout << "YES" << endl;
        FOR(i, 1, n+1){
            cout << color[i] << ' ';
        }
    }else{
        cout << "NO" << endl;
    }
}
```

5 Binary Search

5.1 upper bound

```
// 用 upper_bound() 找第一個大於目標 target=d[i]+r 的元素
void solve() {
    int n, r;
    cin >> n >> r;

    vector<int>d(n);
    FOR(i, 0, n) cin >> d[i];

    int ans = 0;
    FOR(i, 0, n){
        auto it_upper = upper_bound(ALL(d), d[i]+r);
        ans += d.end()-it_upper;
    }

    cout << ans << endl;
}
```

6 Decimal Precision

6.1 setprecision

```
#include <iostream>
#include <iomanip> // 引入 iomanip 標頭檔
// 包含在 <bits/stdc++.h> 中，不需要額外引入

int main() {
    double pi = 3.141596;
    double data = 123.456789;

    // 輸出 pi 精確到小數點後 7 位
    std::cout << std::fixed << std::setprecision(7) << pi <<
        std::endl; // 輸出 3.1415960

    // 輸出 data 精確到小數點後 2 位
    std::cout << std::fixed << std::setprecision(2) << data <<
        std::endl; // 輸出 123.46 (四捨五入)

    return 0;
}
```

7 Dynamic Programming

7.1 Edit Distance

```
// named as "edit distance" or "Levenshtein distance"
void solve() {
    string s1, s2;
    cin >> s1 >> s2;

    int l1 = s1.size();
    int l2 = s2.size();

    vector < vector < int > > dp(l1+1, vector < int > (l2+1, 0)
        );
    FOR(i, 0, l1+1) dp[i][l2] = l1-i;
    FOR(j, 0, l2+1) dp[l1][j] = l2-j;

    for(int i=l1-1; i>=0; i--){
        for(int j=l2-1; j>=0; j--){
            if(s1[i]==s2[j]){
                dp[i][j] = dp[i+1][j+1];
            }else{
                int val = min(dp[i+1][j], dp[i][j+1]);
                val = min(val, dp[i+1][j+1]);
                dp[i][j] = val + 1;
            }
        }
    }

    cout << dp[0][0] << endl;
}
```

7.2 Longest Increasing Subsequence

```
// 最長遞增子序列 (Longest Increasing Subsequence, LIS)
// O(n Log n) 解法
void solve() {
    int n;
    cin >> n;
    vector < int > x(n);
    for(int i=0; i<n; i++){
        cin >> x[i];
    }

    vector < int > arr;
    int m = 0;
    for(int i=0; i<n; i++){
        if(m==0){
            arr.push_back(x[i]);
            m++;
        }else if(x[i] > arr[m-1]){
            arr.push_back(x[i]);
            m++;
        }else{
            int l = 0;
            int r = m-1;
            while(l<=r){
                int mid = l + (r - l) / 2;
                if(arr[mid] < x[i]){
                    l = mid + 1;
                }else{
                    r = mid - 1;
                }
            }
            arr[l] = x[i];
        }
    }
    cout << arr.size();
}
```

7.3 Longest Common Subsequence

```
// 最長共同子序列 (Longest Common Subsequence, LCS)
// O(nm) 解法
void solve() {
    int n, m;
    cin >> n >> m;

    vector < int > arr(n);
    FOR(i, 0, n) cin >> arr[i];
    vector < int > brr(m);
    FOR(i, 0, m) cin >> brr[i];

    vector < vector < int > > dp(n+1, vector < int > (m+1, 0));
    FOR(i, 0, n){
        FOR(j, 0, m){
            if(arr[i]==brr[j]){
                dp[i+1][j+1] = dp[i][j] + 1;
            }else{
                dp[i+1][j+1] = max(dp[i+1][j], dp[i][j+1]);
            }
        }
    }

    stack < int > ans;
    int x = n, y = m;
    while(!(x==0 or y==0)){
        if(dp[x][y]>dp[x-1][y] and dp[x][y]>dp[x][y-1]){
            ans.push(arr[x-1]);
            x--;
            y--;
        }else if(dp[x][y]>dp[x-1][y]){
            y--;
        }else{
            x--;
        }
    }
    while(!ans.empty()){
        cout << ans.top() << ' ';
        ans.pop();
    }
}
```

8 Math Theorems

8.1 Picks Theorem

```
// 計算所有頂點在整數格點上的簡單多邊形的面積
// 使用條件：
//   - 多邊形不自交、頂點順時針或逆時針排列皆可
//   - 頂點座標皆為整數
//
// 面積公式：Area = I + B / 2 - 1
//   I: 多邊形內部的整數格點數
//   B: 多邊形邊界上的整數格點數
//
// 使用的方法：
// vector<Point> poly = {
//     {0, 0}, {4, 0}, {4, 3}, {0, 3}  // <-- 修改成題目給的的點
// };
// ll I = interior_points(poly);      // 內部整點數
// ll B = boundary_points(poly);      // 邊界整點數
// double area = I + B / 2.0 - 1;     // 多邊形面積（浮點數）

typedef long long ll;
ll gcd(ll a, ll b) { return b ? gcd(b, a % b) : abs(a); }

struct Point {
    ll x, y;
    Point(ll x = 0, ll y = 0) : x(x), y(y) {}
};

// 計算兩倍的面積（避免使用浮點數）
ll double_area(const vector<Point>& poly) {
    ll A = 0;
    int n = poly.size();
    for (int i = 0; i < n; ++i) {
        Point a = poly[i], b = poly[(i + 1) % n];
        A += (a.x * b.y - b.x * a.y);
    }
    return abs(A);  // 回傳的是 2 * 面積
}

// 計算邊界上的整數格點數
ll boundary_points(const vector<Point>& poly) {
    ll B = 0;
    int n = poly.size();
    for (int i = 0; i < n; ++i) {
        Point a = poly[i], b = poly[(i + 1) % n];
        B += gcd(abs(a.x - b.x), abs(a.y - b.y));
    }
    return B;
}

// 計算內部整數格點數
ll interior_points(const vector<Point>& poly) {
    ll A2 = double_area(poly);  // 2 * area
    ll B = boundary_points(poly);
    return (A2 - B + 2) / 2;
}
```

9 Python

9.1 string

```python
# --- 字符串建立與基本操作 ---
s = 'This is a book.' # 單行字符串範例
s_multi_line = "Hello\nWorld" # 多行字符串，使用 \n 換行符
s_raw = r"C:\Users\Name" # 原始字符串，不解釋反斜線為轉義字符
print(f"原始字符串範例: {s_raw}") # C:\Users\Name

s_comp_1 = "Hello"
s_comp_2 = "World"

# 字符串連接 (使用 + 運算符)
s_combined = s_comp_1 + " " + s_comp_2
print(f"字符串連接: {s_combined}") # Hello World

# 字符串重複 (使用 * 運算符)
s_repeated = "abc" * 3
print(f"字符串重複: {s_repeated}") # abcabcabc

# 取得字符串長度 (使用 len() 函數)
print(f"字符串 '{s}' 的長度: {len(s)}") # 15

# 字符串索引 (從 0 開始，負數索引從末尾算起)
print(f"第一個字符: {s[0]}") # T
print(f"最後一個字符: {s[-1]}") # .

# 字符串切片 (與列表切片類似，[start:end:step]，end 不包含)
print(f"索引 5 到 7: {s[5:8]}") # is
print(f"從開頭到索引 4 (不含): {s[:4]}") # This
print(f"索引 10 到末: {s[10:]}") # book.
print(f"反轉字符串: {s[::-1]}") # .koob a si sihT

# 字符串成員判斷 (使用 in / not in 運算符)
print(f"'is' 在 '{s}' 中嗎? {'is' in s}") # True
print(f"'xyz' 不在 '{s}' 中嗎? {'xyz' not in s}") # True

# --- 字符串方法 ---
s = 'This is a book.' # 範例字符串
s2 ='apple ,banana ,cat,dog' # 範例字符串2
s3 ='Hello*&' # 範例字符串3
s4 = 'Python123' # 範例字符串4

# 查找
print(f"查找 'is' 的索引: {s.find('is')}") # 2 (回傳第一個找到的子字符串索引)
print(f"查找 'xyz' 的索引: {s.find('xyz')}") # -1 (未找到返回 -1)

# 替換
print(f"替換 'This' 為 'That': {s.replace('This', 'That')}") # 產生新字符串 That is a book.

# 大小寫轉換
print(f"轉為大寫: {s.upper()}") # THIS IS A BOOK.
print(f"轉為小寫: {s.lower()}") # this is a book.
print(f"每個單詞開頭大寫: {s.title()}") # This Is A Book.
print(f"第一個字母大寫: {s.capitalize()}") # This is a book.

# 分割字符串
print(f"使用 ',' 分割: {s2.split(',')}") # ['apple', 'banana ', 'cat', 'dog']
# 注意：如果 s2 是 'apple,banana,cat,dog' (無空格)，應該用 s2.split(',')

# 合併字符串 (非常重要，將列表中的字符串用指定分隔符連接起來)
words_list = ['apple', 'banana', 'cherry']
print(f"使用 ',' 連接列表: {','.join(words_list)}") # apple,banana,cherry
nums_list = [1, 2, 3]
print(f"使用 ' ' 連接數字列表: {' '.join(map(str, nums_list))}") # 1 2 3

# 清除頭尾字符
print(f"移除頭尾的 '*&': {s3.strip('*&')}") # Hello

# 字符串判斷 (回傳布林值 True 或 False)
# function 皆有添加 is，回傳布林
print(f"是否全為大寫: {s.isupper()}") # False
print(f"是否全為小寫: {s.islower()}") # False
print(f"是否每個單詞開頭大寫: {s.istitle()}") # False (因為 'is' 是小寫)
print(f"是否為數字: {s4.isdigit()}") # False
print(f"是否只含數字或字母: {s4.isalnum()}") # True
print(f"是否為字母: {s4.isalpha()}") # False
print(f"字符 'o' 的出現次數: {s.count('o')}") # 2
```

9.2 string methods

```python
s = 'This is a book.' # 範例字符串
s2 ='apple ,banana ,cat,dog' # 範例字符串2
s3 ='Hello*&' # 範例字符串3
s4 = 'Python123' # 範例字符串4

# 字符串查找與替換
print(s.find('is')) # 2 (回傳第一個找到的子字符串索引)
print(s.replace('This', 'That')) # 產生新字符串 That is a book. (替換子字符串)

# 字符串分割
print(s2.split(', ')) # ['apple', 'banana', 'cat,dog'] (根據分隔符分割字符串為列表)

# 字符串修剪
print(s3.strip('*&')) # Hello (修掉字符串頭尾指定的字符)

# 字符串大小寫轉換
print(s.upper()) # THIS IS A BOOK. (轉為大寫)

# 字符串判斷 (回傳布林值 True 或 False)
# function 皆有添加 is，回傳布林
print(s.capitalize()) # This is a book. (將字符串第一個字母轉大寫，其他轉小寫)
print(s.iscapitalize()) # False (檢查第一個字母是否大寫，且其他字母是否小寫，這裏s中有大寫IS)
print(s.count('o')) # 2 (計算指定字符出現的次數)
print(s.title()) # This Is A Book. (將每個單詞的首字母轉大寫)
print(s.istitle()) # False (檢查每個單詞開頭是否都大寫)
print(s2.islower()) # False (檢查是否皆為小寫，因為有大寫字母)
print(s4.isdigit()) # False (檢查是否皆為數字)
print(s4.isalnum()) # True (檢查是否只含數字或字母)
print(s4.isalpha()) # False (檢查是否皆為字母)
```

9.3 EOF input

```python
while True:
    try:
        line = input() # 嘗試從標準輸入讀取一行
        print(line.strip().upper()) # 範例：將每行內容轉為大寫並輸出
        numbers = list(map(int, line.split())) # 範例：如果一行有多個數字，讀取為列表
        print(numbers)
    except:
        break
```

9.4 structure set

```python
# --- 建立集合 ---
# 集合是無序且元素不重複的數據結構，常用於去重、判斷關聯性。

a = set() # 建立一個空集合
print(f"空集合: {a}") # set()

# 從列表建立集合 (自動去重)
b = set([1, 2, 3, 4, 5, 1, 2, 3])
print(f"從列表建立: {b}") # {1, 2, 3, 4, 5}

# 從字符串建立集合 (自動去重，並將字符打散)
d = set('hello')
print(f"從字符串建立: {d}") # {'l', 'o', 'h', 'e'} (順序不固定)

# 使用大括號 {} 直接建立集合 (如果不是空集合)
# 注意：{} 單獨使用會建立空字典，而非空集合。
e = {0, 1, 'a', False} # False 等同於 0，只保留 0
print(f"使用大括號建立: {e}") # {0, 1, 'a'} (順序不固定)

# --- 加入與移除項目 ---
my_set = {1, 2, 3}
my_set.add(4) # 加入一個項目
my_set.add(2) # 加入重複項目不會有影響
print(f"加入後: {my_set}") # {1, 2, 3, 4}

my_set.remove(2) # 移除項目 (若項目不存在會報錯 KeyError)
print(f"移除 2 後: {my_set}") # {1, 3, 4}
```

```

my_set.discard(100) # 移除項目 (若項目不存在不會報錯)
print(f"移除 100 後 (不存在): {my_set}") # {1, 3, 4}

# --- 集合運算 (交集、聯集、差集、對稱差集) ---
set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 5, 6, 7}

# 交集 (共同的元素)
print(f"交集 (方法): {set1.intersection(set2)}") # {3, 4, 5}
print(f"交集 (符號): {set1 & set2}") # {3, 4, 5}

# 聯集 (所有不重複的元素)
print(f"聯集 (方法): {set1.union(set2)}") # {1, 2, 3, 4, 5, 6, 7}
print(f"聯集 (符號): {set1 | set2}") # {1, 2, 3, 4, 5, 6, 7}

# 差集 (在 set1 中，但不在 set2 中的元素)
print(f"差集 (方法): {set1.difference(set2)}") # {1, 2}
print(f"差集 (符號): {set1 - set2}") # {1, 2}

# 對稱差集 (在 set1 或 set2 中，但不同時在兩者中的元素)
print(f"對稱差集 (方法): {set1.symmetric_difference(set2)}") # {1, 2, 6, 7}
print(f"對稱差集 (符號): {set1 ^ set2}") # {1, 2, 6, 7}

# --- 子集合、超集合 ---
set_A = {1, 2, 3, 4, 5}
set_B = {2, 3, 4}
set_C = {1, 2, 3, 4, 5}
set_D = {5, 6, 7}

# issubset() 檢查是否為子集合 (B 是否包含於 A)
print(f"B 是 A 的子集合嗎? {set_B.issubset(set_A)}") # True
print(f"A 是 B 的子集合嗎? {set_A.issubset(set_B)}") # False

# issuperset() 檢查是否為超集合 (A 是否包含 B)
print(f"A 是 B 的超集合嗎? {set_A.issuperset(set_B)}") # True

# 比較運算符
print(f"B <= A (B是A的子集合): {set_B <= set_A}") # True
print(f"B < A (B是A的真子集合): {set_B < set_A}") # True (A 包含 B 且 A 有 B 沒有的元素)
print(f"A >= B (A是B的超集合): {set_A >= set_B}") # True
print(f"A > B (A是B的真超集合): {set_A > set_B}") # True
print(f"A <= C (A是C的子集合): {set_A <= set_C}") # True (兩者相同也算子集合)
print(f"A < C (A是C的真子集合): {set_A < set_C}") # False (因為兩者完全相同)
print(f"D <= A (D是A的子集合): {set_D <= set_A}") # False

# --- len() 取得長度 ---
print(f"集合 {set_A} 的長度: {len(set_A)}") # 5

# --- in 檢查是否存在 ---
print(f"3 在集合 {set_A} 中嗎? {3 in set_A}") # True
print(f"100 不在集合 {set_A} 中嗎? {100 not in set_A}") # True
```

9.5 structures common

```
from collections import deque, defaultdict, Counter # 從 collections 模組導入常用結構
import heapq # 導入 heapq 模組，用於優先級隊列 (最小堆)

# --- 1. 元組 (Tuple) ---
# 元組是有序、不可變、可包含重複元素的序列。
# 通常用於當你不想讓數據被修改時，或作為字典的鍵/集合的元素。

my_tuple = (1, 2, 'a', 3)
print(f"元組範例: {my_tuple}") # (1, 2, 'a', 3)
print(f"訪問元組元素: {my_tuple[0]}") # 1
# my_tuple[0] = 10 # 錯誤：元組不可變

# --- 2. deque (雙端隊列) ---
# `deque` (double-ended queue) 是一個列表優化，用於在兩端快速添加和刪除元素。
# 在兩端進行 `append()` 和 `pop()` 操作時，`deque` 的時間複雜度是 O(1)。
# 適用於 BFS (廣度優先搜索)、滑動窗口等問題。

my_deque = deque()
my_deque.append(1) # 在右端添加
my_deque.append(2)
my_deque.appendleft(0) # 在左端添加
print(f"Deque 添加後: {my_deque}") # deque([0, 1, 2])

right_item = my_deque.pop() # 從右端移除並返回
left_item = my_deque.popleft() # 從左端移除並返回
print(f"Deque 移除後: {my_deque}, 移出右端: {right_item}, 移出左端: {left_item}") # deque([1]), 2, 0

# --- 3. defaultdict (默認字典) ---
# `defaultdict` 是 `dict` 的子類，當你訪問一個不存在的鍵時，會自動創建該鍵並賦予預設的默認值。
# 非常適合用於計數、分組、建立鄰接列表 (圖) 等場景，避免 KeyError。

# 以 int 作為默認值的 defaultdict (常用於計數)
counts = defaultdict(int)
words = ["apple", "banana", "apple"]
for word in words:
    counts[word] += 1 # 如果 'apple' 不存在，會自動初始化為 0 再加 1
print(f"Defaultdict 計數: {counts}") # defaultdict(<class 'int'>, {'apple': 2, 'banana': 1})
print(f"訪問不存在的鍵 'grape': {counts['grape']}") # 自動創建 'grape': 0
print(f"字典更新後: {counts}") # defaultdict(<class 'int'>, {'apple': 2, 'banana': 1, 'grape': 0})

# 以 list 作為默認值的 defaultdict (常用於建立圖的鄰接列表)
graph = defaultdict(list)
edges = [(1, 2), (1, 3), (2, 4)]
for u, v in edges:
    graph[u].append(v) # 如果鍵 u 不存在，自動初始化為 [] 再 append
print(f"Defaultdict 鄰接列表: {graph}") # defaultdict(<class 'list'>, {1: [2, 3], 2: [4]})

# --- 4. Counter (計數器) ---
# `Counter` 是 `dict` 的子類，專門用於計數可哈希對象。

data = ['a', 'b', 'c', 'a', 'b', 'a']
char_counts = Counter(data)
print(f"Counter 計數: {char_counts}") # Counter({'a': 3, 'b': 2, 'c': 1})

print(f"'a' 的數量: {char_counts['a']}") # 3
print(f"'z' 的數量 (不存在返回 0): {char_counts['z']}") # 0

print(f"最常見的 2 個元素: {char_counts.most_common(2)}") # [('a', 3), ('b', 2)]

# --- 5. heapq (堆隊列 / 優先級隊列) ---
# `heapq` 模組實現了最小堆 (min-heap) 算法。
# 堆是二叉樹的一種，所有父節點的值都小於或等於其子節點的值。
# 在競賽中常用於實現優先級隊列 (最小堆)，例如 Dijkstra 算法。

min_heap = [] # 創建一個空列表作為堆
heapq.heappush(min_heap, 3) # 將元素推入堆中 (保持堆屬性)
heapq.heappush(min_heap, 1)
heapq.heappush(min_heap, 4)
print(f"堆元素: {min_heap}") # [1, 3, 4] (列表表示，不保證所有父子關係，但最小元素在索引 0)

min_element = heapq.heappop(min_heap) # 彈出堆中最小的元素
print(f"彈出最小元素: {min_element}, 堆: {min_heap}") # 1, [3, 4] (或 [3,4] 視內部結構而定)

# 堆化一個現有列表
unheap_list = [5, 2, 8, 1, 9]
heapq.heapify(unheap_list) # 將列表轉換為堆
print(f"堆化列表: {unheap_list}") # [1, 2, 8, 5, 9] (最小元素在索引 0)
```

9.6 float error

```
import math # 導入 math 模組

# 浮點數精度問題範例
a = 0.1 + 0.2
b = 0.3
print(a == b) # False (由於浮點數表示問題，a不完全等於0.3)

# 浮點數比較：使用容忍度 (EPSILON)
EPSILON = 1e-9 # 定義一個很小的容忍值
print(abs(a - b) < EPSILON) # True (如果兩數差距小於EPSILON，視為相等)

# 浮點數比較：使用 math.isclose() (Python 3.5+ 推薦)
# rel_tol 是相對容忍度，abs_tol 是絕對容忍度
print(math.isclose(a, b, rel_tol=1e-9)) # True (更穩健的比較方法)
```