一、语法和常用函数

```
"""语法糖和常用函数"""
print(bin(9)) #bin函数返回二进制，形式为0b1001
dict.items()#同时调用key和value
print(round(3.123456789,5)# 3.12346
print("{:.2f}".format(3.146)) # 3.15
a,b=b,a
dict.get(key,default=None) # 其中，my_dict是要操作的字典，key是要查找的键，default是可选参数，表示当指定的键不存在时要返回的默认值
ord() # 字符转ASCII
chr() # ASCII转字符
for index,value in enumerate([a,b,c]): # 每个循环体里把索引和值分别赋给index和value。如第一次循环中index=0,value="a"
b=c.swapcase()#返回一个新字符串，其中所有的字母大小互换
if char.islower():
      char.upper()#手动进行大小写转换
if a.isdigit()#判断是不是只有数字
 #eval的用法：用于字符串计算
result = eval("2 + 3 * 4")
print(result)  # 输出 14

map(int, "110")` #会将字符串`"110"`中的每个字符 '1', '1'`, '0'` 转换成整数，得到 `[1, 1, 0]`。
```

```
# 二进制转十进制
binary_str = "1010"
decimal_num = int(binary_str, 2) # 第一个参数是字符串类型的某进制数，第二个参数是他的进制，最终转化为整数
print(decimal_num)  # 输出 10
#正则表达式搜索
import re
s = input()
r = re.search('h.*e.*l.*l.*o', s)#在字符串s中是不是出现了完整的hello
print(['YES', 'NO'][r == None])
#打印列表
print(*c[i])` #打印这些解包后的元素，并且默认情况下它们之间用空格分隔。

while True:
   try:
   except EOFError:

n.replace(n[b], "")``
str=str.replace()` #方法不会修改原字符串，而是返回一个新的字符串。你需要将结果重新赋值给 `n`

#enumerate
my_list = ['apple', 'banana', 'cherry']    #Index: 0, Value: apple
for index, value in enumerate(my_list):
    print(f"Index: {index}, Value: {value}")
 for index, value in enumerate(my_list, start=1):  # 索引从 1 开始
    print(f"Index: {index}, Value: {value}")
```

```
any(x)#判断x对象是否为空对象，如果都为空、0、false，则返回false，如果不都为空、0、false，则返回true

all(x)#如果all(x)参数x对象的所有元素不为0、''、False或者x为空对象，则返回True，否则返回False

n = int(input())
print('NO' if all([n%i for i in (4,7,47,74,447,474,477,747,774)]) else 'YES')
```

二、工具

1. **质数筛（埃氏筛）：

```
# 胡睿诚 23数院
# 埃氏筛 基本够用
N=20
primes = []
is_prime = [True]*N
is_prime[0] = False;is_prime[1] = False
for i in range(1,N):
    if is_prime[i]:
        primes.append(i)
        for k in range(2*i,N,i): #用素数去筛掉它的倍数
            is_prime[k] = False
print(primes)
# [2, 3, 5, 7, 11, 13, 17, 19]


# 使用埃拉托斯特尼筛法来计算所有小于等于 10^6 的质数
def sieve(limit):
    is_prime = [True] * (limit + 1)
    is_prime[0] = is_prime[1] = False  # 0 和 1 不是质数
    for i in range(2, int(math.sqrt(limit)) + 1):
        if is_prime[i]:
            for j in range(i * i, limit + 1, i):
                is_prime[j] = False
    return is_prime

# 创建一个用于判断某数是否为质数的布尔列表
limit = 10**6
is_prime_list = sieve(limit)
```

2. 简单题可以多循环（提醒）

```
x=int(input())
cube=[i**3 for i in range(x+1)]
for a in range(3,x+1):
    for b in range(2,a):
        for c in range(b,a):
            for d in range(c,a):
                if cube[a] ==cube[b]+cube[c]+cube[d]:
                    print("Cube = "+str(a)+", Triple =
("+str(b)+","+str(c)+","+str(d)+")")
```

3. 在改写后变成了零碎的东西

(1) math

```
import math
print(math.ceil(1.5)) # 2 还有floor
print(math.pow(2,3)) # 8.0
print(math.pow(2,2.5)) # 5.656854249492381
print(9999999>math.inf) # False
print(math.sqrt(4)) # 2.0
print(math.log(100,10)) # 2.0  math.log(x,base) 以base为底，x的对数
print(math.comb(5,3)) # 组合数，C53
print(math.factorial(5)) # 5!
math.gcd(x, y)#返回 `x` 和 `y` 的最大公约数。
print（math.gcd(12, 8)`#`4`
```

math函数做出来的是float，有时候需要转成int

(2) lru_cache

```
from functools import lru_cache
@lru_cache(maxsize=None)#None不行就试1024、2048什么的
```

(3) 二分查找 bisect 和 传统 动态时仍需要运用传统二分查找

```
import bisect
sorted_list = [1,3,5,7,9] #[(0)1, (1)3, (2)5, (3)7, (4)9]
position = bisect.bisect_left(sorted_list, 6)
print(position)  # 输出：3，因为6应该插入到位置3，才能保持列表的升序顺序

bisect.insort_left(sorted_list, 6)
print(sorted_list)  # 输出：[1, 3, 5, 6, 7, 9]，6被插入到适当的位置以保持升序顺序

sorted_list=(1,3,5,7,7,7,9)
print(bisect.bisect_left(sorted_list,7))
print(bisect.bisect_right(sorted_list,7))
# 输出：3 6
```

```python
#求在一条河流中移除最少的石头以保证两块石头之间的最小距离尽可能大
L, n, m = map(int, input().split())  # L: 河流长度, n: 石头数量, m: 最多允许移除的石头数量
rock = [0]  # 起点位置
for i in range(n):
    rock.append(int(input()))  # 逐个输入石头的位置
rock.append(L)  # 终点位置
def check(x):
    num = 0  # 统计需要移除的石头数量
    now = 0  # 当前所在石头的位置
    for i in range(1, n + 2):  # 遍历每块石头
        if rock[i] - now < x:  # 如果两石头间的距离小于 x，则移除当前石头
            num += 1
        else:  # 如果距离大于等于 x，更新当前所在的位置
            now = rock[i]
    return num > m  # 如果移除的石头数量超过 m，则返回 True（此 x 不可行）
lo, hi = 0, L + 1  # 初始二分查找的范围
ans = -1  # 用于记录最终结果
while lo < hi:
    mid = (lo + hi) // 2  # 取中间值作为当前猜测的最小距离

    if check(mid):  # 如果当前距离不可行（需要移除的石头超过限制）
        hi = mid  # 缩小范围到左半部分
    else:  # 如果当前距离可行
        ans = mid  # 更新答案为当前的 mid
        lo = mid + 1  # 缩小范围到右半部分
```

(4) 年份calendar包

```python
import calendar
print(calendar.isleap(2020)) # True
```

(5) heapq 优先队列

```python
import heapq # 优先队列可以实现以log复杂度拿出最小（大）元素
lst=[1,2,3]
heapq.heapify(lst) # 将lst优先队列化
heapq.heappop(lst) # 从队列中弹出树顶元素，指的是最小的那个数，即最左侧（默认最小，相反数调转）
heapq.heappush(lst,element) # 把元素压入堆中
```

(6) Counter包

```python
from collections import Counter
# O(n)
# 创建一个待统计的列表
data = ['apple', 'banana', 'apple', 'orange', 'banana', 'apple']
# 使用Counter统计元素出现次数
counter_result = Counter(data) # 返回一个字典类型的东西
# 输出统计结果
print(counter_result) # Counter({'apple': 3, 'banana': 2, 'orange': 1})
print(counter_result["apple"]) # 3
```

(7) default_dict

```
from collections import defaultdict
# 创建一个defaultdict，值的默认工厂函数为int，表示默认值为0
char_count = defaultdict(int)
# 统计字符出现次数
input_string = "hello"
for char in input_string:
    char_count[char] += 1
print(char_count)  # 输出 defaultdict(<class 'int'>, {'h': 1, 'e': 1, 'l': 2, 'o': 1})）
```

(8) itertools包 (排列组合等)

```
import itertools
my_list = ['a', 'b', 'c']
permutation_list1 = list(itertools.permutations(my_list))
permutation_list2 = list(itertools.permutations(my_list, 2))
combination_list = list(itertools.combinations(my_list, 2))
bit_combinations = list(itertools.product([0, 1], repeat=4))

print(permutation_list1)
# [('a', 'b', 'c'), ('a', 'c', 'b'), ('b', 'a', 'c'), ('b', 'c', 'a'), ('c', 'a', 'b'), ('c', 'b', 'a')]
print(permutation_list2)
# [('a', 'b'), ('a', 'c'), ('b', 'a'), ('b', 'c'), ('c', 'a'), ('c', 'b')]
print(combination_list)
# [('a', 'b'), ('a', 'c'), ('b', 'c')]
print(bit_combinations)
# [(0, 0, 0, 0), (0, 0, 0, 1), (0, 0, 1, 0), (0, 0, 1, 1), (0, 1, 0, 0), (0, 1, 0, 1), (0, 1, 1, 0), (0, 1, 1, 1), (1, 0, 0, 0), (1, 0, 0, 1), (1, 0, 1, 0), (1, 0, 1, 1), (1, 1, 0, 0), (1, 1, 0, 1), (1, 1, 1, 0), (1, 1, 1, 1)]
```

(9) stack （栈） 后进先出

```
#辅助栈 快速堆猪
a=[]
m=[]
while True:
    try:
        s=input().split()
        if s[0]=='pop':
            if a:
                a.pop()
                if m:
                    m.pop()
        elif s[0]=='min':
            if m:
                print(m[-1])
        else:
            num=int(s[1])
            a.append(num)
            if not m:
```

```
                m.append(num)
            else:
                m.append(min(num,m[-1]))
    except EOFError:
        break
```

（10）list

```
list=[]
list.append(obj)#在列表末尾添加新的对象
list.count(obj)#统计某个元素在列表中出现的次数
list.extend(seq)#在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）
list.index(obj)#从列表中找出某个值第一个匹配项的索引位置
list.insert(index, obj)#将对象插入列表
list.pop([index=-1])#移除列表中的一个元素（默认最后一个元素），并且返回该元素的值
list.remove(obj)#移除列表中某个值的第一个匹配项
list.reverse()#反向列表中元素
list.sort(cmp=None, key=None, reverse=False)#对原列表进行排序
```

（11）二进制判断2的幂次

当 `x` 是 2 的幂次时，`x - 1` 的二进制表示中的所有位都与 `x` 的二进制表示中的 1 位相对应的位置相反。因此，`x & (x - 1)` 的结果为 0。

（12）切片 s[start:stop:step]

- 负数索引表示从序列的末尾开始计数。
- `-1` 表示最后一个元素，`-2` 表示倒数第二个元素，依此类推。

```
s = "Hello, world!"
print(s[-6:-1])  # 输出: worl
```

- 对于可变序列（如列表），你可以使用切片进行赋值。
- 通过组合 `start`、`stop` 和 `step` 参数，你可以实现多种不同的切片操作。
- 对于不可变序列（如字符串），切片只能用于读取；对于可变序列（如列表），还可以用于修改。

（14）按位异或运算

`lst[i] ^= 1` 是使用了 **按位异或运算符 (`^`)**，它表示对当前值进行翻转操作。这种写法用于在每次操作时**切换**牢房的状态，即如果牢房是锁着的 (`0`)，就将其解锁为 `1`；如果牢房是解锁的 (`1`)，就将其锁上为 `0`。

具体来说，`^` 是异或运算：0 ^ 1 = 1`（锁着变为解锁）`1 ^ 1 = 0` （解锁变为锁上）

```
t = int(input())
for _ in range(t):
    n = int(input())
    lst = [0]*n
    for i in range(2,n+1): #i是轮数
        for j in range(i-1,n,i):#j是索引 每隔i个数翻转一次
            lst[j]^=1
    print(lst.count(0))
```

（15）字典

字典应该不能用.sort(),应该可以用sorted(字典)

```
# 定义罗马数字和整数的映射关系
roman_to_int_map = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M':
1000}

# 定义整数到罗马数字的映射列表（从大到小顺序）
int_to_roman_map = [
    (1000, 'M'), (900, 'CM'), (500, 'D'), (400, 'CD'),
    (100, 'C'), (90, 'XC'), (50, 'L'), (40, 'XL'),
    (10, 'X'), (9, 'IX'), (5, 'V'), (4, 'IV'), (1, 'I')]
# 罗马数字转整数
def roman_to_int(s):
    total = 0
    prev_value = 0
    for char in s:
        value = roman_to_int_map[char]
        if value > prev_value:
            total += value - 2 * prev_value  # 处理特殊情况，如IV, IX
        else:
            total += value
        prev_value = value
    return total
# 整数转罗马数字
def int_to_roman(num):
    result = []
    for value, symbol in int_to_roman_map:
        while num >= value:
            result.append(symbol)
            num -= value
    return ''.join(result)
# 主函数，判断输入是罗马数字还是整数
def main():
    # 输入处理
    input_data = input().strip()

    # 判断输入是整数还是罗马数字
    if input_data.isdigit():
        # 输入是整数
        num = int(input_data)
        print(int_to_roman(num))
    else:
        # 输入是罗马数字
        print(roman_to_int(input_data))
```

```
# 调用主函数
if __name__ == "__main__":
    main()
```

（16）.zfill 与 .lstrip

方法.zfill 只用于字符串！！！！不用于列表！！！！填0

```
m = m.zfill(len(n))
```

使用 `zfill` 方法来确保两个字符串具有相同的长度。`zfill` 方法会在字符串前面填充 `0`，直到达到指定的长度。

方法.lstrip去除字符串左侧特定字符（比如0

`.lstrip('0')` 是 Python 字符串方法 `lstrip()` 的一种使用方式，用于去除字符串左侧（开头）的特定字符。在这个例子中，它用于去除字符串开头的零（`'0'`）。

(17)functools.cmp_to_key 自定义函数的比较

在 Python 的 `sorted()` 函数中，使用自定义比较函数来决定元素的排列顺序。当我们通过比较函数比较两个元素时：

- **如果返回正数**，Python 会认为**第一个元素应该排在第二个元素之后**（即 `y` 在 `x` 前面）。
- **如果返回负数**，则相反，**第一个元素应该排在第二个元素之前**（即 `x` 在 `y` 前面）
- ```
  import functools
  def compare(x,y):
      if x+y>y+x:
          return -1
      elif x+y<y+x:
          return 1
      else:
          return 0
  n=int(input())
  c=list(map(str,input().split()))
  c_max = sorted(c,key = functools.cmp_to_key(compare))
  c_min= sorted(c,key = functools.cmp_to_key(lambda x,y:compare(y,x)))
  print(''.join(map(str,c_max)),end = " ")
  print(''.join(map(str,c_min))
  ```
  ```

4.ASCII表

二进制	十进制	十六进制	图形	二进制	十进制	十六进制	图形
0100 0000	64	40	@	0110 0000	96	60	`
0100 0001	65	41	A	0110 0001	97	61	a
0100 0010	66	42	B	0110 0010	98	62	b
0100 0011	67	43	C	0110 0011	99	63	c
0100 0100	68	44	D	0110 0100	100	64	d
0100 0101	69	45	E	0110 0101	101	65	e
0100 0110	70	46	F	0110 0110	102	66	f
0100 0111	71	47	G	0110 0111	103	67	g
0100 1000	72	48	H	0110 1000	104	68	h
0100 1001	73	49	I	0110 1001	105	69	i
0100 1010	74	4A	J	0110 1010	106	6A	j
0100 1011	75	4B	K	0110 1011	107	6B	k
0100 1100	76	4C	L	0110 1100	108	6C	l
0100 1101	77	4D	M	0110 1101	109	6D	m
0100 1110	78	4E	N	0110 1110	110	6E	n
0100 1111	79	4F	O	0110 1111	111	6F	o
0101 0000	80	50	P	0111 0000	112	70	p
0101 0001	81	51	Q	0111 0001	113	71	q
0101 0010	82	52	R	0111 0010	114	72	r
0101 0011	83	53	S	0111 0011	115	73	s
0101 0100	84	54	T	0111 0100	116	74	t
0101 0101	85	55	U	0111 0101	117	75	u
0101 0110	86	56	V	0111 0110	118	76	v
0101 0111	87	57	W	0111 0111	119	77	w
0101 1000	88	58	X	0111 1000	120	78	x
0101 1001	89	59	Y	0111 1001	121	79	y
0101 1010	90	5A	Z	0111 1010	122	7A	z

ord 函数与chr函数 实现字母和数字的转换（分大小写）

`ord()` 函数`ord('A')` 返回 65，`ord('Z')` 返回 90。`chr()` 函数例如，`chr(65)` 返回 'A'，`chr(90)` 返回 'Z'。

5. 判断完全平方数 import math　sqrt_num = math.isqrt(num) if sqrt_num * sqrt_num == num

6.join的用法 join只能操作字符串列表想用它操作数字列表时可以

```
print(" ".join(map(str,a)))
```

三、递归与DFS（常用模版）

1. 最大通域面积（DFS）**（不需要回溯）

```
#我的写法
import sys
sys.setrecursionlimit(20000)

def dfs(x,y,c):
    global cnt
    c[x][y]='.'
    for dx,dy in directions:
        nx,ny=x+dx,y+dy
        if 0<=nx<N and 0<=ny<M and c[nx][ny]=='W':
            cnt+=1
            dfs(nx,ny,c)
for _ in range(int(input())):
    N,M=map(int,input().split())
    c=[list(input()) for i in range(N)]
    max_number=0
    directions=[(-1,0),(1,0),(0,-1),(0,1),(-1,1),(1,1),(1,-1),(-1,-1)]
    for i in range(N):
        for j in range(M):
            if c[i][j]=='W':
                cnt=1
                dfs(i,j,c)
                max_number=max(max_number,cnt)
    print(max_number)
```

2. 迷宫路径（DFS）（需要回溯）

```
dx=[-1,0,1,0]
dy=[0,1,0,-1]
def dfs(maze,x,y):
    global cnt

    for i in range(4):
        nx=x+dx[i]
        ny=y+dy[i]
        if maze[nx][ny]=='e':
            cnt+=1
            continue
        if maze[nx][ny]==0:
            maze[x][y]=1
            dfs(maze,nx,ny)
            maze[x][y]=0
    return
n,m=map(int,input().split())
maze=[[-1]*(m+2)]
for _ in range(n):
    temp=[-1]+list(map(int,input().split()))+[-1]
    maze.append(temp)
maze.append([-1]*(m+2))
```

```
maze[1][1]="s"
maze[n][m]="e"
cnt=0
dfs(maze,1,1)
print(cnt)
```

3.全排列（递归）

```
def quan(n, path, used, result):
    if len(path) == n:
        result.append(path[:])
    for i in range(1, n+1):
        if used[i]:
            continue
        used[i] = True
        path.append(i)

        quan(n, path, used, result)
        path.pop()
        used[i] = False
def pai(n):
    result = []
    used = [False] * (n + 1)
    quan(n, [], used, result)

    for i in result:
        print(" ".join(map(str, i)))
n = int(input())
pai(n)
```

四、dp问题

1. 斐波那契数列的lru-cache_dp

```
from functools import lru_cache
@lru_cache(maxsize=20)
```

2. 01背包问题 （拿或不拿，数目有限）

```
#二维矩阵 B是包的容量，N是东西的数目
N,B = map(int,input().split())
price=[0]+[int(x) for x in input().split()]
weight =[0]+[int(x) for x in input().split()]
bag = [[0]*(B+1) for i in range(N+1)]
for i in range(1,N+1):
    for j in range(1,B+1):
        if weight[i]<=j:
            bag[i][j]=max(bag[i-1][j],bag[i-1][j-weight[i]]+price[i])
        else:
            bag[i][j]=bag[i-1][j]
print(bag[-1][-1])
```

3.完全背包问题 （种类有限，数目无限

```
#求最小硬币数，用正无穷
n,m= map(int,input().split()) #m是需要凑的金额，n为面额的种类
zhi = list(map(int,input().split()))#具体面额
dp=[float('inf')]*(m+1)
dp[0]=0
for i in range(1,m+1):
    for j in zhi:
        if j<=i:
             dp[i]=min(dp[i-j]+1,dp[i])
if dp[m] != float('inf'):
    print(dp[m])
else:
    print(-1)#无法凑成的情况
```

```
#求最大能切割形成的段数，用负无穷
n,a,b,c = map(int,input().split())#n为原始丝带的长度，a，b，c为各种种类长度
dp=[0]+[float('-inf')]*n
for i in range(1,n+1):
    for j in (a,b,c):
        if i>=j:
            dp[i]=max(dp[i],dp[i-j]+1)

print(dp[n])
```

4.最长下降子序列

```
#拦截导弹
k = int(input())
m=list(map(int,input().split()))
dp =[1]*k #dp[i]表示以i元素为结尾的递减序列有多长
for i in range(1,k):
    for j in range(i):
        if m[i]<=m[j]:
            dp[i]=max(dp[i],dp[j]+1)
print(max(dp))
```

5.回文序列

```
#双指针，中心扩散法，不是指从最中心扩散，而是把每个锚点当作最中心去处理，去向两侧扩散
class Solution:
    def longestPalindrome(self, s: str) -> str:
        n=len(s)
        if n<2:
            return(s)
        else:
            start = 0;max_len=1
            def kuosan(x,y):
                while x>=0 and y<n and s[x]==s[y]:
                    x-=1
                    y+=1
                return x+1,y-1
```

```
            for i in range(n):
                x1,y1=kuosan(i,i)
                x2,y2=kuosan(i,i+1)
                if y1-x1+1>max_len:
                    start,max_len=x1,y1-x1+1
                if y2-x2+1>max_len:
                    start,max_len=x2,y2-x2+1
            return s[start:start+max_len]
```

6.冒泡排序

```
m=int(input())
n=int(input())
c=list(map(str,input().split()))
for i in range(len(c)-1):
    for j in range(i+1, len(c)):
        #print(i,j)
        if int(c[i] + c[j]) < int(c[j] + c[i]):#不加int是字典序
            c[i], c[j] = c[j], c[i]
dp=[[0]*(m+1) for _ in range(n+1)]
for i in range(1,n+1):
    for j in range(1,m+1):
        if j>=len(c[i-1]):#01背包问题
            dp[i][j] = max(dp[i - 1][j], int(str(dp[i - 1][j - len(c[i - 1])]) +
c[i - 1])) #为什么是i-1，因为这里的i是从0开始的，没有在c前面加上【0】
        else:
            dp[i][j] = dp[i - 1][j]
print(dp[-1][-1])
```

7.走楼梯

```
n=int(input())#最简单版本
if n<2:
    print(1)
else:
    dp=[-1]*(n+1)
    dp[0]=0
    dp[1]=1
    dp[2]=2
    for i in range(3,n+1):
        dp[i]=dp[i-1]+dp[i-2]
    print(dp[n])
```

8.双指针

```
p = int(input())#军备竞赛左右两侧，也有从左侧开始的
n = [int(x) for x in input().split()]
n.sort()

cnt=0
i=0
j=len(n)-1
```

```
while i<=j:
    if p>=n[i]:
        p-=n[i]
        cnt+=1
        i+=1
    else:
        if j == i:
            break
        if cnt>0:
            p+=n[j]
            cnt-=1
            j-=1
        else:
            break
print(cnt)
```

五、BFS

1. 寻宝 BFS #最短多少步

```
#我的写法
directions=[(-1,0),(1,0),(0,-1),(0,1)]
from collections import deque
def bfs(x,y):
    q=deque()
    q.append((x,y))
    iq[x][y]=True
    step=0
    while q:
        for _ in range(len(q)):
            x, y = q.popleft()
            if c[x][y] == 1:
                return step
            for dx,dy in directions:
                nx,ny=x+dx,y+dy
                if 0<=nx<m and 0<=ny<n and not iq[nx][ny] and c[nx][ny] !=2:
                    q.append((nx,ny))
                    iq[nx][ny]=True
        step+=1
    return 'NO'
m,n=map(int,input().split())
iq=[[False]*n for _ in range(m)]
c=[list(map(int,input().split())) for i in range(m)]
print(bfs(0,0))
```

2.两座孤岛最短距离 bfs+dfs

```
from collections import deque
def dfs(x,y,c,queue):
    c[x][y]=2
    queue.append((x,y))
    for dx,dy in directions:
        nx,ny=x+dx,y+dy
```

```
        if 0<=nx<n and 0<=ny<n and c[nx][ny]==1:
            dfs(nx,ny,c,queue)
distance=0
def bfs(c,queue):
    global distance
    while queue:
        for _ in range(len(queue)):
            x,y=queue.popleft()
            for dx,dy in directions:
                nx,ny=x+dx,y+dy
                if 0<=nx <n and 0<=ny<n:
                    if c[nx][ny]==1:
                        return distance
                    elif c[nx][ny]==0:
                        c[nx][ny]=2
                        queue.append((nx,ny))
        distance+=1
    return distance
n = int(input())
c=[list(map(int,input())) for i in range(n)]
directions=[(-1,0),(0,1),(1,0),(0,-1)]
queue=deque()
def jisuan(n,c):
    for i in range(n):
        for j in range(n):
            if c[i][j]==1:
                dfs(i,j,c,queue)
                return bfs(c,queue)#必须用函数，用两层break的化会输出错误
print(jisuan(n,c))

#Dijk算法 等下好好钻研一下 heapq !
import heapq
n=int(input())
land=[]
dir=[[0,1],[0,-1],[1,0],[-1,0]]
for i in range(n):
    land.append(input())
visited=[[True]*(len(land[i])) for i in range(len(land))]
def find(x1,y1):
    pos=[]
    heapq.heappush(pos,(0,x1,y1))
    visited[x1][y1]=False
    while pos:
        step,x,y=heapq.heappop(pos)
        #print(step,x,y)
        if land[x][y]=="1" and step!=0:
            return step
        for dx,dy in dir:
            nx,ny=x+dx,y+dy
            if 0<=nx<n and 0<=ny<len(land[nx]) and visited[nx][ny]:
                visited[nx][ny]=False
                if land[nx][ny]==land[x][y] and land[x][y]!="0":
                    heapq.heappush(pos,(step,nx,ny))
                else:
                    heapq.heappush(pos,(step+1,nx,ny))
mark=False
```

```
for i in range(n):
    for j in range(len(land[i])):
        if land[i][j]=="1":
            print(find(i,j)-1)
            exit()
```

3.变换的迷宫

```
from collections import deque
def bfs(x,y):
    visited=set()
    visited.add((0,x,y))
    q=deque([(0,x,y)])
    while q:
        time,x,y=q.popleft()
        if m[x][y]=="E":
            return time
        for dx,dy in directions:
            nx,ny=x+dx,y+dy
            if 0<=nx<r and 0<=ny<c  and ((time+1)%k,nx,ny) not in visited:
                if m[nx][ny]!='#' or (time+1)%k==0:
                    q.append((time+1,nx,ny))
                    visited.add(((time+1)%k,nx,ny))
    return "Oop!"
directions=[(1,0),(-1,0),(0,1),(0,-1)]
t=int(input())
for _ in range(t):
    r,c,k=map(int,input().split())
    m=[list(input()) for _ in range(r)]
    for i in range(r):
        for j in range(c):
            if m[i][j]=='S':
                print(bfs(i,j))
```

六、矩阵

```
#螺旋矩阵
def matrix(n):
    c=[[0]*n for _ in range(n)]
    num=1
    top=0;bottom=n-1;left=0;right=n-1
    while num<=n*n:
        for i in range(left,right+1):
            c[top][i]=num
            num+=1
        top+=1
        for j in range(top,bottom+1):
            c[j][right]=num
            num+=1
        right-=1
        for l in range(right,left-1,-1):
            c[bottom][l]=num
            num+=1
        bottom-=1
```

```
        for k in range(bottom,top-1,-1):
            c[k][left]=num
            num+=1
        left+=1
    return c

def xing(c):
    for i in c:
        print(" ".join(map(str,i)))


n= int(input())
p_matrix = matrix(n)
xing(p_matrix)
```

保护圈

```
n,m = map(int,input().split())
mx =[[0]*(m+2) for j in range(n+2)]
c=[]
for _ in range(n):
  c.append(list(map(int,input().split())))
for i in range(1,n+1):
    for j in range(1,m+1):
        mx[i][j]=c[i-1][j-1]
new_mx = [[0]*(m+2) for j in range(n+2)]
for i in range(1,n+1):
    for j in range(1,m+1):
        x =mx[i-1][j-1]+mx[i-1][j]+mx[i-1][j+1]+mx[i][j-1]+mx[i][j+1]+mx[i+1][j-
1]+mx[i+1][j]+mx[i+1][j+1]
        if mx[i][j]==1:
            if x<2:
                new_mx[i][j]=0
            elif x==2 or x==3:
                new_mx[i][j]=1
            else:
                new_mx[i][j]=0
        else:
            if x==3:
                new_mx[i][j]=1
for i in range(1,n+1):
    print(' '.join(map(str,new_mx[i][1:m+1])))
```

矩阵乘法

```
n,m1,m2=map(int,input().split())
x=[[0]*n for i in range(n)]
y=[[0]*n for j in range(n)]
z=[[0]*n for k in range(n)]
for i in range(m1):
    a,b,c = map(int,input().split())
    x[a][b]=c
for j in range(m2):
    d,e,f = map(int,input().split())
```

```
    y[d][e]=f
for i in range(n):
    for j in range(n):
        for l in range(n):
            z[i][j]+=x[i][l]*y[l][j]
        if z[i][j] != 0:
            print(i,j,z[i][j])
```

卷积运算

```
m,n,p,q = map(int,input().split())
c=[]
d=[]
for _ in range(m):
    c.append(list(map(int,input().split())))
for _ in range(p):
    d.append(list(map(int,input().split())))
e =[[0 for j in range(n+1-q)] for i in range(m+1-p)]
for i in range(m+1-p):
    for j in range(n+1-q):
        for k in range(p):
            for l in range(q):
                e[i][j]+=c[i+k][j+l]*d[k][l]
for i in range(m+1-p):
    print(' '.join(map(str,e[i])))
```

七、区间问题

区间合并 合并所有有交集的区间，最后问合并之后的区间 （按左端点排序，维护前面区间最右端点为初始值

【步骤一】：按照区间**左端点**从小到大排序。

【步骤二】：维护前面区间中最右边的端点为ed。从前往后枚举每一个区间，判断是否应该将当前区间视为新区间。

假设当前遍历到的区间为第i个区间 $[l_i, r_i]$，有以下两种情况：

- $l_i \leq ed$：说明当前区间与前面区间**有交集**。因此**不需要**增加区间个数，但需要设置 $ed = max(ed, r_i)$。
- $l_i > ed$: 说明当前区间与前面**没有交集**。因此**需要**增加区间个数，并设置 $ed = max(ed, r_i)$。

选择不相交区间 要求选择尽量多的区间，使得这些区间互不相交，求可选取的区间的最大数量

【步骤一】：按照区间**右端点**从小到大排序。

【步骤二】：从前往后依次枚举每个区间。

假设当前遍历到的区间为第i个区间 $[l_i, r_i]$，有以下两种情况：

- $l_i \leq ed$：说明当前区间与前面区间有交集。因此直接跳过。
- $l_i > ed$: 说明当前区间与前面没有交集。因此选中当前区间，并设置 $ed = r_i$。

区间选点问题 给出一堆区间，取尽量少的点，使得每个区间内至少有一个点（不同区间内含的点可以是同一个），如进程检测，与选择不相交区间做法一致，按右端点排列

区间覆盖问题 给出一堆区间和一个目标区间，问最少选择多少区间可以覆盖掉题中给出的这段目标区间

【步骤一】：按照区间左端点从小到大排序。

步骤二】：**从前往后**依次枚举每个区间，在所有能覆盖当前目标区间起始位置start的区间之中，选择**右端点**最大的区间。

假设右端点最大的区间是第i个区间，右端点为 r_i。

最后将目标区间的start更新成r_i

```
        clips.sort()

        st, ed = 0, time
        res = 0

        i = 0
        while i < len(clips) and st < ed:
            maxR = 0
            # 找到所有起点小于等于 st 的片段，并记录这些片段的最大终点 maxR
            while i < len(clips) and clips[i][0] <= st:
                maxR = max(maxR, clips[i][1])
                i += 1

            if maxR <= st:
                # 无法继续覆盖
                return -1

            # 更新 st 为 maxR，并增加结果计数
            st = maxR
            res += 1

            if maxR >= ed:
                # 已经覆盖到终点
                return res

        # 如果没有成功覆盖到终点
        return -1
```

区间分组问题 给出一堆区间，问最少可以将这些区间分成多少组使得每个组内的区间互不相交 左端点从小到大

```
from typing import List
import heapq

class Solution:
    def minmumNumberOfHost(self, n: int, startEnd: List[List[int]]) -> int:
        # 按左端点从小到大排序
        startEnd.sort(key=lambda x: x[0])

        # 创建小顶堆
        q = []

        for i in range(n):
            if not q or q[0] > startEnd[i][0]:
                heapq.heappush(q, startEnd[i][1])
            else:
                heapq.heappop(q)
                heapq.heappush(q, startEnd[i][1])

        return len(q)
```

八、 逃生指南

1. 除法是否使用地板除得到整数？（否则 4/2=2.0）

2. 是否有缩进错误?

3. 用于调试的print是否删去?

4. 非一般情况的边界情况是否考虑?

5. 递归中return的位置是否准确？（缩进问题,逻辑问题）

6. 贪心是否最优？有无更优解?

7.元组不可变，其余可变

8.审题是否准确？ 是否漏掉了输出？（参考）

9.超时的时候，是不是有死循环没有break，要么使用前缀和，要么做提前处理

10.空格也算字符！！！！！！

11.有周期就取模，绝对位置不重要，重要的是相对位置！！

12.当e是一个列表，且e的值会在进行过程中变化，num.append((cnt, e.copy()))，因为e是可变的。
不可以a=max(a,b),b=min(a,b)因为a,b是会变的。