CS101 Cheat_Sheet_Edition_3

2023.12.26 compiled by 23工院 武昱达

改正了欧拉筛；

加入了排列组合、阶乘、笛卡尔积等内容

一、语法糖和常用函数

1. Part 1

```
"""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"
```

2.part 2

```
# 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
```

二、工具

1. 素数筛

写法1（欧拉筛）：

```
def Euler_sieve(n):
    primes = [True for _ in range(n+1)]
    p = 2
    while p*p <= n:
        if primes[p]:
            for i in range(p*p, n+1, p):
                primes[i] = False
        p += 1
    primes[0]=primes[1]=False
    return primes
print(Euler_sieve(20))
# [False, False, True, True, False, True, False, True, False, False, False, True,
False, True, False, False, False, True, False, True, False]
```

写法2（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]
```

写法3（欧拉筛）：

```
# 胡睿诚 23数院
N=20
primes = []
is_prime = [True]*N
is_prime[0] = False;is_prime[1] = False
for i in range(2,N):
    if is_prime[i]:
        primes.append(i)
    for p in primes: #筛掉每个数的素数倍
        if p*i >= N:
            break
        is_prime[p*i] = False
        if i % p == 0: #这样能保证每个数都被它的最小素因数筛掉！
            break
print(primes)
# [2, 3, 5, 7, 11, 13, 17, 19]
```

2. 简单题可以多循环（提醒）

例：完美立方

```
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)+")")

#以下是第二种优化方案
n=int(input())
import math
for a in range(2,n+1):
    for b in range(2,int(math.pow(a**3/3,1/3))+1):
        for c in range(b,int(math.pow(a**3/2,1/3))+1):
            for d in range(c,a):
                if a**3==b**3+c**3+d**3:
                    print('Cube = '+str(a)+', Triple =
('+str(b)+','+str(c)+','+str(d)+')')
```

3. 拓展包

(1) math

```
import math
print(math.ceil(1.5)) # 2
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!
```

(2) lru_cache

```
# 需要注意的是，使用@lru_cache装饰器时，应注意以下几点：
# 1.被缓存的函数的参数必须是可哈希的，这意味着参数中不能包含可变数据类型，如列表或字典。
# 2.缓存的大小会影响性能，需要根据实际情况来确定合适的大小或者使用默认值。
# 3.由于缓存中存储了计算结果，可能导致内存占用过大，需谨慎使用。
# 4.可以是多参数的。
```

(3) bisect（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
```

(4) 年份calendar包

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

(5) heapq 优先队列

```
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（默认最小，相反数调转）
heapq.heappush(lst,element) # 把元素压入堆中
```

(6) Counter包

```
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

defaultdict是Python中collections模块中的一种数据结构，它是一种特殊的字典，可以为字典的值提供默认值。当你使用一个不存在的键访问字典时，defaultdict会自动为该键创建一个默认值，而不会引发KeyError异常。

defaultdict的优势在于它能够简化代码逻辑，特别是在处理字典中的值为可迭代对象的情况下。通过设置一个默认的数据类型，它使得我们不需要在访问字典中不存在的键时手动创建默认值，从而减少了代码的复杂性。

使用defaultdict时，首先需要导入collections模块，然后通过指定一个默认工厂函数来创建一个defaultdict对象。一般来说，这个工厂函数可以是int、list、set等Python的内置数据类型或者自定义函数。

```
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)]
```

4.ASCII表

常用 ASCII 码表对照表

ASCII 表

ASCII 值	控制字符	ASCII 值	控制字符	ASCII 值	控制字符	ASCII 值	控制字符
0	NUT	32	(space)	64	@	96	、
1	SOH	33	!	65	A	97	a
2	STX	34	”	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	‘	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DCI	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	X	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	TB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~

5. 判断完全平方数

```
import math
def isPerfectSquare(num):
    if num < 0:
        return False
    sqrt_num = math.isqrt(num)
    return sqrt_num * sqrt_num == num
print(isPerfectSquare(97)) # False
```

三、递归与DFS（常用模版）

由于是按照个人习惯写的，可能与标准模板差异比较大。

1. 八皇后的回溯算法：

```
cols=[i for i in range(8)]
res=[]
def Queen(path,choices,main_diag,vice_diag):
    #退出条件
    if len(path)==8:
        temp=[str(j+1) for j in path]
        res.append("".join(temp))
        return
    #下一步操作
    for j in choices:
        #剪枝操作
        if j in path or j+len(path) in vice_diag or j-len(path) in main_diag:
            continue
        new_path = path + [j]
        new_main_diag = main_diag.copy()
        new_vice_diag = vice_diag.copy()
        new_main_diag.add(j - len(path))
        new_vice_diag.add(j + len(path))
        #下一层递归
        Queen(new_path, choices, new_main_diag, new_vice_diag)
main_diags=set()
vice_diags=set()
#直接调用函数，没有返回值
Queen([],cols,main_diags,vice_diags)
lst=[]
t=int(input())
for _ in range(t):
    lst.append(res[int(input())-1])
for i in lst:
    print(i)
```

2. 最大通域面积（DFS）

```
def dfs(matrix,x,y,visited):
```

```
    #如果越界，或接触边界，或接触已经标记的点，立刻终止递归并返回0
    if (x<0 or x>=len(matrix) or y<0 or y>=len(matrix[0])
            or matrix[x][y]!="W" or visited[x][y]):
        return 0
    visited[x][y]=True
    area=1
    directions = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1),(1, -1), (1, 0),
(1, 1)]
    for dx, dy in directions:
        area += dfs(matrix, x + dx, y + dy, visited)
    return area
def max_adj_area(matrix):
    rows,cols=len(matrix),len(matrix[0])
    # visited 辅助空间
    visited=[[False for _ in range(cols)] for _ in range(rows)]
    max_area=0
    for row in range(rows):
        for col in range(cols):
            if matrix[row][col]=="W" and not visited[row][col]:
                area=dfs(matrix,row,col,visited)
                max_area=max(max_area,area)
    return max_area
T=int(input())
for _ in range(T):
    N,M=map(int,input().split())
    matrix_1=[input() for _ in range(N)]
    print(max_adj_area(matrix_1))
```

3. 迷宫路径（DFS）

```
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)
```

4. I'm glad that I've had my flight——回溯

dfs生成排列

```
from recviz import recviz


maxn = 11
hashTable = [False]*maxn  # 当整数i已经在数组 P中时为 true

@recviz
def increasing_permutaions(n, prefix=[]):
    if len(prefix) == n:  # 递归边界，已经处理完排列的1~位
        return [prefix]
```

```

    result = []
    for i in range(1, n+1):
        if hashTable[i]:
            continue

        hashTable[i] = True  #记i已在prefix中
        # 把i加入当前排列，处理排列的后续号位
        result += increasing_permutaions(n, prefix+[i])
        hashTable[i] = False #处理完为i的子问题，还原状态

    return result


n = int(input())
result = increasing_permutaions(n)
for r in result:
    print(r)
```

四、dp问题

1. 斐波那契数列的简单dp（手动）

```
def Fplus(x):
    if x<=2:
        return 1
    else:
        if dp[x]!=-1:
            return dp[x]
        else:
            return Fplus(x-1)+Fplus(x-2)
```

```
lst_res=[]
for i in range(int(input())):
    n=int(input())
    dp=[-1 for _ in range(n+1)]
    temp=Fplus(n)
    lst_res.append(temp)
for i in lst_res:
    print(i)
```

2. 斐波那契数列的lru-cache_dp

```
from functools import lru_cache
@lru_cache(maxsize=20)
def F(x):
    if x<=2:
        return 1
    else:
        return F(x-1)+F(x-2)
print(F(30))
```

3. 小偷背包（采药）

```
t,m=map(int,input().split())
lst_tm=[]

for _ in range(m):
    temp=tuple(map(int,input().split()))
    if temp[0]<=t:
        lst_tm.append(temp)
lst_tm.sort(key=lambda x: x[0])
dp=[[0]*(t+1) for _ in range(len(lst_tm)+1)]
print(dp)

#在左上角加了一个保护圈，使横坐标背包的容量从1开始。
def MaxValue(item,time_temp):
    value_item=lst_tm[item-1][1]
    time_per=lst_tm[item-1][0]
    if item==1:
        if time_per<=time_temp:
            dp[item][time_temp]+=value_item
            return dp[item][time_temp]
    else:
        if time_per>time_temp:
            if dp[item-1][time_temp]!=0:
                return dp[item-1][time_temp]
            else:
                dp[item][time_temp]=MaxValue(item-1,time_temp)
                return dp[item][time_temp]
        else:
            if dp[item-1][time_temp-time_per]*\
                    dp[item-1][time_temp]!=0:
                return max(dp[item-1][time_temp-time_per]*\
                    dp[item-1][time_temp])
```

```
            else:
                dp[item][time_temp]=max(MaxValue(item-1,time_temp-time_per),
                                        MaxValue(item-1,time_temp))
                return dp[item][time_temp]

print(MaxValue(len(lst_tm),t))
```

五、BFS

1. 寻宝 BFS

```
# 苏王捷 23工院
# 武昱达注：用heapq替代了双向队列，用入堆模拟入队，用pop模拟出队。
import heapq
def bfs(x,y,matrix):
    # 定义步空间
    dx,dy=[1,0,-1,0],[0,1,0,-1]
    # 定义队列（heapq实现）和visited辅助空间
    queue,visited=[],set()
    heapq.heappush(queue,[0,x,y])
    visited.add((x,y))
    # while queue 队列非空则继续
    while queue:
        # 拆解当前访问坐标
        step,x,y=heapq.heappop(queue)
        # 当前访问元素的退出条件
        if matrix[x][y]==1:
            return step
        # 访问当前元素的比邻元素
        for i in range(4):
            # 取每种可能的下一步
            nx,ny=x+dx[i],y+dy[i]
            # 若不是障碍物且未经访问则入队
            if matrix[nx][ny]!=2 and (nx,ny) not in visited:
                heapq.heappush(queue,[step+1,nx,ny])
                visited.add((nx,ny))
    # 最终返回结果
    return "NO"

m,n=map(int,input().split())
matrix=[[2]*(n+2)]
for _ in range(m):
    matrix.append([2]+list(map(int,input().split()))+[2])
matrix.append([2]*(n+2))
print(bfs(1,1,matrix))
```

六、 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. 正难则反（参考 #蒋子轩 23工院# 乌鸦坐飞机）

8. 审题是否准确？ 是否漏掉了输出？ （参考）