计概B cheat sheet

```
"""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"
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
```

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!
```

排序sort加lambda

```
list.sort(key=lambda x: x[i])#也可以用dic[x，或者不用lambda,直接def函数也可
```

列表倒序 `list[::-1]`

大小写切换&删空格

```
str.upper()     str.lower()     str.title()
str.rstrip()    str.lstrip()    str.strip()
```

进制：2进制 `bin()` 输出0b...，八进制 `oct()` 输出0o...，十六进制 `hex()` 输出0x...，`int(str,base)` 可以把字符串按照指定进制转为十进制默认base=10

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
```

闰年

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

counter

```
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
```

itertools

```
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)]
```

质数筛

```
n=10**4
prime=[True for _ in range(n+1)]
p=2
while p*p<=n:
    if prime[p]:
        for i in range(p*p,n+1,p):
            prime[i]=False
    p+=1
primes=set([p for p in range(2,n+1) if prime[p]])
```

埃氏筛（基本上够用了）

```
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
```

欧拉筛（线性筛）

```
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
```

分解因数

```
def decompositions(n,minfactor):
    if n==1:
        return 1
    count=0
    for i in range(minfactor,n+1):
        if n%i==0:
            count+=decompositions(n//i,i)
    return count
print(decompositions(x,2))
```

确定平方相关：

```
def is_sq(n):
    if int(n**0.5)**2==n:
        return True
```

```
    else:
        return False

def sq_sum(m):
    count = 0
    for i in range(1,m//2+1):
        if is_sq(i) and is_sq(m-i):
            count+=1
    if count==0:
        return False
    return True
```

找中位数：

```
def find_median(lst):
    n = len(lst)
    new_list = sorted(lst)
    if n%2==1:
        return new_list[n//2]
    else:
        return (new_list[n//2]+new_list[n//2-1])/2
```

输入带括号的怎么拆：

```
pairs = [i[1:-1] for i in input().split()]
distances = [sum(map(int,i.split(','))) for i in pairs]
```

二分查找

```
#注意边界命名
#注意跳出循环

lst = list(map(int,input().split()))
lst.sort()

n = int(input())

left = 0
right = len(lst)-1

while left <= right:
    mid = (left+right)//2
    if lst[mid] < n:
        left = mid+1
    elif lst[mid] > n:
        right = mid-1
    else:
        print(mid)
        break
else:
    print("not in")
```

双指针

```python
def three_sum(nums):
    nums.sort()
    n = len(nums)
    result = set()

    for i in range(n):
        if i > 0 and nums[i] == nums[i-1]:
            continue  # 跳过重复的数字

        left, right = i + 1, n - 1
        while left < right:
            total = nums[i] + nums[left] + nums[right]
            if total < 0:
                left += 1
            elif total > 0:
                right -= 1
            else:
                result.add((nums[i], nums[left], nums[right]))
                while left < right and nums[left] == nums[left + 1]:
                    left += 1
                while left < right and nums[right] == nums[right - 1]:
                    right -= 1
                left += 1
                right -= 1

    return result

# 读取输入数据
numbers = list(map(int, input().split()))

# 查找并打印结果
found = three_sum(numbers)
print(len(found))  # 打印符合条件的三元组数量
for triple in found:
    print(triple)  # 打印每个三元组
```

OJ18164：剪绳子

```python
#使用堆
import heapq

n = int(input())
segments = list(map(int, input().split()))
heapq.heapify(segments)  # Convert list into a heap
connected_segs = []

while len(segments) > 1:
    smallest = heapq.heappop(segments)  # Pop the smallest element
    next_smallest = heapq.heappop(segments)  # Pop the next smallest element
    connected_seg = smallest + next_smallest
    heapq.heappush(segments, connected_seg)  # Push the sum back into the heap
    connected_segs.append(connected_seg)
```

```
print(sum(connected_segs))
```

heap用法

```
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
```

最大最小整数

```
n = int(input())
num = input().split()

for i in range(n-1):
    for j in range(i+1,n):
        if num[i]+num[j]<num[j]+num[i]:
            num[i],num[j] = num[j],num[i]
ans = "".join(num)
num.reverse()
print(ans+" "+"".join(num))
```

permutation

```
from itertools import permutations

m = int(input())
mx = [input().split() for _ in range(m)]

def v(i):
    a,b,c,d = i
    l2 = list(permutations([a,b,c,d], 4))
    for j in l2:
        a,b,c,d = j
        for u in ('+','-'):
            for j in ('+','-'):
                for k in ('+', '-'):
                    if eval(a+u+b+j+c+k+d)==24:
                        return 'YES'
    return 'NO'

for i in mx:
    print(v(i))
```

红蓝玫瑰

```
r=list(input())
n=len(r)
R=[0]*n
B=[0]*n
if r[0]=="R":R[0]=0;B[0]=1
else:R[0]=1;B[0]=0
```

```
for i in range(n-1):
    if r[i+1]=="R":
        R[i+1]=R[i]
        B[i+1]=min(R[i],B[i])+1
    else:
        R[i+1]=min(R[i],B[i])+1
        B[i+1]=B[i]
print(R[-1])
```

无限制的输入

```
while True:
    try:
        a, b = input().split()
    except EOFError:
        break
```

最长公共子序列

```
while True:
    try:
        a, b = input().split()
    except EOFError:
        break

    alen = len(a)
    blen = len(b)

    dp = [[0]*(blen+1) for i in range(alen+1)]

    for i in range(1, alen+1):
        for j in range(1, blen+1):
            if a[i-1]==b[j-1]:
                dp[i][j] = dp[i-1][j-1] + 1
            else:
                dp[i][j] = max(dp[i-1][j], dp[i][j-1])


    print(dp[alen][blen])
```

0-1背包问题

```
dp = [0]*T
for i in range(n):
    for t in range(T,time[i]-1,-1):
        dp[t] = max(dp[t],dp[t-time[i]]+value[i])
ans = dp[T]
```

```
n,b=map(int, input().split())
price=[0]+[int(i) for i in input().split()]
weight=[0]+[int(i) for i in input().split()]
bag=[[0]*(b+1) for _ 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(price[i]+bag[i-1][j-weight[i]], bag[i-1][j])
        else:
            bag[i][j]=bag[i-1][j]
print(bag[-1][-1])
```

```
def best_add_up(n, t, prices):
    # 初始化DP数组，大小为t + max(prices) + 1，以容纳所有可能的价格
    max_price = max(prices)
    dp = [0] + [-1] * (t + max_price)

    # 动态规划填表
    for price in prices:
        for j in range(t + max_price, price - 1, -1):
            if dp[j - price] != -1:
                dp[j] = max(dp[j], dp[j - price] + price)

    # 寻找最接近t的值
    for i in range(t, t + max_price + 1):
        if dp[i] != -1:
            return dp[i]

    return 0

# 读取输入
n, t = map(int, input().split())
prices = list(map(int, input().split()))

# 最佳凑单
result = best_add_up(n, t, prices)
print(result)
```

摆动数列

```
n = int(input())

nums = list(map(int,input().split()))

if n<2:
    print(n)
else:
    length = 1
    prev_diff = 0
    for i in range(1,n):
        diff = nums[i]-nums[i-1]
        if (prev_diff<=0 and diff>0) or (prev_diff>=0 and diff<0):
            prev_diff = diff
            length+=1
```

```
    print(length)
```

最长上升子序列

```
n = int(input())
*b, = map(int, input().split())
dp = [1]*n

for i in range(1, n):
    for j in range(i):
        if b[j] < b[i]:
            dp[i] = max(dp[i], dp[j]+1)

print(max(dp))
```

剪丝带（完全背包）

```
n, a, b, c = map(int, input().split())
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-j] + 1, dp[i])

print(dp[n])
```

dfs

最大连通域

```
dire = [[-1,-1],[-1,0],[-1,1],[0,-1],[0,1],[1,-1],[1,0],[1,1]]

area = 0
def dfs(x,y):
    global area
    if matrix[x][y] == '.':return
    matrix[x][y] = '.'
    area += 1
    for i in range(len(dire)):
        dfs(x+dire[i][0], y+dire[i][1])


for _ in range(int(input())):
    n,m = map(int,input().split())

    matrix = [['.' for _ in range(m+2)] for _ in range(n+2)]
    for i in range(1,n+1):
        matrix[i][1:-1] = input()

    sur = 0
    for i in range(1, n+1):
        for j in range(1, m+1):
            if matrix[i][j] == 'W':
```

```
                area = 0
                dfs(i, j)
                sur = max(sur, area)
    print(sur)
```

```
def neighbor(i, j, graph):
    neighbors = []
    directions = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0),
(1, 1)]
    N, M = len(graph), len(graph[0])

    for di, dj in directions:
        ni, nj = i + di, j + dj
        if 0 <= ni < N and 0 <= nj < M and graph[ni][nj] == "W":
            neighbors.append((ni, nj))
    return neighbors

def dfs(graph_tree, node, visited):
    if node not in visited:
        visited.add(node)
        count = 1
        for neighbor in graph_tree[node]:
            count += dfs(graph_tree, neighbor, visited)
        return count
    return 0

# 读取输入
n = int(input())
for _ in range(n):
    N, M = map(int, input().split())
    graph = [list(input().strip()) for _ in range(N)]

    graph_tree = {}
    for i in range(N):
        for j in range(M):
            if graph[i][j] == "W":
                graph_tree[(i, j)] = neighbor(i, j, graph)

    maxim = 0
    visited = set()
    for location in graph_tree:
        area = dfs(graph_tree, location, visited)
        maxim = max(maxim, area)

    print(maxim)
```

最大联通数

```
dir = [[-1, 0], [1, 0], [0, -1], [0, 1], [-1, -1], [-1, 1], [1, -1], [1, 1]]
def dfs(x, y):
    if x < 0 or x >= N or y < 0 or y >= M or board[x][y] != 'W':
        return
    board[x][y] = '.'
    for i in range(len(dir)):
```

```
        dfs(x + dir[i][0], y + dir[i][1])

T = int(input())
while T!=0:
    T -= 1
    N, M = map(int, input().split())
    board = []
    ans = 0
    for i in range(N):
        board.append(list(input()))
    for i in range(N):
        for j in range(M):
            if board[i][j] == 'W':
                ans += 1
                dfs(i, j)
    print(ans)
```

bfs

kefa and park

```
cat = dict()
graph = dict()
visited = set()
queue = [1]
res = 0
n,m = map(int, input().split())
a = list(map(int, input().split()))
for _ in range(n-1):
    x,y = map(int, input().split())
    if x not in graph.keys():
        graph[x] = []
    if y not in graph.keys():
        graph[y] = []
    graph[x].append(y)
    graph[y].append(x)

cat[1] = a[0]
while len(queue)>0:
    x = queue.pop(0)
    visited.add(x)
    if cat[x]>m:
        continue
    b=0
    for k in graph[x]:
        if k not in visited:
            if a[k-1]==1:
                cat[k] = cat[x] + 1
            else:
                cat[k] = 0
            queue.append(k)
            b=1
    if b==0:
        res+=1
print(res)
```

datetime

```
# 请改为同学的代码
import datetime

n = int(input())
for _ in range(n):
    sm,sd,b,em,ed = map(int,input().split())
    start_date = datetime.date(2023,sm,sd)
    end_date = datetime.date(2023,em,ed)
    interval = end_date - start_date
    m = interval.days
    print(b*(2**m))
```

pixels

```
#
def check_square(matrix, i, j):
    try:
        if matrix[i][j] == matrix[i+1][j] == matrix[i][j+1] == matrix[i+1][j+1]
== 1:
            return True
    except IndexError:
        return False
    return False

n, m, k = map(int, input().split())

grid = [[0 for _ in range(m)] for _ in range(n)]

first_square_move = 0

for move in range(k):
    i, j = map(int, input().split())

    grid[i-1][j-1] = 1

    if first_square_move == 0:
        for x in range(max(0, i-2), min(n, i+1)):
            for y in range(max(0, j-2), min(m, j+1)):
                if check_square(grid, x, y):
                    first_square_move = move+1
                    break
            if first_square_move != 0:
                break

print(first_square_move)
```

矩阵处理

```
#
n,m = map(int,input().split())
matrix = []
for _ in range(n):
```

```
    lst = list(map(int,input().split()))
    matrix.append(lst)


def adjacent_cells(i,j,n,m):
    count=0
    check = [(i-1,j-1),(i-1,j),(i-1,j+1),(i,j-1),(i,j+1),(i+1,j-1),(i+1,j),
(i+1,j+1)]
    for location in check:
        if 0<=location[0]<n and 0<=location[1]<m:
            if matrix[location[0]][location[1]]==1:
                count+=1
    return count


new_matrix = [row.copy() for row in matrix]

for i in range(n):
    for j in range(m):
        count = adjacent_cells(i, j, n, m)
        if matrix[i][j] == 1:
            if count < 2 or count > 3:
                new_matrix[i][j] = 0
        elif matrix[i][j] == 0:
            if count == 3:
                new_matrix[i][j] = 1

matrix = new_matrix
for row in matrix:
    print(' '.join(map(str, row)))
```

双指针

```
while l < r:
    if num[l-1] == vmin:
      l += 1
      vmin += 1
    elif num[l-1] == vmax:
      l += 1
      vmax -= 1
    elif num[r-1] == vmin:
      r -= 1
      vmin += 1
    elif num[r-1] == vmax:
      r -= 1
      vmax -= 1
    else:
      break

if l < r:
    print(l, r)
else:
    print(-1)
```

二分查找案例

```
#
shop_num = int(input())
drink_price = list(map(int,input().split()))
sorted_price = sorted(drink_price)

days = int(input())
for n in range(days):
    coins = int(input())
    left, right = 0,shop_num-1
    while left<=right:
        mid = (left+right)//2
        if coins>=sorted_price[mid]:
            left = mid+1
        else:
            right = mid-1
    print(left)
```

扩充矩阵避免索引问题

```
expanded_matrix = []

expanded_matrix.append([2] * (cols + 2))

for row in original_matrix:
    expanded_matrix.append([2] + row + [2])

expanded_matrix.append([2] * (cols + 2))
```

参考 更换方向

```
#
def generate_spiral_matrix(n):
    matrix = [[0] * n for _ in range(n)]

    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]
    current_direction = 0

    row, col = 0, 0
    for i in range(1, n * n + 1):
        matrix[row][col] = i
        next_row, next_col = row + directions[current_direction][0], col +
directions[current_direction][1]

        if 0 <= next_row < n and 0 <= next_col < n and matrix[next_row][next_col]
== 0:
            row, col = next_row, next_col
        else:
            current_direction = (current_direction + 1) % 4
            row, col = row + directions[current_direction][0], col +
directions[current_direction][1]

    return matrix
```

```
def print_matrix(matrix):
    for row in matrix:
        print(' '.join(map(str, row)))

n = int(input())
matrix = generate_spiral_matrix(n)
for row in matrix:
    print(' '.join(map(str, row)))
```

三角形路径dp

```
#
n = int(input())
triangle = []
for _ in range(n):
    triangle.append(list(map(int,input().split())))

def max_path_sum(triangle):
    n = len(triangle)
    dp = [[0] * n for _ in range(n)]
    dp[n-1] = triangle[n-1]

    for i in range(n-2, -1, -1):
        for j in range(i+1):
            dp[i][j] = max(dp[i+1][j], dp[i+1][j+1]) + triangle[i][j]

    return dp[0][0]

print(max_path_sum(triangle))
```

经典0-1背包问题

```
#
T, M = map(int, input().split())
dp = [0] * (T + 1)

for _ in range(M):
    time, value = map(int, input().split())
    for t in range(T, time - 1, -1):
        dp[t] = max(dp[t], dp[t - time] + value)

print(dp[T])
```

垃圾炸弹

```
d = int(input())
n = int(input())

city = [[0]*1025 for _ in range(1025)]

for _ in range(n):
    x,y,i = map(int,input().split())
    for p in range(max(x-d,0),min(x+d+1,1025)):
        for q in range(max(y-d,0),min(y+d+1,1025)):
```

```
            city[p][q]+=i

num = 0
maximum_blast = 0

for p in range(1025):
    for q in range(1025):
        if city[p][q]>maximum_blast:
            maximum_blast = city[p][q]
            num = 1
        elif city[p][q]==maximum_blast:
            num+=1

print(num,maximum_blast)
```

```
def dfs(x,y):
    if matrix[x][y]==9:
        print('yes')
        exit()
    for k in range(4):
        nx,ny=x+dx[k],y+dy[k]
        if 0<=nx<n and 0<=ny<n and (nx,ny) not in visited and matrix[nx][ny]!=1:
            if type==1:
                if nx==n-1 or matrix[nx+1][ny]==1:
                    continue
                if matrix[nx+1][ny]==9:
                    print('yes')
                    exit()
            else:
                if ny==n-1 or matrix[nx][ny+1]==1:
                    continue
                if matrix[nx][ny+1]==9:
                    print('yes')
                    exit()
            visited.add((nx,ny))
            dfs(nx,ny)
n=int(input())
dx=[1,0,0,-1]
dy=[0,1,-1,0]
matrix=[list(map(int,input().split())) for _ in range(n)]
visited=set()
for i in range(n):
    for j in range(n):
        if matrix[i][j]==5:
            if i<n-1 and matrix[i+1][j]==5:
                type=1
            else:
                type=2
            dfs(i,j)
            print('no')
            exit()
```

幸福的寒假生活dp

```
```

```
n = int(input())
act = [None]*61
for _ in range(n):
    s,e = map(int, input().split())
    if act[s]==None:
        act[s] = e
    elif act[s] > e:
        act[s] = e

dp = [1]*61
for i in range(61):
    if act[i]!=None:
        for j in range(i):
            if act[j]!=None and act[j]<i:
                dp[i] = max(dp[i], dp[j]+1)
print(max(dp))
```