cheet sheet

1. buff 叠起来

lru_cache（一定要加递归的def，不然报错）

```
from functools import lru_cache
@lru_cache(maxsize=128)  # 设置缓存大小为 128
#@lru_cache(maxsize=None)
```

最大递归深度 （这个随便用，没啥副作用，RE变TLE的小妙招）

```
import sys
sys.setrecursionlimit(10000)  # 设置最大递归深度为10000
```

2. 步入石器时代（有用函数）

defaultdict

```
from collections import defaultdict
aza = defaultdict(int)  # 默认为 0
boza = defaultdict(list)  # 默认为 []
ciza = defaultdict(set) # 默认为 set()
duza = defaultdict(str) # 默认为 ''
eza = defaultdict(float) # 默认为 0.0
fyza = defaultdict(dict) # 默认为 {}
Jetfire = defaultdict(tuple) # 默认为 ()
```

heap

```
import heapq
data = [3, 1, 4, 1, 5, 9, 2, 6]
heapq.heapify(data) # 变得部分有序
# 向堆中插入元素
heapq.heappush(data, 7)
# 弹出堆中最小的元素
min_element = heapq.heappop(data)
# 获取堆中最大的两个元素
largest_two = heapq.nlargest(2, data)
# 获取堆中最小的两个元素
smallest_two = heapq.nsmallest(2, data)
```

bisect

```
import bisect
arr = [1, 3, 5, 7, 9, 11, 13, 15, 17]
target = 9
insert_position_left = bisect.bisect_left(arr, target)
insert_position_right = bisect.bisect_right(arr, target)
print(f"目标元素 {target} 应该插入的位置(最左侧)是 {insert_position_left}")
print(f"目标元素 {target} 应该插入的位置(最右侧)是 {insert_position_right}")
insert_position = bisect.bisect(arr, target)
print(f"元素 {target} 应该插入的位置是：{insert_position}")
```

deque

```
from collections import deque
# 创建一个空的 deque 对象
d = deque()
# 在 deque 的右侧插入元素
d.append(1)
# 在 deque 的左侧插入元素
d.appendleft(0)
right_pop = d.pop()
# 从 deque 的左侧删除元素
left_pop = d.popleft()
```

其他

保留n位小数：
round(原数字，保留位数);
'%.nf'%原数字;
'{:.nf}'.format(原数字);
n位有效数字：
'%.ng'%原数字;
'{:.ng}'.format(原数字)

2.经典算法

排序、栈、队列

逆波兰表达式求值

```
stack=[]
for t in s:
    if t in '+-*/':
        b,a=stack.pop(),stack.pop()
        stack.append(str(eval(a+t+b)))
    else:
        stack.append(t)
print(f'{float(stack[0]):.6f}')
```

中序表达式转后序表达式

```
pre={'+':1,'-':1,'*':2,'/':2}
for _ in range(int(input())):
    expr=input()
    ans=[]; ops=[]
    for char in expr:
        if char.isdigit() or char=='.':
            ans.append(char)
        elif char=='(':
            ops.append(char)
        elif char==')':
            while ops and ops[-1]!='(':
                ans.append(ops.pop())
            ops.pop()
        else:
            while ops and ops[-1]!='(' and pre[ops[-1]]>=pre[char]:
                ans.append(ops.pop())
            ops.append(char)
    while ops:
        ans.append(ops.pop())
    print(''.join(ans))
```

最大全0子矩阵

```
for row in ma:
    stack=[]
    for i in range(n):
        h[i]=h[i]+1 if row[i]==0 else 0
        while stack and h[stack[-1]]>h[i]:
            y=h[stack.pop()]
            w=i if not stack else i-stack[-1]-1
            ans=max(ans,y*w)
        stack.append(i)
    while stack:
        y=h[stack.pop()]
        w=n if not stack else n-stack[-1]-1
        ans=max(ans,y*w)
print(ans)
```

求逆序对数

```
from bisect import *
a=[]
rev=0
for _ in range(n):
    num=int(input())
    rev+=bisect_left(a,num)
    insort_left(a,num)
ans=n*(n-1)//2-rev
```

```
def merge_sort(a):
    if len(a)<=1:
        return a,0
    mid=len(a)//2
    l,l_cnt=merge_sort(a[:mid])
    r,r_cnt=merge_sort(a[mid:])
    merged,merge_cnt=merge(l,r)
    return merged,l_cnt+r_cnt+merge_cnt
def merge(l,r):
    merged=[]
    l_idx,r_idx=0,0
    inverse_cnt=0
    while l_idx<len(l) and r_idx<len(r):
        if l[l_idx]<=r[r_idx]:
            merged.append(l[l_idx])
            l_idx+=1
        else:
            merged.append(r[r_idx])
            r_idx+=1
            inverse_cnt+=len(l)-l_idx
    merged.extend(l[l_idx:])
    merged.extend(r[r_idx:])
    return merged,inverse_cnt
```

树

根据前中序得后序、根据中后序得前序

```
def postorder(preorder,inorder):
    if not preorder:
        return ''
    root=preorder[0]
    idx=inorder.index(root)
    left=postorder(preorder[1:idx+1],inorder[:idx])
    right=postorder(preorder[idx+1:],inorder[idx+1:])
    return left+right+root
```

```
def preorder(inorder,postorder):
    if not inorder:
        return ''
    root=postorder[-1]
    idx=inorder.index(root)
    left=preorder(inorder[:idx],postorder[:idx])
    right=preorder(inorder[idx+1:],postorder[idx:-1])
    return root+left+right
```

层次遍历

```
from collections import deque
def levelorder(root):
    if not root:
        return ""
    q=deque([root])
    res=""
    while q:
        node=q.popleft()
        res+=node.val
        if node.left:
            q.append(node.left)
        if node.right:
            q.append(node.right)
    return res
```

解析括号嵌套表达式

```
def parse(s):
    node=Node(s[0])
    if len(s)==1:
        return node
    s=s[2:-1]; t=0; last=-1
    for i in range(len(s)):
        if s[i]=='(': t+=1
        elif s[i]==')': t-=1
        elif s[i]==',' and t==0:
            node.children.append(parse(s[last+1:i]))
            last=i
    node.children.append(parse(s[last+1:]))
    return node
```

二叉搜索树的构建

```
def insert(root,num):
    if not root:
        return Node(num)
    if num<root.val:
        root.left=insert(root.left,num)
    else:
        root.right=insert(root.right,num)
    return root
```

并查集

```
class UnionFind:
    def __init__(self,n):
        self.p=list(range(n))
        self.h=[0]*n
    def find(self,x):
        if self.p[x]!=x:
            self.p[x]=self.find(self.p[x])
        return self.p[x]
    def union(self,x,y):
        rootx=self.find(x)
        rooty=self.find(y)
        if rootx!=rooty:
            if self.h[rootx]<self.h[rooty]:
                self.p[rootx]=rooty
            elif self.h[rootx]>self.h[rooty]:
                self.p[rooty]=rootx
            else:
                self.p[rooty]=rootx
                self.h[rootx]+=1
```

字典树的构建

```
def insert(root,num):
    node=root
    for digit in num:
        if digit not in node.children:
            node.children[digit]=TrieNode()
        node=node.children[digit]
        node.cnt+=1
```

图

bfs

```
from collections import deque
def bfs(graph, start_node):
    queue = deque([start_node])
    visited = set()
    visited.add(start_node)
    while queue:
        current_node = queue.popleft()
        for neighbor in graph[current_node]:
            if neighbor not in visited:
                visited.add(neighbor)
                queue.append(neighbor)
```

棋盘问题（回溯法）

```
def dfs(row, k):
    if k == 0:
        return 1
    if row == n:
        return 0
    count = 0
    for col in range(n):
        if board[row][col] == '#' and not col_occupied[col]:
            col_occupied[col] = True
            count += dfs(row + 1, k - 1)
            col_occupied[col] = False
    count += dfs(row + 1, k)
    return count
col_occupied = [False] * n
print(dfs(0, k))
```

dijkstra

```python
# 1.使用vis集合
def dijkstra(start,end):
    heap=[(0,start,[start])]
    vis=set()
    while heap:
        (cost,u,path)=heappop(heap)
        if u in vis: continue
        vis.add(u)
        if u==end: return (cost,path)
        for v in graph[u]:
            if v not in vis:
                heappush(heap,(cost+graph[u][v],v,path+[v]))
# 2.使用dist数组
import heapq
def dijkstra(graph, start):
    distances = {node: float('inf') for node in graph}
    distances[start] = 0
    priority_queue = [(0, start)]
    while priority_queue:
        current_distance, current_node = heapq.heappop(priority_queue)
        if current_distance > distances[current_node]:
            continue
        for neighbor, weight in graph[current_node].items():
            distance = current_distance + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(priority_queue, (distance, neighbor))
    return distances
```

kruskal

```python
uf=UnionFind(n)
edges.sort()
ans=0
for w,u,v in edges:
    if uf.union(u,v):
        ans+=w
print(ans)
```

prim

```
vis=[0]*n
q=[(0,0)]
ans=0
while q:
    w,u=heappop(q)
    if vis[u]:
        continue
    ans+=w
    vis[u]=1
    for v in range(n):
        if not vis[v] and graph[u][v]!=-1:
            heappush(q,(graph[u][v],v))
print(ans)
```

拓扑排序

```
from collections import deque
def topo_sort(graph):
    in_degree={u:0 for u in graph}
    for u in graph:
        for v in graph[u]:
            in_degree[v]+=1
    q=deque([u for u in in_degree if in_degree[u]==0])
    topo_order=[]
    while q:
        u=q.popleft()
        topo_order.append(u)
        for v in graph[u]:
            in_degree[v]-=1
            if in_degree[v]==0:
                q.append(v)
    if len(topo_order)!=len(graph):
        return []
    return topo_order
```

3.经典题目

道路

```
import heapq
from collections import defaultdict

MAX_COINS = int(input())  # 最大金币数
CITY_COUNT = int(input())  # 城市数目
ROAD_COUNT = int(input())

# 存储道路信息的字典，使用 defaultdict 初始化
roads = defaultdict(list)

for _ in range(ROAD_COUNT):
    start, end, length, money = map(int, input().split())
    start, end = start - 1, end - 1
    roads[start].append((end, length, money))

def bfs(start, end, max_coins):
    queue = [(0, max_coins, start)]  # (距离, 剩余金币, 当前城市)
    visited = set()
    while queue:
        distance, coins, city = heapq.heappop(queue)
        if city == end:
            return distance
        visited.add((city, coins))
        for next_city, road_length, road_money in roads[city]:
            if coins >= road_money:
                new_distance = distance + road_length
                if (next_city, coins - road_money) not in visited:
                    heapq.heappush(queue, (new_distance, coins - road_money,
next_city))
    return -1
print(bfs(0, CITY_COUNT - 1, MAX_COINS))
```

单调栈

```
def find_left_greater(nums):
    stack = []
    result = [-1] * len(nums)
    for i in range(len(nums)):
        while stack and nums[stack[-1]] < nums[i]: result[stack.pop()] = nums[i]
        stack.append(i)
    return result
```

单调队列

```
# 使用示例
nums = [2, 1, 4, 3, 5]
print(find_left_greater(nums))  # 输出: [-1, -1, 2, 2, 4]
from collections import deque
def max_in_window(nums, k):
    queue = deque()
    result = []
    for i in range(len(nums)):
        # 将队列中比当前元素小的都弹出
        while queue and nums[queue[-1]] < nums[i]: queue.pop()
        queue.append(i)
        # 如果窗口左边界已经不在队列中了,则将其从队列中移除
        if queue[0] == i - k: queue.popleft()
        # 如果窗口大小达到k,则将队列头部元素(即窗口内最大值)加入结果
        if i >= k - 1: result.append(nums[queue[0]])
    return result
# 使用示例
nums = [1, 3, -1, -3, 5, 3, 6, 7]
print(max_in_window(nums, 3))  # 输出: [3, 3, 5, 5, 6, 7]
```

舰队海域出击!

```
print('Let it go!!!!!!')
```