欧拉筛:

```
def Sieve_of_Euler(n):
    is_prime=[True]*(n+1)
    primes=[ ]
    for i in range(2,n+1):
        if is_prime[i]:
            primes.append(i)
        for j in range(len(primes)):
            if primes[j]*i>n:
                break
            is_prime[primes[j]*i]=False
            if i%primes[j]==0:
                break
    return primes
```

埃氏筛:

```
def Sieve_of_Eratosthenes(n):
    is_prime=[True]*(n+1)
        for i in range(2,int(n**0.5)+1):
            if is_prime[i]:
                for j in range(i*i,n+1,i):
                    is_prime[j]=False
    primes=[i for i in range(2,n+1) if is_prime[i]]
    return primes
```

注: 质数, 平方数, 立方数, 数列mod n余数的周期性, 若超时/超内存, 可借助程序先print出来!

二分查找:

```
def binary_search(nums,target):
    left,right=0,len(nums)
    while left<right:
        mid=(left+right)//2
        if nums[mid]<target:
            left=mid
        else:
            right=mid-1
    return left
```

注: 二分查找的代码中3.5.7.8行涉及加减1的问题. 应具体问题具体分析. target为题目目标有时更复杂

DFS: "图" "坐标" "已访问"

```
def dfs(board,x,y,visited):
    if x,y越界:
        return
    elif visited[a][b] or 不符合题目要求:
      (或者(a,b) in visited)
        return
    visited[a][b]=True (或者visited.add((a,b)))
    "初始化"&根据题意进行相关操作
    for dx in [-1,0,1]:
        for dy in [-1,0,1]:
            if dx==dy==0:
                continue
            nx,ny=dx+x,dy+y
            dfs(board,nx,ny,visited)
```
{"邻居"为8个,可根据需要改为4个}

```
board=读取输入
visited=[[False]*M for i in range(N)]
(或者visited=set( ))
x,y=据题目要求选择起点.
```

注: 模板的DFS和BFS均为二维. 若为一维则不用visited, 改为是否访问该节点形成分叉: return 访问 or 不访问

BFS:

```
from collections import deque
def bfs(起点):
    queue=deque([起点])
    visited=set( )
    while queue:
        x,y=queue.popleft( )
        if (x,y) in visited:
            continue
        visited.add((x,y))
        根据题意进行相关操作
        directions=[(0,1),(0,-1),(1,0),(-1,0)]
        for i,(dx,dy) in enumerate(directions):
            nx,ny=x+dx,y+dy
            (i可用于判断前进方向是否一致)
            if nx,ny越界or (nx,ny) in visited:
                continue
            if 不符合题目要求
                continue
            visited.add((nx,ny))
            queue.append((nx,ny))
```

(用堆也行. 具体情况具体分析) (要输出的东西)

注: 使用函数与递归时, 注意副本创建, eg. 应是append(列表[:])

矩阵越界问题：
套保护圈 或者 min( ). max( ) 坐标边界

堆：
import heapq
heap=[ ]
heapq.heappush(heap,元素) #将元素加入heap中
heapq.heappop(heap) #弹出heap中最小数
heapq.heapify(列表) #将列表转换成堆

堆排序
```
def heap_sort(nums):
    heapq.heapify(nums)
    n=len(nums)
    for i in range(n):
        nums[i]=heapq.heappop(nums)
    return nums
```

冒泡排序：
```
def bubble_sort(arr):
    n=len(arr)
    for i in range(n):
        for j in range(n-i-1):
            if arr[j]>arr[j+1]:
                arr[j],arr[j+1]=arr[j+1],arr[j]
    return arr
```

约瑟夫问题：
```
def monkey_king(n,m): n为总数,m为报数出列
    if n==1:
        return 1
    else:
        x=0
        for i in range(2,n+1):
            x=(x+m)%i
        return x+1
```

字典：
empty_dict={} #创建空字典
dict.get(a,b) #在dict中访问键a的值若没有键a则将b赋给a.
dict[键]=新值
for i in dict.keys( ) 遍历字典中的键
for 键,值 in dict.items( ) 遍历字典中的键,值

集合：
empty_set=set( ) #创建空集合
set.add( ) #添加元素
set1|set2 并集
set1&set2 交集
set1-set2 差集

保留n位小数.
print("{:.nf}".format(变量))

from collections import defaultdict
字典名=defaultdict(int/list)

bisect.bisect_left(a,x)
a:已排序的序列 (最长上升子序列)
x:要插入的元素
返回值为x在已排序序列a中的时插入位置(多个相等返回最左边)

dp问题：
①简单的形如斐波那契数列：
类比有递推关系的高中相关问题,列状态转移方程
②二维dp：数字三角形/矩阵/最长公共子序列/回文序列(用[::-1]倒转)
注：若递归次数过多,可用lru_cache( )
若发现有重复计算,可用memo={ }储存
区分是子串(必须连续)还是子序列
③背包问题：
1° 01背包 最大容量p. 物品数量n (取或不取) lists=[[w,v]*n] w重量 v价值
```
dp=[0]*(p+1)
for i in range(p+1):
    for j in range(p,-1,-1): #必须逆序
        if lists[i-1][0]<=j:
            dp[j]=max(dp[j],dp[j-lists[i-1][0]]+...)
```
dp[p]答案

2° 完全背包(可无限取)

m:最大容量 n:数量

weight = [0, w1, ..., wn]

value = [0, v1, ..., vn].

dp = [0]*(m+1)

for i in range(1, n+1):

  for j in range(1, m+1): #正序

    if w[i]<=j:

      dp[j] = max(dp[j], dp[j-w[i]]+c[i])

注:如果w[i]与c[i]可用直接表达,则for循环可直接修改

3° 多重背包(可有限取)

n:数量. m:最大容量

dp = [0]*(m+1)

for _ in range(n):

  weight, value, count = ...

  t = 1

  while c>=t:

    for j in range(m, w*t-1, -1)

      dp[j] = max(dp[j], dp[j-w*t]+t*v)

    c -= t

    t *= 2 #二进制,省时

  if c:

    for j in range(m, w*c-1, -1):

      dp[j] = max(dp[j], dp[j-c*w]+c*v)

4° 二维限制

dp[j][k] = max(dp[j][k], dp[j-w[i]][k-b[i]]+c[i])

5° "恰好装满":除第一行第一列,其余初始值设成-inf(float)

5° 方案数:max→sum,去掉"+v[i]"

第一行第一列初始化为1.

7° 追踪具体的:dp = [[0]*(最大容量+1) for _ in range(数量+1)]

flag同上.

if j>=w[i] dp[i][j]更新同时flag[i][j] = { flag[i-1][j]不放进去 / flag[i-1]放进去

sol = [0]*数量, while flag[n][w]!=0:

  temp = flag[n][w]

  sol[temp-1] = 1.

  w -= w[temp] (weight:列表)

  n = temp-1

八皇后问题:

ans = []

def dfs(result=[], i=0, row=[], diag_1=set(), diag_2=set()):

  if i==8:

    ans.append(result)

    return

  for j in range(8):

    if j not in row and i+j not in diag_1 and i-j not in diag_2:

      dfs(result+[j+1], i+1, row+[j], diag_1|{i+j}, diag_2|{i-j})

2022决战双十一生成购买方案:

def dfs(n, goods, nums, plan, plans):

  if num==n+1:

    plans.append(plan[:])

    return

  for j in goods[num].keys():

    plan.append(j)

    dfs(...)

    plan.pop

  return.

如果列表为空,考虑以下几点:

①global; ②创建副本,如return. [:].

③实在不行换方法