sheet paper

最简单的递归（求阶乘）

```
def jie_cheng(num)
    if num == 1:
        return 1
    else:
        return num *jie_cheng(num-1)
print(jie_cheng(n))
```

```
def is_leap_year(year):
    # 判断是否为闰年的逻辑
    if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0 and year % 3200 !=
0):
        return "Y"
    else:
        return "N"

# 读取输入
a = int(input())

# 输出结果
print(is_leap_year(a))
```

```
def min_max_animals(a):
    # 最少动物数：尽可能多的兔子
    min_rabbits = a // 4
    min_animals = min_rabbits + (a % 4) // 2
    if a % 4 != 0 and a % 4 != 2:
        min_animals = 0  # 如果脚的总数不能被2整除，则没有满足条件的答案

    # 最多动物数：尽可能多的鸡
    max_chickens = a // 2
    max_animals = max_chickens + (a % 2) // 4
    if a % 2 != 0:
        max_animals = 0  # 如果脚的总数是奇数，则没有满足条件的答案

    return min_animals, max_animals

# 读取输入
a = int(input())

# 计算最少和最多动物数
min_animals, max_animals = min_max_animals(a)

# 输出结果
print(min_animals, max_animals)
```

```
m, n = map(int, input().split())
print(m * n // 2)
```

```
s1 = input().lower()
s2 = input().lower()
if s1 < s2:
    print("-1")
elif s1 > s2:
    print("1")
else:
    print("0")
```

```
s1 = input().lower()
s2 = input().lower()
if s1 < s2:
    print("-1")
elif s1 > s2:
    print("1")
else:
    print("0")
```

```
import math
print(math.sqrt(4))#平方根
print(math.pi)#圆周率
from math import sqrt, pi
```

```
import math

while True:
    n = float(input())
    if math.isclose(n, 0.00, rel_tol=1e-5) :#浮点数与0比较需要 用
math.isclose(....)，而不能直接用 == 0.00判断
        break

    cnt = 0
    tot = 0
    while  True:
        cnt += 1
        tot += 1/(1+cnt)
        if tot>n:
            break

    print(cnt, "card(s)")
```

为了更准确地进行浮点数的比较，可以使用`math.isclose()`函数或自定义的比较函数，这些函数允许你指定一个容差值来比较浮点数的接近程度。

100以内的完全平方数对应的开关是开的。即，开关编号为1, 4, 9, 16, 25, 36, 49, 64, 81, 100的开关是开的。

```
for i in range(int(input())):
    print(int(int(input())**0.5))
```

计算机思维

```
for _ in range(int(input())):
    n = int(input())
    lst = [0]*n

    for j in range(2,n+1):
        for i in range(j-1,n,j):
            lst[i] ^= 1

    print(lst.count(0))
```

第二种类型题

```
n = int(input("请输入开关的总数n: "))
for num in range(1, n + 1):
    square_root = int(num ** 0.5)
    if square_root ** 2 == num:
        print(num, end=' ')
```

```
n = int(input())  # 获取事件的总数量
events = list(map(int, input().split()))  # 获取表示事件的整数列表，用空格隔开
free_officers = 0  # 初始时空闲警员数量为0
untreated_crimes = 0  # 初始时未处理犯罪数量为0
for num in events:  # 遍历每个事件
    if num == -1:  # 如果事件代表发生犯罪
        if free_officers == 0:  # 检查是否有空闲警员
            untreated_crimes += 1  # 若没有空闲警员，未处理犯罪数量加1
        else:
            free_officers -= 1  # 若有空闲警员，空闲警员数量减1，表示该警员去处理犯罪了
    else:
        free_officers += num  # 如果事件代表招募警员，增加空闲警员数量
print(untreated_crimes)  # 输出未处理犯罪的数量
```

```
t = int(input())  # 获取测试用例的数量
for _ in range(t):
    a, b = map(int, input().split())  # 解析每个测试用例中的两个整数a和b
    remainder = a % b  # 计算a除以b的余数
    if remainder == 0:  # 如果余数为0，说明a已经能被b整除，不需要操作
        print(0)
    else:
        print(b - remainder)  # 否则，需要将a增加到能被b整除，增加的量就是b减去余数
```

```
matrix = []
for _ in range(5):
    row = list(map(int, input().split()))
    matrix.append(row)

# 找到数字1所在的行索引和列索引
row_index = -1
col_index = -1
for i in range(5):
    for j in range(5):
        if matrix[i][j] == 1:
            row_index = i
            col_index = j
```

```
            break
    if row_index!= -1:
        break

# 计算行方向需要的交换次数
row_moves = abs(row_index - 2)
# 计算列方向需要的交换次数
col_moves = abs(col_index - 2)

print(row_moves + col_moves)
```

```
m, n = map(int, input().split())
print(m * n // 2)
```

```
n = int(input())
count = 0
for _ in range(n):
    petya, vasya, tonya = map(int, input().split())
    if petya + vasya + tonya >= 2:
        count += 1
print(count)
```

```
s1 = input().lower()
s2 = input().lower()
if s1 < s2:
    print("-1")
elif s1 > s2:
    print("1")
else:
    print("0")
```

```
n, m, a = map(int, input().split())
# 计算沿着长度方向需要的石板数量，向上取整
x = (n + a - 1) // a
# 计算沿着宽度方向需要的石板数量，向上取整
y = (m + a - 1) // a
print(x * y)
```

```
import math

while True:
    n = int(input())
    if n == 0:
        break

    max_time = float("inf")
    for _ in range(n):
        speed, time = map(int, input().split())
        if time < 0:
            continue
        arrival_time = math.ceil((4500 / speed) * 3.6 + time)
        max_time = min(max_time, arrival_time)
```

```
    print(max_time)
```

```
s = input()
gap = ord('a') - ord('A')

ans = []
for i in s:
    if 'A' <= i <= 'Z':
        ans += chr(ord(i) + gap)
    elif 'a' <= i <= 'z':
        ans += chr(ord(i) - gap)
    else:
        ans += i

print(''.join(ans))
```

```
def check(num):
    if num%7 == 0:
        return True

    for i in str(num):
        if i=='7':
            return True

    return False

n = int(input())
a = []
for i in range(n+1):
    if check(i) == False:
        a.append(i)

print(sum(i**2 for i in a))
```

```
import math
n = int(input())
for i in range(n):
    a, b, c = map(float, input().split())
    if b == 0:
        b = -b
    delta = b ** 2 - 4 * a * c
    if delta > 0:
        x1 = (-b + math.sqrt(delta)) / (2 * a)
        x2 = (-b - math.sqrt(delta)) / (2 * a)
        x1 = format(x1, ".5f")
        x2 = format(x2, ".5f")
        print(f"x1={x1};x2={x2}")
    elif delta == 0:
        t = (-b) / (2 * a)
        x1 = format(t, ".5f")
        print(f"x1=x2={x1}")
    else:
        d = format(math.sqrt(-delta) / (2 * a), ".5f")
```

```
        re = format((-b) / (2 * a), ".5f")
        print(f"x1={re}+{d}i;x2={re}-{d}i")
```

```
while True:#约瑟夫问题
    n, m = map(int, input().split())
    if n + m == 0:
        break
    a = 1  # 初始化 a 为 0，表示从 0 开始编号
    for i in range(2, n + 1):
        a = (a + m - 1) % i + 1
    print(a)  # 最终结果需要加 1，因为编号从 1 开始
```

02773: 采药

dp, http://cs101.openjudge.cn/practice/02773

辰辰是个很有潜能、天资聪颖的孩子，他的梦想是称为世界上最伟大的医师。为此，他想拜附近最有威望的医师为师。医师为了判断他的资质，给他出了一个难题。医师把他带到个到处都是草药的山洞里对他说：“孩子，这个山洞里有一些不同的草药，采每一株都需要一些时间，每一株也有它自身的价值。我会给你一段时间，在这段时间里，你可以采到一些草药。如果你是一个聪明的孩子，你应该可以让采到的草药的总价值最大。”

如果你是辰辰，你能完成这个任务吗?

输入

输入的第一行有两个整数T（1 <= T <= 1000）和M（1 <= M <= 100），T代表总共能够用来采药的时间，M代表山洞里的草药的数目。接下来的M行每行包括两个在1到100之间（包括1和100）的的整数，分别表示采摘某株草药的时间和这株草药的价值。

输出

输出只包括一行，这一行只包含一个整数，表示在规定的时间内，可以采到的草药的最大总价值。

使用 **二维数组** 和 **递归 + LRU Cache** 来解决问题。

1. 使用 二维数组 的方法：

如果你使用二维数组的动态规划方法，可以将状态转移视为 **二维的背包问题**。我们使用一个 `dp[i][j]` 数组来表示在前 `i` 个草药中，使用不超过 `j` 时间的最大价值。

动态规划转移方程：

- `dp[i][j]` 表示使用前 `i` 个草药，且时间不超过 `j` 时的最大价值。
- 对于每个草药

  ```
  (time, value)
  ```

  ，有两种选择：
  - 不采摘它，`dp[i][j] = dp[i-1][j]`。
  - 采摘它，`dp[i][j] = dp[i-1][j-time] + value`，前提是 `j >= time`。

二维数组实现：

```python
def max_value_with_2d_array(T, M, herbs):
    # dp[i][j]表示前i个草药，时间不超过j时的最大价值
    dp = [[0] * (T + 1) for _ in range(M + 1)]

    for i in range(1, M + 1):
        time, value = herbs[i - 1]
        for j in range(T + 1):
            # 不采摘当前草药
            dp[i][j] = dp[i - 1][j]
            # 采摘当前草药，前提是时间允许
            if j >= time:
                dp[i][j] = max(dp[i][j], dp[i - 1][j - time] + value)

    return dp[M][T]

# 输入读取
T, M = map(int, input().split())  # 读取T和M
herbs = []

for _ in range(M):
    time, value = map(int, input().split())
    herbs.append((time, value))

# 计算并输出结果
print(max_value_with_2d_array(T, M, herbs))
```

说明：

1. `dp[i][j]`：表示前 `i` 株草药，在总时间 `j` 内能获得的最大价值。
2. 如果不采摘第 `i` 株草药，`dp[i][j]` 就等于 `dp[i-1][j]`。
3. 如果采摘第 `i` 株草药，且时间 `j` 足够，`dp[i][j]` 就等于 `dp[i-1][j-time] + value`。

最终的结果就是 `dp[M][T]`，即前 `M` 株草药，在总时间 `T` 内可以获得的最大价值。

时间复杂度：

- **O(M * T)**，其中 `M` 是草药的数量，`T` 是最大时间。

---

2. 使用 递归 + LRU Cache 方法：

递归 + **LRU Cache**（最少最近使用缓存）是一种更灵活的解法，可以通过缓存中间计算的结果来避免重复计算。我们可以通过递归的方式，分别计算在某一时间限制下，是否选择某个草药，并且将结果缓存，以便重复使用。

递归思路：

- 对于每一株草药

  ```
  (time, value)
  ```

  ，我们有两种选择：

1. **不选择** 这株草药，计算剩余时间内能获得的最大价值。
2. **选择** 这株草药，计算剩余时间内能获得的最大价值。

递归的基本形式是 `max_value(time_left, idx)`，表示从第 `idx` 株草药开始，在剩余时间 `time_left` 内可以获得的最大价值。

递归 + LRU Cache 实现：

```
from functools import lru_cache

def max_value_with_recursion(T, M, herbs):
    @lru_cache(None)
    def helper(time_left, idx):
        if idx == M:  # 如果已经考虑完所有草药
            return 0
        time, value = herbs[idx]

        # 选择不采摘这株草药
        max_val = helper(time_left, idx + 1)

        # 选择采摘这株草药（前提是时间足够）
        if time_left >= time:
            max_val = max(max_val, helper(time_left - time, idx + 1) + value)

        return max_val

    return helper(T, 0)

# 输入读取
T, M = map(int, input().split())  # 读取T和M
herbs = []

for _ in range(M):
    time, value = map(int, input().split())
    herbs.append((time, value))

# 计算并输出结果
print(max_value_with_recursion(T, M, herbs))
```

说明：

1. `helper(time_left, idx)`：从草药 `idx` 开始，剩余时间为 `time_left`，返回能获得的最大价值。
2. **缓存机制**：使用 `@lru_cache(None)` 来缓存函数的计算结果，避免重复计算相同状态。
3. 如果剩余时间不够，就跳过当前草药，递归处理下一个草药；如果可以采摘，则递归计算剩余时间内能获得的最大价值。

时间复杂度：

- **O(M * T)**，其中 `M` 是草药的数量，`T` 是最大时间，虽然有递归调用，但缓存机制使得每个状态只会被计算一次。

总结：

- **二维数组方法**：适合传统的动态规划，使用 `dp[i][j]` 来表示前 `i` 个草药，使用不超过 `j` 时间的最大价值。
- **递归 + LRU Cache 方法**：适合用递归思想求解，并通过缓存机制避免重复计算，适合不需要显式构造 DP 数组的情况。

对于这个问题，使用动态规划是最常见的解法，而递归 + LRU Cache 更加灵活，适合递归式问题的求解。

门口的树：输入的第一行有两个整数L（1 <= L <= 10000）和 M（1 <= M <= 100），L代表马路的长度，M代表区域的数目，L和M之间用一个空格隔开。接下来的M行每行包含两个不同的整数，用一个空格隔开，表示一个区域的起始点和终止点的坐标。

输出

输出包括一行，这一行只包含一个整数，表示马路上剩余的树的数目。

```
L, m = map(int, input().split())

dp = [1]*(L+1)

for i in range(m):
    s, e = map(int, input().split())
    for j in range(s, e+1):
        dp[j] = 0

print(dp.count(1))
```

使用while循环输出1，2，3，4，5，6，7，8，9，10

```
a=1
while a<11:
    print(a,end="")
    a+=1
求1--100的所有数的和
a=1
sum=0
while a<101:
    sum+=a
    a+=1
print（“sum=%d"%sum)

numbers=[23,24,2,6,55,89,100]
jishu=[]
oushu=[]

while numbers:
    num=numbers.pop()
    if (num%2==0):
        oushu.append(num)
    else:
        jishu.append(num)
print(jishu)
print(oushu)
```

```
numbers=[23,24,2,6,55,89,100]
jishu=[]
oushu=[]

for num in numbers:
    if num%2==0:
        oushu.append(num)
    else:
        jishu.append()
print(jishu)
print(oushu)

def add(x,y):
    return x+y
def max(x,y):
    """取两个数的最大值"""
    if x > y:
        return x
    else:
        return y
sum=add(1,2)
print(sum)

m=max(4,6)
print(m)
```

定义不同类型的变量 num = 10 # 整数 float_num = 3.14 # 浮点数 string = "Hello, World" # 字符串 is_true = True # 布尔值 list_data = [1, 2, 3] # 列表 tuple_data = (4, 5, 6) # 元组 set_data = {7, 8, 9} # 集合 dict_data = {"key": "value"} # 字典 # 类型转换 new_num = float(num) # 整数转浮点数 new_str = str(num) # 整数转字符串

import math # 使用 math 模块中的函数，比如计算平方根 square_root = math.sqrt(16) print("16的平方根是：", square_root)