**ĐỀ CƯƠNG ÔN KIỂM TRA THỰC HÀNH**

Cấu trúc

import os

os.system('cls' if os.name == 'nt' else 'clear')

Mục lục

[**CVT Giải phương trình đệ qui dạng tổng quát với hàm tiến triển là hàm nhân** 1](#_Toc139466249)

[**Xây dựng lớp Danhsach – Code ok** 3](#_Toc139466250)

[**Xây dựng một lớp trong ngôn ngữ lập trình của bạn lựa chọn để quản lý một danh sách liên kết các số nguyên. Trong lớp này, bạn phải xây dựng các phương thức cho việc thêm, tìm kiếm, sửa, xóa và xuất các phần tử.** 4](#_Toc139466251)

[Merge Sort (Class Sapxep kethua Danhsach) - Ok 6](#_Toc139466252)

[**CÁC PHƯƠNG PHÁP SẮP XẾP THUỘC LỚP HÀM N2** 7](#_Toc139466253)

[Phương pháp chọn (Selection Sort) (Class Sapxep\_N^2 kethua Danhsach) 7](#_Toc139466254)

[Phương pháp chèn (Insertion Sort) (Class Sapxep\_N^2 kethua Danhsach) 8](#_Toc139466255)

[Phương pháp nổi bọt (Bubble Sort) (Class Sapxep\_N^2 kethua Danhsach) 9](#_Toc139466256)

[**CÁC PHƯƠNG PHÁP SẮP XẾP THUỘC LỚP HÀM NLogN** 10](#_Toc139466257)

[Phương pháp Heap Sort (Class Sapxep\_NLogN kethua Danhsach) - Ok 10](#_Toc139466258)

[Phương pháp Quick Sort (Class Sapxep\_NLogN kethua Danhsach) - Ok 11](#_Toc139466259)

[**CÁC PHƯƠNG PHÁP SẮP XẾP THUỘC LỚP HÀM N** 12](#_Toc139466260)

[Phương pháp Radix Sort (Class Sapxep\_N kethua Danhsach) - Ok 12](#_Toc139466261)

[Phương pháp Bin Sort (Class Sapxep\_N kethua Danhsach) 13](#_Toc139466262)

[Trường hợp đơn giản 13](#_Toc139466263)

[Trường hợp tổng quát 13](#_Toc139466264)

[Trường hợp tập giá trị lớn 13](#_Toc139466265)

[**CÁC GIẢI THUẬT TÌM KIẾM NỘI (Xây dựng lớp TimKiem kế thừa từ lớp DanhSach)** 15](#_Toc139466266)

[Tìm kiếm tuyến tính (Linear Search) 15](#_Toc139466267)

[Không đệ qui 15](#_Toc139466268)

[Đệ qui - Ok 15](#_Toc139466269)

[Tìm kiếm nhị phân (Binary Search) 16](#_Toc139466270)

[Không đệ qui 16](#_Toc139466271)

[Đệ qui 16](#_Toc139466272)

[**KỸ THUẬT CHIA ĐỂ TRỊ** 17](#_Toc139466273)

[Bài toán nhân các số nguyên lớn 17](#_Toc139466274)

[Thuật toán thông thường 17](#_Toc139466275)

[Thuật toán áp dụng kỹ thuật chia để trị 17](#_Toc139466276)

[Nhân hai ma trận: 18](#_Toc139466277)

[Viết một chương trình cho phép nhân hai ma trận vuông cấp N lớn sử dụng thuật toán chia để trị. 18](#_Toc139466278)

[Nhân hai ma trận nhiều giá trị: 19](#_Toc139466279)

[Tìm UCLN và BCNN sử dụng kỹ thuật chia để trị: 22](#_Toc139466280)

[Tìm giá trị nhỏ nhất trong ma trận vuông cấp N bằng kỹ thuật chia để trị 23](#_Toc139466281)

[TH1: Đệ qui, chia hàng 23](#_Toc139466282)

[Trường hợp nhận số chẵn 23](#_Toc139466283)

[Có trường hợp nếu nhập số lẻ, chẵn: 23](#_Toc139466284)

[TH2 (Khó): Đệ qui, Chia thành 4 ô vuông 25](#_Toc139466285)

[Trường hợp số chẵn 25](#_Toc139466286)

[Trường hợp số lẽ, chẵn: 25](#_Toc139466287)

[**KỸ THUẬT THAM ĂN** 27](#_Toc139466288)

[Thuật toán thông thường 27](#_Toc139466289)

[Thuật toán áp dụng kỹ thuật tham ăn 27](#_Toc139466290)

[ATM: 29](#_Toc139466291)

[-----Không giới hạn số tờ----- 29](#_Toc139466292)

[----Giới hạn số tờ là 1----- 30](#_Toc139466293)

[-----Số tờ tùy chọn ----- 31](#_Toc139466294)

[Bài toán tìm đường đi ngắn nhất (TSP) của người giao hàng 32](#_Toc139466295)

[Bài toán cá ba lô (knapsack) - Kỹ thuật tham ăn 33](#_Toc139466296)

[Cái ba lô: 34](#_Toc139466297)

[----Không giới hạn----- 34](#_Toc139466298)

[-------Tùy chọn theo số lượng----- 35](#_Toc139466299)

[-----Mỗi đồ vật chỉ chọn 1--- 36](#_Toc139466300)

[**KỸ THUẬT QUI HOẠCH ĐỘNG (Dynamic programming)** 37](#_Toc139466301)

[Tính số tổ hợp 37](#_Toc139466302)

[Bằng thuật toán đệ qui, áp dụng kỹ thuật qui hoạch động với CTDL là bảng 37](#_Toc139466303)

[Week7\_Baitap2\_Mang2chieu: 37](#_Toc139466304)

[Bằng thuật toán đệ qui, áp dụng kỹ thuật qui hoạch động với CTDL là vector 38](#_Toc139466305)

[Week7\_Baitap2\_Mang1Chieu: 38](#_Toc139466306)

[Bài toán cái ba lô - Week7\_Baitap3: 39](#_Toc139466307)

[TH1: Chọn đồ vắn có đơn giá cao nhất cho đến khi không thể chọn thêm đồ vật nào class Do\_vat: 40](#_Toc139466308)

[TH2: Có số lượng 40](#_Toc139466309)

[TH3: Chọn tối đa một đồ vật của từng loại 41](#_Toc139466310)

[**KỸ THUẬT QUAY LUI** 43](#_Toc139466311)

[Định trị cây biểu thức số học 43](#_Toc139466312)

[Mô phỏng trò chơi ca rô 44](#_Toc139466313)

[Cắt tỉa Alpha – Beta 45](#_Toc139466314)

[Vét cạn định trị: 48](#_Toc139466315)

[**KỸ THUẬT TÌM KIẾM ĐỊA PHƯƠNG** 50](#_Toc139466316)

[Tìm số nguyên tố 50](#_Toc139466317)

[Bài toán cây phủ tối thiểu Minimum Spanning Tree (MST) 51](#_Toc139466318)

[Bài toán tìm đường đi ngắn nhất (TSP) của người giao hàng 53](#_Toc139466319)

[**TẬP TIN** 54](#_Toc139466320)

[TẬP TIN TUẦN TỰ 54](#_Toc139466321)

[VCT mô phỏng CTDL và các phép toán cơ bản như: tìm, thêm, xóa, sửa cho 54](#_Toc139466322)

[tập tin tuần tự. 54](#_Toc139466323)

[TẬP TIN BẢN BĂM 55](#_Toc139466324)

[VCT mô phỏng CTDL và các phép toán cơ bản như: tìm, thêm, xóa, sửa cho tập tin băm. 55](#_Toc139466325)

[**ĐỀ THAM KHẢO HỌC THÊM** 56](#_Toc139466326)

[Xây dựng lớp HinhHoc: 56](#_Toc139466327)

[Tìm ước chung lớn nhất bằng thuật toán Euclid: 56](#_Toc139466328)

[Tìm các dãy con liên tiếp có tổng lớn nhất trong mảng: 56](#_Toc139466329)

[Sắp xếp Shell Sort (Class Sapxep\_TheoKhoangCach kế thừa từ Danhsach): 57](#_Toc139466330)

[Đếm số lượng đảo ngược trong mảng: 57](#_Toc139466331)

[Bài toán N-Queens sử dụng kỹ thuật quay lui: 57](#_Toc139466332)

[Áp dụng kỹ thuật tham lam để tối ưu hóa việc sử dụng tài nguyên máy tính: 57](#_Toc139466333)

[Phương pháp tìm kiếm Interpolation Search: 58](#_Toc139466334)

[Cài đặt B-Tree và thực hiện các thao tác cơ bản: 58](#_Toc139466335)

[Tìm chu trình Hamiltonian trong đồ thị: 58](#_Toc139466336)

# **CVT Giải phương trình đệ qui dạng tổng quát với hàm tiến triển là hàm nhân**

import math

def G\_PT\_DQ\_TQ\_HamNhan(a, b, d\_func, n):

def f(b, n):

return b\*\*n

def ktraHamNhan(d\_func, m, n):

return d\_func(m\*n) == d\_func(m) \* d\_func(n)

def case\_1():

return n \*\* (math.log(a, b))

def case\_2():

return n \*\* (math.log(d\_func(b), b))

def case\_3():

return n \*\* (math.log(a, b)) \* math.log(n, b)

if ktraHamNhan(d\_func, a, b):

d\_b = d\_func(b)

if a > d\_b:

return case\_1()

elif a < d\_b:

return case\_2()

else:

return case\_3()

else:

raise ValueError("d(n) không phải là hàm nhân. Không thể giải phương trình.")

# Ví dụ sử dụng

def d\_func(n):

return n \*\* 2 # Hàm tiến triển d(n) = n^2

try:

a = int(input("Nhâp a = "))

b = int(input("Nhập b = "))

n = int(input("Nhập số mũ của n = "))

result = G\_PT\_DQ\_TQ\_HamNhan(a, b, d\_func, n)

print(f"T({n}) = {result}")

except ValueError as e:

print(e)

**-- BaoThang - Ok**

import math

def f(b, n):

return b\*\*n

def d(n):

return n # Hàm d(n) = n

def ktraHamNhan(d, m, n):

return d(m\*n) == d(m) \* d(n)

def T(a, b, n):

if ktraHamNhan(d, a, b):

if a > f(b, n):

n = int(math.log(a, b))

print(f"T(n) = O(n^{n})")

elif a < f(b, n):

n = int(math.log(f(b, n), b))

print(f"T(n) = O(n^{n})")

else:

n = int(math.log(a, b))

print(f"T(n) = O((n^{n})log{b}n)")

else:

print("d(n) không phải là hàm nhân. Không thể giải phương trình.")

a = int(input("Nhâp a = "))

b = int(input("Nhập b = "))

somu\_n = int(input("Nhập số mũ của n = "))

T(a, b, somu\_n)

# **Xây dựng lớp Danhsach – Code ok**

class Danhsach: # Đề kiểm tra Thực hành

def \_\_init\_\_(self, lst=None):

if lst is None:

self.ds = []

else:

self.ds = lst

def nhap(self):

n = int(input("Nhập số lượng phần tử: "))

for i in range(n):

self.ds.append(int(input(f"Nhập phần tử thứ {i+1}: ")))

def xuat(self):

for i in self.ds:

print(i, end=' ')

print()

def tim(self, x):

if x in self.ds:

return self.ds.index(x)

else:

return -1

def them(self, x):

self.ds.append(x)

def xoa(self, x):

if x in self.ds:

self.ds.remove(x)

def sua(self, index, y):

if 0 <= index < len(self.ds):

self.ds[index] = y

else:

print("Index out of range")

def LoaiGiatriTrung(self): # Đề kiểm tra Thực hành

self.ds = list(dict.fromkeys(self.ds))

# Sử dụng lớp DanhSach

ds1 = Danhsach()

ds1.nhap()

ds1.xuat()

x = int(input("Nhập số cần tìm: "))

index = ds1.tim(x)

if index != -1:

print(f"Số {x} tìm thấy ở vị trí {index+1}")

else:

print(f"Số {x} không tìm thấy trong danh sách")

ds1.them(int(input("Nhập số cần thêm: ")))

ds1.xuat()

ds1.xoa(int(input("Nhập số cần xóa: ")))

ds1.xuat()

# **Xây dựng một lớp trong ngôn ngữ lập trình của bạn lựa chọn để quản lý một danh sách liên kết các số nguyên. Trong lớp này, bạn phải xây dựng các phương thức cho việc thêm, tìm kiếm, sửa, xóa và xuất các phần tử.**

class Node:

def \_\_init\_\_(self, data=None):

self.data = data

self.next = None

class DanhSach:

def \_\_init\_\_(self):

self.head = None

# Phương thức thêm phần tử vào cuối danh sách

def them(self, data):

if not self.head:

self.head = Node(data)

else:

cur = self.head

while cur.next:

cur = cur.next

cur.next = Node(data)

# Phương thức tìm kiếm phần tử trong danh sách

def tim(self, data):

cur = self.head

while cur:

if cur.data == data:

return True

cur = cur.next

return False

# Phương thức sửa phần tử trong danh sách

def sua(self, cu, moi):

cur = self.head

while cur:

if cur.data == cu:

cur.data = moi

return True

cur = cur.next

return False

# Phương thức xóa phần tử trong danh sách

def xoa(self, data):

if self.head:

if self.head.data == data:

self.head = self.head.next

return True

else:

cur = self.head

while cur.next:

if cur.next.data == data:

cur.next = cur.next.next

return True

cur = cur.next

return False

# Phương thức xuất toàn bộ danh sách

def xuat(self):

phan\_tu = []

cur = self.head

while cur:

phan\_tu.append(cur.data)

cur = cur.next

return phan\_tu

# Ví dụ về cách sử dụng lớp

ds = DanhSach()

ds.them(1)

ds.them(2)

ds.them(3)

print(ds.xuat()) # Kết quả: [1, 2, 3]

print(ds.tim(2)) # Kết quả: True

ds.sua(2, 5)

print(ds.xuat()) # Kết quả: [1, 5, 3]

ds.xoa(5)

print(ds.xuat()) # Kết quả: [1, 3]

### Merge Sort (Class Sapxep kethua Danhsach) - Ok

class SapXep(Danhsach):

def \_\_init\_\_(self, lst=None):

super().\_\_init\_\_(lst)

@staticmethod

def **merge\_sort**(arr):

n = len(arr)

if n <= 1:

return arr

mid = n // 2

left = arr[:mid]

right = arr[mid:]

return SapXep().merge(SapXep.merge\_sort(left),SapXep.merge\_sort(right))

def merge(self,left,right):

result = []

i = j = 0

while i < len(left) and j < len(right):

if left[i] < right[j]:

result.append(left[i])

i += 1

else:

result.append(right[j])

j += 1

result.extend(left[i:])

result.extend(right[j:])

return result

def SapXepMerge(self):

self.ds = self.merge\_sort(self.ds)

ds\_sapxep = SapXep([5,6,67,1,5,6,4])

print("Danh sách trước khi sắp xếp:")

ds\_sapxep.xuat()

ds\_sapxep.SapXepMerge()

print("Danh sách sau khi sắp xếp:")

ds\_sapxep.xuat()

# **CÁC PHƯƠNG PHÁP SẮP XẾP THUỘC LỚP HÀM N2**

## Phương pháp chọn (Selection Sort) (Class Sapxep\_N^2 kethua Danhsach)

class SapXepNmu2(Danhsach): **# Ok**

def \_\_init\_\_(self, lst=None):

super().\_\_init\_\_(lst)

def **selection\_sort**(self):

n = len(self.ds)

for i in range(n):

min\_temp = i

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

if self.ds[j] < self.ds[min\_temp]:

min\_temp = j

if min\_temp != i:

self.ds[i], self.ds[min\_temp] = self.ds[min\_temp],self.ds[i]

ds\_sapxep = SapXepN2()

#ds\_sapxep.nhap()

print("Danh sách trước khi sắp xếp:")

ds\_sapxep.xuat()

ds\_sapxep.selection\_sort()

print("Danh sách sau khi sắp xếp:")

ds\_sapxep.xuat()

## Phương pháp chèn (Insertion Sort) (Class Sapxep\_N^2 kethua Danhsach)

Có 2 kiểu:

- Chèn bình thường

- Chèn nhị phân

def insertion\_sort(self): **# Ok**

n = len(self.ds)

k = 1

while k < n:

x = self.ds[k]

pos = k - 1

while pos >= 0 and x < self.ds[pos]:

self.ds[pos + 1] = self.ds[pos]

pos -= 1

self.ds[pos + 1] = x

k += 1

ds\_sapxep = SapXepN2()

ds\_sapxep.nhap()

print("Danh sách trước khi sắp xếp:")

ds\_sapxep.xuat()

ds\_sapxep.insertion\_sort()

print("Danh sách sau khi sắp xếp:")

ds\_sapxep.xuat()

**----- # InsertionSort** Tìm kiếm nhị phân # Ok

def binary\_search(self,val,start,end):

if start == end:

return start

mid = (start + end) // 2

if self.ds[mid] < val:

return self.binary\_search(val,mid + 1,end)

else:

return self.binary\_search(val,start,mid)

def InsertionSort2(self):

for i in range(1, len(self.ds)):

val = self.ds[i]

pos = self.binary\_search(val, 0, i)

j = i

while j > pos:

self.ds[j] = self.ds[j - 1]

j -= 1

self.ds[pos] = val

## Phương pháp nổi bọt (Bubble Sort) (Class Sapxep\_N^2 kethua Danhsach)

Có 3 kiểu:

- BubbleSort cơ bản

- BubbleSort tối ưu

- BubbleSort hai chiều

def BubbleSort(self): **# Ok**

n = len(self.ds)

for i in range(n):

for j in range(n - 1, i, -1):

if self.ds[j] < self.ds[j - 1]:

self.ds[j], self.ds[j - 1] = self.ds[j - 1], self.ds[j]

return self.ds

def BubbleSort1(self): **# Ok**

n = len(self.ds)

for i in range(n):

swapped = False

for j in range(n - 1, i, -1):

if self.ds[j] < self.ds[j - 1]:

self.ds[j], self.ds[j - 1] = self.ds[j - 1], self.ds[j]

swapped = True

if not swapped:

break

return self.ds

def BubbleSort2(self): **# Ok**

n = len(self.ds)

swapped = True

start = 0

end = n - 1

while swapped == True:

swapped = False

for i in range(start,end):

if self.ds [i] > self.ds[i+1]:

self.ds[i], self.ds[i+1] = self.ds[i+1], self.ds[i]

swapped = True

if swapped == False:

break

swapped = False

end -= 1

for i in range(end-1,start-1,-1):

if self.ds [i] > self.ds[i+1]:

self.ds[i], self.ds[i+1] = self.ds[i+1], self.ds[i]

swapped = True

start += 1

return self.ds

# **CÁC PHƯƠNG PHÁP SẮP XẾP THUỘC LỚP HÀM NLogN**

## Phương pháp Heap Sort (Class Sapxep\_NLogN kethua Danhsach) - Ok

class **Sort\_NlogN**(DanhSach):

def \_\_init\_\_(self, lst=None):

super().\_\_init\_\_(lst)

def CreateHeap(self, n): # Đề kiểm tra Thực hành

for t in range(n // 2, -1, -1):

i = t

j = 2 \* i + 1

while j < n:

# Giảm dần thì đổi < thành >

if j < n - 1 and self.ds[j] < **self.ds[j + 1]**:

j += 1

# Giảm dần thì đổi < thành >

if self.ds[i] < self.ds[j]:

self.ds[i], self.ds[j] = self.ds[j], self.ds[i]

i = j

j = 2 \* i + 1

else:

break

def HeapSort(self):

n = len(self.ds)

for p in range(n, -1, -1):

self.CreateHeap(p)

if p != 0:

self.ds[0], self.ds[p - 1] = self.ds[p - 1], self.ds[0]

ds\_sapxep = Sort\_NlogN([4,1,2,3,4])

print("Danh sách trước khi sắp xếp:")

ds\_sapxep.xuat()

ds\_sapxep.HeapSort()

print("Danh sách sau khi sắp xếp:")

ds\_sapxep.xuat()

## Phương pháp Quick Sort (Class Sapxep\_NLogN kethua Danhsach) - Ok

def PartitionSort(self, first, last):

if first >= last:

return

x = self.ds[(first + last) // 2]

i = first

j = last

while i <= j:

while self.ds[i] < x:

i += 1

while self.ds[j] > x:

j -= 1

if i <= j:

self.ds[i], self.ds[j] = self.ds[j], self.ds[i]

i += 1

j -= 1

self.PartitionSort(first, j)

self.PartitionSort(i, last)

def QuickSort(self):

self.PartitionSort(0, len(self.ds) - 1)

# **CÁC PHƯƠNG PHÁP SẮP XẾP THUỘC LỚP HÀM N**

## Phương pháp Radix Sort (Class Sapxep\_N kethua Danhsach) - Ok

class Sort\_N(Danhsach):

def \_\_init\_\_(self, lst=None):

super().\_\_init\_\_(lst)

def CountingSort(self, exp):

n = len(self.ds)

output = [0] \* n

count = [0] \* 10

for i in range(n):

index = (self.ds[i] // exp)

count[index % 10] += 1

for i in range(1, 10):

count[i] += count[i - 1]

i = n - 1

while i >= 0:

index = (self.ds[i] // exp)

# Giảm dần: output[**n** - count[(index % 10)**]]** = self.ds[i]

output[count[index % 10] - 1] = self.ds[i]

count[index % 10] -= 1

i -= 1

for i in range(n):

self.ds[i] = output[i]

def RadixSort(self):

exp = 1

max1 = max(self.ds)

while max1 // exp > 0:

self.CountingSort(exp)

exp \*= 10

## Phương pháp Bin Sort (Class Sapxep\_N kethua Danhsach)

### Trường hợp đơn giản

class Sort\_N(Danhsach):

def \_\_init\_\_(self, lst=None):

super().\_\_init\_\_(lst)

def **BinSort**(self):

max\_val = max(self.ds) + 1

bins = [None] \* max\_val

for i in self.ds:

if bins[i] is None:

bins[i] = i

self.ds = [i for i in bins if i is not None]

return self.ds

### Trường hợp tổng quát

import math

class Sort\_N(Danhsach):

def \_\_init\_\_(self, lst=None):

super().\_\_init\_\_(lst)

def **BinSort\_TQ**(self):

min\_val = math.floor(min(self.ds))

max\_val = math.ceil(max(self.ds)) + 1

bins = [[] for \_ in range(min\_val, max\_val)]

for i in self.ds:

bins[math.floor(i) - min\_val].append(i)

sorted\_array = []

for bin in bins:

sorted\_array.extend(bin)

self.ds = sorted\_array

return self.ds

### Trường hợp tập giá trị lớn

def BinSortLarge(self):

min\_val = min(self.ds)

max\_val = max(self.ds) + 1

n = len(self.ds)

bin\_range = (max\_val - min\_val) / n

bins = [[] for \_ in range(n)]

for i in self.ds:

index = int((i - min\_val) // bin\_range)

bins[index].append(i)

sorted\_array = []

for bin in bins:

bin.sort() # use other sorting method for individual bins

sorted\_array.extend(bin)

self.ds = sorted\_array

return self.ds

ds\_sapxep = Sort\_N([100000, 200000, 25000, 75000, 60000, 350000, 120000, 500000])

print("Danh sách trước khi sắp xếp:")

ds\_sapxep.xuat()

ds\_sapxep.BinSortLarge()

print("Danh sách sau khi sắp xếp:")

ds\_sapxep.xuat()

# **CÁC GIẢI THUẬT TÌM KIẾM NỘI (Xây dựng lớp TimKiem kế thừa từ lớp DanhSach)**

## Tìm kiếm tuyến tính (Linear Search)

### Không đệ qui

class TimKiem(Danhsach): **# Ok**

def \_\_init\_\_(self, lst=None):

super().\_\_init\_\_(lst)

def Timkiem\_Tuyentinh(self, x):

for i in range(len(self.ds)):

if self.ds[i] == x:

return i

return -1

ds\_timkiem = TimKiem([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Kết quả sẽ là 4 vì phần tử 5 nằm ở vị trí thứ 4 (đếm từ 0)

print(ds\_timkiem.Timkiem\_Tuyentinh(5))

# Kết quả sẽ là -1 vì không tìm thấy phần tử 11 trong danh sách

print(ds\_timkiem.Timkiem\_Tuyentinh(11))

### Đệ qui - Ok

def Timkiem\_Tuyentinh\_DQ(self, x, index=0): # Đề kiểm tra Thực hành

if index >= len(self.ds): # Có lớp bảo vệ thêm cho điều kiện đúng

return -1

if self.ds[index] == x:

return index

return self.Timkiem\_Tuyentinh\_DQ(x, index+1)

## Tìm kiếm nhị phân (Binary Search)

### Không đệ qui

def Timkiem\_Nhiphan(self, x):

low = 0

high = len(self.ds) - 1

while low <= high:

mid = (high + low) // 2

if self.ds[mid] < x:

low = mid + 1

elif self.ds[mid] > x:

high = mid - 1

else:

return mid

return -1

### Đệ qui

def Timkiem\_Nhiphan\_DQ(self, x, low=0, high=None):

if high is None:

high = len(self.ds) - 1

if high >= low:

mid = (high + low) // 2

if self.ds[mid] == x:

return mid

elif self.ds[mid] > x:

return self.Timkiem\_Nhiphan\_DQ(x, low, mid - 1)

else:

return self.Timkiem\_Nhiphan\_DQ(x, mid + 1, high)

else:

return -1

ds\_timkiem = TimKiem([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

# Kết quả sẽ là 4 vì phần tử 5 nằm ở vị trí thứ 4 (đếm từ 0)

print(ds\_timkiem.Timkiem\_Nhiphan\_DQ(5))

# Kết quả sẽ là -1 vì không tìm thấy phần tử 11 trong danh sách

print(ds\_timkiem.Timkiem\_Nhiphan\_DQ(11))

# **KỸ THUẬT CHIA ĐỂ TRỊ**

## Bài toán nhân các số nguyên lớn

### Thuật toán thông thường

def multiply\_bigints(a, b):

# Chuyển đổi số nguyên thành chuỗi

a = str(a)

b = str(b)

# Khởi tạo kết quả là 0

result = 0

# Duyệt qua mỗi chữ số của số nguyên thứ hai

for i in range(len(b)):

# Nhân số nguyên thứ nhất với chữ số hiện tại của số nguyên thứ hai

# sau đó dịch chuyển kết quả sang trái i chữ số

temp = int(a) \* int(b[len(b)-1-i]) \* (10\*\*i)

# Cộng kết quả vào tổng

result += temp

return result

# print(multiply\_bigints(12345678901234567890, 98765432109876543210))

print(multiply\_bigints(123456, 654321))

### Thuật toán áp dụng kỹ thuật chia để trị

def Big\_int\_mult(X, Y):

# Nếu một trong hai số là 0, kết quả là 0

if X == 0 or Y == 0:

return 0

# Xử lý dấu của số

s = 1 if X \* Y >= 0 else -1

X, Y = abs(X), abs(Y)

if X < 10 or Y < 10:

return s \* X \* Y

else:

n = max(len(str(X)), len(str(Y)))

m = n // 2

a = X // 10\*\*(m)

b = X % 10\*\*(m)

c = Y // 10\*\*(m)

d = Y % 10\*\*(m)

ac = Big\_int\_mult(a, c)

bd = Big\_int\_mult(b, d)

ad\_plus\_bc = Big\_int\_mult(a+b, c+d) - ac - bd

return s \* (ac \* 10\*\*(2\*m) + (ad\_plus\_bc \* 10\*\*m) + bd)

X = 123456789012345678901234567890

Y = 1234567890

result = Big\_int\_mult(X,Y)

print("Kết quả là: ",result)

print(X\*Y)

## Nhân hai ma trận:

### Viết một chương trình cho phép nhân hai ma trận vuông cấp N lớn sử dụng thuật toán chia để trị.

def split\_matrix(A):

size = len(A)

mid = size // 2

return (

[row[:mid] for row in A[:mid]],

[row[mid:] for row in A[:mid]],

[row[:mid] for row in A[mid:]],

[row[mid:] for row in A[mid:]]

)

def add\_matrix(A, B):

n = len(A)

C = [[0 for j in range(n)] for i in range(n)]

for i in range(n):

for j in range(n):

C[i][j] = A[i][j] + B[i][j]

return C

def subtract\_matrix(A, B):

n = len(A)

C = [[0 for j in range(n)] for i in range(n)]

for i in range(n):

for j in range(n):

C[i][j] = A[i][j] - B[i][j]

return C

def join\_matrix(C11, C12, C21, C22):

n = len(C11)

C = [[0 for \_ in range(2\*n)] for \_ in range(2\*n)]

for i in range(n):

for j in range(n):

C[i][j] = C11[i][j]

C[i][j+n] = C12[i][j]

C[i+n][j] = C21[i][j]

C[i+n][j+n] = C22[i][j]

return C

def strassen(A, B, size):

# Trường hợp cơ sở: size=1

if size == 1:

return [[A[0][0] \* B[0][0]]]

else:

# Chia ma trận A và B thành 4 phần

new\_size = size // 2

A11, A12, A21, A22 = split\_matrix(A)

B11, B12, B21, B22 = split\_matrix(B)

# Tính toán P1 đến P7

P1 = strassen(add\_matrix(A11, A22), add\_matrix(B11, B22), new\_size)

P2 = strassen(add\_matrix(A21, A22), B11, new\_size)

P3 = strassen(A11, subtract\_matrix(B12, B22), new\_size)

P4 = strassen(A22, subtract\_matrix(B21, B11), new\_size)

P5 = strassen(add\_matrix(A11, A12), B22, new\_size)

P6 = strassen(subtract\_matrix(A21, A11), add\_matrix(B11, B12), new\_size)

P7 = strassen(subtract\_matrix(A12, A22), add\_matrix(B21, B22), new\_size)

# Tính toán C11, C12, C21, C22

C11 = add\_matrix(subtract\_matrix(add\_matrix(P1, P4), P5), P7)

C12 = add\_matrix(P3, P5)

C21 = add\_matrix(P2, P4)

C22 = add\_matrix(subtract\_matrix(add\_matrix(P1, P3), P2), P6)

# Gộp các ma trận C thành kết quả cuối cùng

return join\_matrix(C11, C12, C21, C22)

A = [[1, 3], [7, 5]]

B = [[6, 8], [4, 2]]

n = len(A)

print(strassen(A, B, n)) # Kết quả: [[18, 14], [62, 66]]

### Nhân hai ma trận nhiều giá trị:

def split\_matrix(A):

size = len(A)

mid = size // 2

return (

[row[:mid] for row in A[:mid]],

[row[mid:] for row in A[:mid]],

[row[:mid] for row in A[mid:]],

[row[mid:] for row in A[mid:]]

)

def join\_matrix(C, new\_size, x, y):

size = len(C)

for i in range(size):

for j in range(size):

new\_size[i+x][j+y] = C[i][j]

def add\_matrix(A, B):

n = len(A)

C = [[0 for j in range(n)] for i in range(n)]

for i in range(n):

for j in range(n):

C[i][j] = A[i][j] + B[i][j]

return C

def subtract\_matrix(A, B):

n = len(A)

C = [[0 for j in range(n)] for i in range(n)]

for i in range(n):

for j in range(n):

C[i][j] = A[i][j] - B[i][j]

return C

def strassen(A, B, size):

if size == 1:

return [[A[0][0] \* B[0][0]]]

new\_size = size // 2

A11, A12, A21, A22 = split\_matrix(A)

B11, B12, B21, B22 = split\_matrix(B)

P1 = strassen(add\_matrix(A11, A22), add\_matrix(B11, B22), new\_size)

P2 = strassen(add\_matrix(A21, A22), B11, new\_size)

P3 = strassen(A11, subtract\_matrix(B12, B22), new\_size)

P4 = strassen(A22, subtract\_matrix(B21, B11), new\_size)

P5 = strassen(add\_matrix(A11, A12), B22, new\_size)

P6 = strassen(subtract\_matrix(A21, A11), add\_matrix(B11, B12), new\_size)

P7 = strassen(subtract\_matrix(A12, A22), add\_matrix(B21, B22), new\_size)

C11 = add\_matrix(subtract\_matrix(add\_matrix(P1, P4), P5), P7)

C12 = add\_matrix(P3, P5)

C21 = add\_matrix(P2, P4)

C22 = add\_matrix(subtract\_matrix(add\_matrix(P1, P3), P2), P6)

C = [[0 for \_ in range(size)] for \_ in range(size)]

join\_matrix(C11, C, 0, 0)

join\_matrix(C12, C, 0, new\_size)

join\_matrix(C21, C, new\_size, 0)

join\_matrix(C22, C, new\_size, new\_size)

return C

def get\_matrix\_input(size):

return [list(map(int, input().split())) for \_ in range(size)]

def pad\_matrix(A, size):

while len(A) < size:

A.append([0]\*len(A[0]))

for row in A:

while len(row) < size:

row.append(0)

return A

def main():

n = int(input("Nhập kích thước ma trận vuông (n x n): "))

size = 1

while size < n: # Tìm kích thước lũy thừa của 2 gần nhất

size \*= 2

print("Nhập ma trận A")

A = pad\_matrix(get\_matrix\_input(n), size)

print("Nhập ma trận B")

B = pad\_matrix(get\_matrix\_input(n), size)

C = strassen(A, B, size)

print("Kết quả nhân hai ma trận A và B là:")

for row in C[:n]:

print(" ".join(str(x) for x in row[:n]))

if \_\_name\_\_ == "\_\_main\_\_":

main()

## Tìm UCLN và BCNN sử dụng kỹ thuật chia để trị:

Viết một chương trình Python để tìm ước chung lớn nhất (UCLN) và bội chung nhỏ nhất (BCNN) của hai số nguyên dương sử dụng kỹ thuật chia để trị.

*Để tìm UCLN của hai số, ta có thể sử dụng thuật toán Euclid, mà có thể được cài đặt dưới dạng đệ quy, và đây cũng là một dạng của kỹ thuật chia để trị. Bội chung nhỏ nhất của hai số `a` và `b` có thể được tính thông qua công thức: `BCNN(a, b) = |a \* b| / UCLN(a, b)`.*

def ucln(a, b):

if b == 0:

return a

return ucln(b, a % b)

def bcnn(a, b):

return abs(a \* b) // ucln(a, b)

# Nhập hai số nguyên dương

a = int(input("Nhập số nguyên dương thứ nhất: "))

b = int(input("Nhập số nguyên dương thứ hai: "))

# Tìm UCLN và BCNN

ucln\_result = ucln(a, b)

bcnn\_result = bcnn(a, b)

# In kết quả

print(f"Ước chung lớn nhất của {a} và {b} là: {ucln\_result}")

print(f"Bội chung nhỏ nhất của {a} và {b} là: {bcnn\_result}")

*Trong chương trình này:*

*- `ucln` là hàm đệ quy để tìm ước chung lớn nhất của hai số `a` và `b` sử dụng thuật toán Euclid.*

*- `bcnn` là hàm dùng để tìm bội chung nhỏ nhất của hai số `a` và `b` thông qua công thức đã đề cập.*

## Tìm giá trị nhỏ nhất trong ma trận vuông cấp N bằng kỹ thuật chia để trị

### TH1: Đệ qui, chia hàng

### Trường hợp nhận số chẵn

def minMatrix(matrix, row\_start, row\_end):

# Kiểm tra ma trận rỗng

if not matrix or not matrix[0]:

return None

if row\_start == row\_end: # chỉ có một hàng

return min(matrix[row\_start])

# chia ma trận thành hai nửa

mid = row\_start + (row\_end - row\_start) // 2

# tìm giá trị nhỏ nhất trong mỗi nửa bằng đệ qui

min1 = minMatrix(matrix, row\_start, mid)

min2 = minMatrix(matrix, mid+1, row\_end)

# giá trị nhỏ nhất của toàn bộ ma trận là giá trị nhỏ nhất giữa min1 và min2

return min(min1, min2)

N = 4

matrix = [[5,2,7,4],

[3,2,8,8],

[9,7,3,5],

[6,4,9,7]]

min\_value = minMatrix(matrix, 0, N-1)

print("Giá trị nhỏ nhất trong ma trận là:", min\_value)

### Có trường hợp nếu nhập số lẻ, chẵn:

def find\_min(matrix, start\_row, end\_row, start\_col, end\_col):

# Trường hợp cơ sở: nếu chỉ còn một phần tử trong ma trận

if start\_row > end\_row or start\_col > end\_col:

return float("inf")

if start\_row == end\_row and start\_col == end\_col:

return matrix[start\_row][start\_col]

# Chia ma trận thành 4 phần nhỏ hơn

mid\_row = (start\_row + end\_row) // 2

mid\_col = (start\_col + end\_col) // 2

# Tìm giá trị nhỏ nhất trong mỗi phần nhỏ hơn

min1 = find\_min(matrix, start\_row, mid\_row, start\_col, mid\_col)

min2 = find\_min(matrix, start\_row, mid\_row, mid\_col + 1, end\_col)

min3 = find\_min(matrix, mid\_row + 1, end\_row, start\_col, mid\_col)

min4 = find\_min(matrix, mid\_row + 1, end\_row, mid\_col + 1, end\_col)

# Trả về giá trị nhỏ nhất trong cả bốn phần

return min(min1, min2, min3, min4)

# Ví dụ sử dụng hàm:

# N = 4

# matrix = [

# [7, 2, 8],

# [6, 4, 3],

# [5, 9, 1]

# ]

# Nhập kích thước của ma trận từ người dùng

N = int(input("Nhập kích thước của ma trận (N): "))

# Tạo ma trận

matrix = []

# Nhập giá trị cho từng phần tử trong ma trận

print("Nhập giá trị cho ma trận:")

for i in range(N):

row = []

for j in range(N):

value = int(input(f"Nhập giá trị cho phần tử ở hàng {i + 1}, cột {j + 1}: "))

row.append(value)

matrix.append(row)

# Tìm và in ra giá trị nhỏ nhất trong ma trận

print(f"Giá trị nhỏ nhất trong ma trận là: {find\_min(matrix, 0, len(matrix) - 1, 0, len(matrix[0]) - 1)}")

### TH2 (Khó): Đệ qui, Chia thành 4 ô vuông

### Trường hợp số chẵn

def find\_min(matrix, start\_row, end\_row, start\_col, end\_col):

# Trường hợp cơ sở: nếu chỉ còn một phần tử trong ma trận

if start\_row == end\_row and start\_col == end\_col:

return matrix[start\_row][start\_col]

# Chia ma trận thành 4 phần nhỏ hơn

mid\_row = (start\_row + end\_row) // 2

mid\_col = (start\_col + end\_col) // 2

# Tìm giá trị nhỏ nhất trong mỗi phần nhỏ hơn

min1 = find\_min(matrix, start\_row, mid\_row, start\_col, mid\_col)

min2 = find\_min(matrix, start\_row, mid\_row, mid\_col + 1, end\_col)

min3 = find\_min(matrix, mid\_row + 1, end\_row, start\_col, mid\_col)

min4 = find\_min(matrix, mid\_row + 1, end\_row, mid\_col + 1, end\_col)

# Trả về giá trị nhỏ nhất trong cả bốn phần

return min(min1, min2, min3, min4)

# Ví dụ sử dụng

matrix = [[4, 2, 9, 1],[8, 3, 6, 2],[5, 7, 4, 3],[1, 6, 8, 4]]

# Gọi hàm find\_min với chỉ số ban đầu của ma trận

min\_value = find\_min(matrix, 0, len(matrix) - 1, 0, len(matrix[0]) - 1)

print("Giá trị nhỏ nhất trong ma trận:", min\_value)

### Trường hợp số lẽ, chẵn:

def find\_min(matrix, start\_row, end\_row, start\_col, end\_col):

# Trường hợp cơ sở: nếu chỉ còn một phần tử trong ma trận

if start\_row > end\_row or start\_col > end\_col:

return float("inf")

if start\_row == end\_row and start\_col == end\_col:

return matrix[start\_row][start\_col]

# Chia ma trận thành 4 phần nhỏ hơn

mid\_row = (start\_row + end\_row) // 2

mid\_col = (start\_col + end\_col) // 2

# Tìm giá trị nhỏ nhất trong mỗi phần nhỏ hơn

min1 = find\_min(matrix, start\_row, mid\_row, start\_col, mid\_col)

min2 = find\_min(matrix, start\_row, mid\_row, mid\_col + 1, end\_col)

min3 = find\_min(matrix, mid\_row + 1, end\_row, start\_col, mid\_col)

min4 = find\_min(matrix, mid\_row + 1, end\_row, mid\_col + 1, end\_col)

# Trả về giá trị nhỏ nhất trong cả bốn phần

return min(min1, min2, min3, min4)

# Ví dụ sử dụng

matrix = [[4, 2, 9],[8, 3, 6],[5, 7, 4]] # Ma trận kích thước lẻ 3x3

# Gọi hàm find\_min với chỉ số ban đầu của ma trận

min\_value = find\_min(matrix, 0, len(matrix) - 1, 0, len(matrix[0]) - 1)

print("Giá trị nhỏ nhất trong ma trận:", min\_value)

# **KỸ THUẬT THAM ĂN**

**Đổi tiền từ máy ATM bằng**

### Thuật toán thông thường

### Thuật toán áp dụng kỹ thuật tham ăn

**Week6\_Baitap2\_SotoGiohanLaS:**

def atm\_withdrawal(amount, denominations, S):

denominations.sort(reverse=True) # Sắp xếp các mệnh giá tiền từ lớn đến nhỏ

result = []

total\_count = 0

for denom in denominations:

count = 0

while amount >= denom and count < S:

amount -= denom

count += 1

total\_count += 1

result.append((denom, count))

if total\_count == S:

break

if amount != 0:

return "Cannot withdraw the desired amount with the given limit S", result

else:

return result

#denominations = [500000, 200000, 100000, 50000]

#amount = 3450000

#S = 5

N=[]

X=int(input('Nhập mệnh giá tờ tiền:'))

while X!=0:

N.append(X)

X = int(input('Nhập mệnh giá tờ tiền:'))

Money=int(input('Nhập số tiền:'))

print(atm\_withdrawal(X, N, S))

S = int(input('Nhập giới hạn tờ mệnh giá:'))

**Week6\_Baitap2\_KhongGioihanSoTo:**

# Câu 2: VCT đổi tiền từ máy ATM với kỹ thuật tham ăn

# - Số tờ không hạn chế

# - Số tờ giới hạn là S

# - Số tờ 1

def Chon(so\_tien, menh\_gia):

# Sắp xếp các mệnh giá tiền từ lớn đến nhỏ

menh\_gia.sort(reverse=True)

# Khởi tạo kết quả

kq = []

for menh\_gia\_tien in menh\_gia:

count = 0

while so\_tien >= menh\_gia\_tien:

so\_tien -= menh\_gia\_tien

count += 1

kq.append((menh\_gia\_tien, count))

# Nếu không thể rút hết số tiền yêu cầu, trả về lỗi

if so\_tien != 0:

return "Lỗi: không thể rút số tiền chính xác", kq

return kq

N = 3450000

X = [500000, 200000, 100000, 50000]

for menh\_gia, so\_to in Chon(N, X):

print(f"Mệnh giá {menh\_gia}: {so\_to} tờ")

**Week 6\_Baitap2\_GioihanSotoLa1:**

def Chon\_GioihanSotoLa1(so\_tien, menh\_gia):

menh\_gia.sort(reverse=True)

kq = []

for menh\_gia\_tien in menh\_gia:

if menh\_gia\_tien <= so\_tien:

kq.append((menh\_gia\_tien, 1))

so\_tien -= menh\_gia\_tien

if so\_tien == 0:

break

if so\_tien != 0:

print("Không thể rút số tiền này với các mệnh giá hiện có.")

else:

print("Các mệnh giá tiền để rút: ", kq)

return kq

N = 3450000

# N = 100000

X = [500000,200000,100000,50000]

for menh\_gia, so\_to in Chon\_GioihanSotoLa1(N, X):

print(f"Mệnh giá {menh\_gia}: {so\_to} tờ")

## ATM:

### -----Không giới hạn số tờ-----

def Chon(X,N):

X = sorted(X, reverse=True)

count = 0

total = 0

phuongan =[]

for i in X:

while total <= N:

count +=1

total += i

if total > N:

total -= i

count -= 1

break

phuongan.append((i,count))

count = 0

return phuongan

N = 3450000

X = [500000,200000,100000,50000]

results=Chon(X,N)

for menh\_gia, so\_to in results:

print(f"Mệnh giá {menh\_gia}: {so\_to} tờ")

### ----Giới hạn số tờ là 1-----

def Chon(X,N):

X = sorted(X, reverse=True)

count = 0

total = 0

so\_to = 1

phuongan = []

for i in X:

while so\_to > 0:

so\_to -= 1

count += 1

total += i

if total > N:

total -= i

count -= 1

phuongan.append((i,count))

count = 0

so\_to = 1

return phuongan

X = [500000,200000,100000,50000]

N = 3450000

results=Chon(X,N)

for menh\_gia, so\_to in results:

print(f"Mệnh giá {menh\_gia}: {so\_to} tờ")

### -----Số tờ tùy chọn -----

S = 5

def Chon(X,N):

X = sorted(X, reverse=True)

count = 0

total = 0

so\_to = S

phuongan = []

for i in X:

while so\_to > 0:

so\_to -= 1

count += 1

total += i

if total > N:

total -= i

count -= 1

break

phuongan.append((i,count))

count = 0

so\_to = S

return phuongan

X = [500000,200000,100000,50000]

N = 3450000

results=Chon(X,N)

for menh\_gia, so\_to in results:

## Bài toán tìm đường đi ngắn nhất (TSP) của người giao hàng

class NguoiBanHang:

def \_\_init\_\_(self, ma\_tran):

self.ma\_tran = ma\_tran

self.so\_thanh\_pho = len(ma\_tran)

def duong\_di\_tham\_lam(self):

da\_tham = [0] # điểm bắt đầu

tong\_khoang\_cach = 0

while len(da\_tham) != self.so\_thanh\_pho:

di\_tu = da\_tham[-1]

khoang\_cach\_nho\_nhat = float('inf')

for thanh\_pho in range(self.so\_thanh\_pho):

if thanh\_pho not in da\_tham and self.ma\_tran[di\_tu][thanh\_pho] < khoang\_cach\_nho\_nhat:

khoang\_cach\_nho\_nhat = self.ma\_tran[di\_tu][thanh\_pho]

thanh\_pho\_tiep\_theo = thanh\_pho

da\_tham.append(thanh\_pho\_tiep\_theo)

tong\_khoang\_cach += khoang\_cach\_nho\_nhat

# Trở về thành phố bắt đầu

tong\_khoang\_cach += self.ma\_tran[da\_tham[-1]][0]

return da\_tham, tong\_khoang\_cach # Trả về danh sách các thành phố đã thăm và tổng khoảng cách

# Định nghĩa một ma trận biểu diễn khoảng cách giữa các thành phố

ma\_tran = [

[0, 10, 15, 20],

[10, 0, 35, 25],

[15, 35, 0, 30],

[20, 25, 30, 0]

]

# Tạo một thể hiện của lớp NguoiBanHang

nguoi\_ban\_hang = NguoiBanHang(ma\_tran)

# Lấy kết quả

da\_tham, tong\_khoang\_cach = nguoi\_ban\_hang.duong\_di\_tham\_lam()

# In kết quả

print("Các thành phố đã thăm theo thứ tự: ", da\_tham)

print("Tổng khoảng cách: ", tong\_khoang\_cach)

## Bài toán cá ba lô (knapsack) - Kỹ thuật tham ăn

**Week7\_Baitap1:**

class Do\_vat:

def \_\_init\_\_(self, Ten, Trong\_luong, Gia\_tri, Don\_gia, Phuong\_an=0):

self.Ten = Ten

self.Trong\_luong = Trong\_luong

self.Gia\_tri = Gia\_tri

self.Don\_gia = Don\_gia

self.Phuong\_an = Phuong\_an

def Chon(Trong\_luong, W): # Trọng lượng của 1 vật, W: Trong luong con lai W cuar ba lo

return 1 if Trong\_luong <= W else 0

def Greedy(dsdv, W):

# Sắp xếp danh sách theo đơn giá giảm dần

dsdv.sort(key=lambda x: x.Don\_gia, reverse=True)

for do\_vat in dsdv:

do\_vat.Phuong\_an = Chon(do\_vat.Trong\_luong, W)

W = W - do\_vat.Phuong\_an \* do\_vat.Trong\_luong

if W == 0:

break

return dsdv # Trả về danh sách đồ vật sau khi đã chọn

# Tạo danh sách các đồ vật

dsdv = [Do\_vat("vật 1", 10, 60, 6),

Do\_vat("vật 2", 20, 100, 5),

Do\_vat("vật 3", 30, 120, 4)]

# Trọng lượng tối đa của ba lô

W = 50

# Chạy thuật toán tham lam

dsdv = Greedy(dsdv, W)

# In ra kết quả

for do\_vat in dsdv:

print(f"Đồ vật {do\_vat.Ten}: Số lần chọn = {do\_vat.Phuong\_an}")

## Cái ba lô:

### ----Không giới hạn-----

class Do\_vat:

def \_\_init\_\_(self,ten,w,gt):

self.ten = ten

self.trong\_luong = w

self.gia\_tri = gt

self.don\_gia = gt / w

self.phuong\_an = 0

def greedy(dsdv, w):

dsdv = list(dsdv)

dsdv.sort(key = lambda x: x.don\_gia, reverse=True)

for i in range(len(dsdv)):

while w >= dsdv[i].trong\_luong:

dsdv[i].phuong\_an = w // dsdv[i].trong\_luong

w -= dsdv[i].phuong\_an \* dsdv[i].trong\_luong

# Tính tổng trọng lượng và tổng giá trị

tong\_trong\_luong = sum([dv.trong\_luong \* dv.phuong\_an for dv in dsdv])

tong\_gia\_tri = sum([dv.gia\_tri \* dv.phuong\_an for dv in dsdv])

return dsdv, tong\_trong\_luong, tong\_gia\_tri

def in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri):

print("Danh sách đồ vật được chọn:")

for dv in dsdv:

if dv.phuong\_an > 0:

print("Tên: ", dv.ten, " - Trọng lượng: ", dv.trong\_luong, " - Giá trị: ", dv.gia\_tri," - Đơn giá:",dv.don\_gia,"- Phương án:",dv.phuong\_an)

print("\nTổng trọng lượng: ", tong\_trong\_luong)

print("Tổng giá trị: ", tong\_gia\_tri)

dsdv = [Do\_vat("A",15,30),

Do\_vat("B",10,25),

Do\_vat("C",2,2),

Do\_vat("D",4,6)]

w = 37

dsdv, tong\_trong\_luong, tong\_gia\_tri = greedy(dsdv, w)

in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri)

### -------Tùy chọn theo số lượng-----

class Do\_vat:

def \_\_init\_\_(self, ten, w, gt, S):

self.ten = ten

self.trong\_luong = w

self.gia\_tri = gt

self.don\_gia = gt / w

self.phuong\_an = 0

self.so\_luong = S

def greedy(dsdv, w):

dsdv = sorted(dsdv, key = lambda x: x.don\_gia, reverse=True)

for dv in dsdv:

while w >= dv.trong\_luong and dv.so\_luong > 0:

dv.phuong\_an += 1

dv.so\_luong -= 1

w -= dv.trong\_luong

# Tính tổng trọng lượng và tổng giá trị

tong\_trong\_luong = sum([dv.trong\_luong \* dv.phuong\_an for dv in dsdv])

tong\_gia\_tri = sum([dv.gia\_tri \* dv.phuong\_an for dv in dsdv])

return dsdv, tong\_trong\_luong, tong\_gia\_tri

dsdv =[Do\_vat("A",15,30,2),

Do\_vat("B",10,25,3),

Do\_vat("C",2,2,4),

Do\_vat("D",4,6,5)]

def in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri):

print("Danh sách đồ vật được chọn:")

for dv in dsdv:

if dv.phuong\_an > 0:

print("Tên: ", dv.ten, " - Trọng lượng: ", dv.trong\_luong, " - Giá trị: ", dv.gia\_tri,"- Số lượng còn lại:",dv.so\_luong," - Đơn giá:",dv.don\_gia,"-Phương án:",dv.phuong\_an)

print("\nTổng trọng lượng: ", tong\_trong\_luong)

print("Tổng giá trị: ", tong\_gia\_tri)

w = 37

dsdv, tong\_trong\_luong, tong\_gia\_tri = greedy(dsdv, w)

in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri)

### -----Mỗi đồ vật chỉ chọn 1---

class Do\_vat:

def \_\_init\_\_(self,ten,w,gt):

self.ten = ten

self.trong\_luong = w

self.gia\_tri = gt

self.don\_gia = gt / w

self.phuong\_an = 0

def greedy(dsdv,w):

dsdv = list(dsdv)

dsdv.sort(key = lambda x:x.don\_gia, reverse=True)

for i in range(len(dsdv)):

dsdv[i].phuong\_an = min((w//dsdv[i].trong\_luong),1)

w -= dsdv[i].phuong\_an \* dsdv[i].trong\_luong

# Tính tổng trọng lượng và tổng giá trị

tong\_trong\_luong = sum([dv.trong\_luong \* dv.phuong\_an for dv in dsdv])

tong\_gia\_tri = sum([dv.gia\_tri \* dv.phuong\_an for dv in dsdv])

return dsdv, tong\_trong\_luong, tong\_gia\_tri

dsdv = [Do\_vat("A",15,30),

Do\_vat("B",10,25),

Do\_vat("C",2,2),

Do\_vat("D",4,6)]

def in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri):

print("Danh sách đồ vật được chọn:")

for dv in dsdv:

if dv.phuong\_an > 0:

print("Tên: ", dv.ten, " - Trọng lượng: ", dv.trong\_luong, " - Giá trị: ", dv.gia\_tri," - Đơn giá:",dv.don\_gia,"-Phương án:",dv.phuong\_an)

print("\nTổng trọng lượng: ", tong\_trong\_luong)

print("Tổng giá trị: ", tong\_gia\_tri)

w = 37

dsdv, tong\_trong\_luong, tong\_gia\_tri = greedy(dsdv, w)

in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri)

# **KỸ THUẬT QUI HOẠCH ĐỘNG (Dynamic programming)**

## Tính số tổ hợp

### Bằng thuật toán đệ qui, áp dụng kỹ thuật qui hoạch động với CTDL là bảng

- Tốt hơn về mặt không gian.

### Week7\_Baitap2\_Mang2chieu:

def Comb(n, k):

# Tạo bảng C với tất cả giá trị ban đầu là 0

C = [[0 for \_ in range(k + 1)] for \_ in range(n + 1)]

# Tính giá trị của bảng C theo công thức của tổ hợp

for i in range(n + 1):

for j in range(min(i, k) + 1):

# Các trường hợp cơ bản

if j == 0 or j == i:

C[i][j] = 1

# Sử dụng công thức tái sử dụng kết quả trước đó

else:

C[i][j] = C[i - 1][j - 1] + C[i - 1][j]

return C[n][k]

n = 10

k = 3

print(Comb(n, k)) # Output: 120

### Bằng thuật toán đệ qui, áp dụng kỹ thuật qui hoạch động với CTDL là vector

- 2 đoạn mã đều không đúng

### Week7\_Baitap2\_Mang1Chieu:

def Comb(n, k): # Đề kiểm tra Thực hành

C = [0 for \_ in range(k+1)]

C[0] = 1

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

j = min(i, k)

while j > 0:

C[j] = C[j] + C[j-1]

j -= 1

return C[k]

print(Comb(5, 2)) # Output: 10

## Bài toán cái ba lô - Week7\_Baitap3:

class DoVat:

def \_\_init\_\_(self, ten, tl, gt, so\_dv\_duoc\_chon=0):

self.ten = ten

self.gt = gt

self.tl = tl

self.so\_dv\_duoc\_chon = so\_dv\_duoc\_chon

# Hàm knapsack sử dụng kỹ thuật qui hoạch động để giải quyết bài toán cái ba lô

def knapsack(dsdv, W, n):

# Khởi tạo bảng K với kích thước (n+1) x (W+1)

K = [[0 for w in range(W + 1)]

for i in range(n + 1)]

# Xây dựng bảng K theo cách từ dưới lên

for i in range(n + 1):

for w in range(W + 1):

# Nếu không có đồ vật nào hoặc khối lượng ba lô bằng 0 thì giá trị lợi ích là 0

if i == 0 or w == 0:

K[i][w] = 0

# Nếu đồ vật thứ i có thể được thêm vào ba lô

elif dsdv[i - 1].tl <= w:

# Chọn giá trị lớn hơn giữa việc không lấy đồ vật thứ i và lấy đồ vật thứ i

K[i][w] = max(dsdv[i - 1].gt

+ K[i - 1][w - dsdv[i - 1].tl], K[i - 1][w])

else:

# Nếu không thể thêm đồ vật thứ i vào ba lô thì giữ nguyên giá trị lợi ích

K[i][w] = K[i - 1][w]

# Trả về giá trị lợi ích tối đa

return K[n][W]

# Khởi tạo danh sách đồ vật

dsdv = [

DoVat("1", 3, 4),

DoVat("2", 5, 5),

DoVat("3", 5, 6),

DoVat("4", 2, 3),

DoVat("5", 1, 1)

]

W = 9 # Khối lượng tối đa của ba lô

n = len(dsdv) # Số lượng đồ vật

print(knapsack(dsdv, W, n)) # In ra giá trị lợi ích tối đa

### TH1: Chọn đồ vắn có đơn giá cao nhất cho đến khi không thể chọn thêm đồ vật nào class Do\_vat:

def \_\_init\_\_(self,ten,w,gt):

self.ten = ten

self.trong\_luong = w

self.gia\_tri = gt

self.don\_gia = gt / w

self.phuong\_an = 0

def greedy(dsdv, w):

dsdv = list(dsdv)

dsdv.sort(key = lambda x: x.don\_gia, reverse=True)

for i in range(len(dsdv)):

while w >= dsdv[i].trong\_luong:

dsdv[i].phuong\_an = w // dsdv[i].trong\_luong

w -= dsdv[i].phuong\_an \* dsdv[i].trong\_luong

# Tính tổng trọng lượng và tổng giá trị

tong\_trong\_luong = sum([dv.trong\_luong \* dv.phuong\_an for dv in dsdv])

tong\_gia\_tri = sum([dv.gia\_tri \* dv.phuong\_an for dv in dsdv])

return dsdv, tong\_trong\_luong, tong\_gia\_tri

def in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri):

print("Danh sách đồ vật được chọn:")

for dv in dsdv:

if dv.phuong\_an > 0:

print("Tên: ", dv.ten, " - Trọng lượng: ", dv.trong\_luong, " - Giá trị: ", dv.gia\_tri," - Đơn giá:",dv.don\_gia,"- Phương án:",dv.phuong\_an)

print("\nTổng trọng lượng: ", tong\_trong\_luong)

print("Tổng giá trị: ", tong\_gia\_tri)

dsdv = [Do\_vat("A",15,30),Do\_vat("B",10,25),Do\_vat("C",2,2),Do\_vat("D",4,6)]

w = 37

dsdv, tong\_trong\_luong, tong\_gia\_tri = greedy(dsdv, w)

in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri)

### TH2: Có số lượng

class Do\_vat:

def \_\_init\_\_(self, ten, w, gt, S):

self.ten = ten

self.trong\_luong = w

self.gia\_tri = gt

self.don\_gia = gt / w

self.phuong\_an = 0

**self.so\_luong = S**

def greedy(dsdv, w):

dsdv = sorted(dsdv, key = lambda x: x.don\_gia, reverse=True)

for dv in dsdv:

while w >= dv.trong\_luong and dv.so\_luong > 0:

dv.phuong\_an += 1

dv.so\_luong -= 1

w -= dv.trong\_luong

# Tính tổng trọng lượng và tổng giá trị

tong\_trong\_luong = sum([dv.trong\_luong \* dv.phuong\_an for dv in dsdv])

tong\_gia\_tri = sum([dv.gia\_tri \* dv.phuong\_an for dv in dsdv])

return dsdv, tong\_trong\_luong, tong\_gia\_tri

dsdv = [Do\_vat("A",15,30,2),

Do\_vat("B",10,25,3),

Do\_vat("C",2,2,4),

Do\_vat("D",4,6,5)]

def in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri):

print("Danh sách đồ vật được chọn:")

for dv in dsdv:

if dv.phuong\_an > 0:

print("Tên: ", dv.ten, " - Trọng lượng: ", dv.trong\_luong, " - Giá trị: ", dv.gia\_tri,"- Số lượng còn lại:",dv.so\_luong," - Đơn giá:",dv.don\_gia,"-Phương án:",dv.phuong\_an)

print("\nTổng trọng lượng: ", tong\_trong\_luong)

print("Tổng giá trị: ", tong\_gia\_tri)

w = 37

dsdv, tong\_trong\_luong, tong\_gia\_tri = greedy(dsdv, w)

in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri)

### TH3: Chọn tối đa một đồ vật của từng loại

class Do\_vat:

def \_\_init\_\_(self,ten,w,gt):

self.ten = ten

self.trong\_luong = w

self.gia\_tri = gt

self.don\_gia = gt / w

self.phuong\_an = 0

def greedy(dsdv,w):

dsdv = list(dsdv)

dsdv.sort(key = lambda x:x.don\_gia, reverse=True)

for i in range(len(dsdv)):

dsdv[i].phuong\_an = min((w//dsdv[i].trong\_luong),1)

w -= dsdv[i].phuong\_an \* dsdv[i].trong\_luong

# Tính tổng trọng lượng và tổng giá trị

tong\_trong\_luong = sum([dv.trong\_luong \* dv.phuong\_an for dv in dsdv])

tong\_gia\_tri = sum([dv.gia\_tri \* dv.phuong\_an for dv in dsdv])

return dsdv, tong\_trong\_luong, tong\_gia\_tri

dsdv = [Do\_vat("A",15,30),Do\_vat("B",10,25),Do\_vat("C",2,2),Do\_vat("D",4,6)]

def in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri):

print("Danh sách đồ vật được chọn:")

for dv in dsdv:

if dv.phuong\_an > 0:

print("Tên: ", dv.ten, " - Trọng lượng: ", dv.trong\_luong, " - Giá trị: ", dv.gia\_tri," - Đơn giá:",dv.don\_gia,"-Phương án:",dv.phuong\_an)

print("\nTổng trọng lượng: ", tong\_trong\_luong)

print("Tổng giá trị: ", tong\_gia\_tri)

w = 37

dsdv, tong\_trong\_luong, tong\_gia\_tri = greedy(dsdv, w)

in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri)

# **KỸ THUẬT QUAY LUI**

## Định trị cây biểu thức số học

- Bao gồm xử lý cho trường hợp chia cho 0

# 1. VCT định trị cây biểu thức số học - Tr61

class Node:

def \_\_init\_\_(self, value, left=None, right=None):

self.value = value

self.left = left

self.right = right

def read\_tree(self):

x = input("Nhập giá trị cho nút: ")

operator\_list = ['+', '-', '\*', '/']

if x in operator\_list:

self.value = x

print(f"Nhập con trái của {x}: ")

self.left = Node(None).read\_tree()

print(f"Nhập con phải của {x}: ")

self.right = Node(None).read\_tree()

else:

self.value = int(x)

return self

def evaluate(self):

if self.left is None and self.right is None: # If leaf node

return self.value

else:

operator = self.value

left\_value = self.left.evaluate()

right\_value = self.right.evaluate()

if operator == '+':

return left\_value + right\_value

elif operator == '-':

return left\_value - right\_value

elif operator == '\*':

return left\_value \* right\_value

elif operator == '/':

try:

return left\_value / right\_value

except ZeroDivisionError:

print("Không thể chia cho 0!")

return 0

def inorder(node):

if node is not None:

inorder(node.left)

print(node.value, end=' ')

inorder(node.right)

# Example usage: + 2 \* 3 4

root = Node(None).read\_tree()

print("Cây biểu thức sau khi duyệt LNR là:")

inorder(root)

print("\nTrị của biểu thức là: ", root.evaluate())

## Mô phỏng trò chơi ca rô

Có 2 kiểu:

- Cắt tỉa alpha - beta

- Vét cạn

# Khởi tạo bàn cờ

board = [[' ' for \_ in range(3)] for \_ in range(3)]

# Vẽ bàn cờ

def draw\_board():

for row in board:

print('|'.join(row))

print('-' \* 7)

# Kiểm tra xem đã có người chiến thắng chưa

def check\_winner(row, col, player):

# Kiểm tra hàng ngang

if board[row][0] == board[row][1] == board[row][2] == player:

return True

# Kiểm tra hàng dọc

if board[0][col] == board[1][col] == board[2][col] == player:

return True

# Kiểm tra đường chéo chính

if board[0][0] == board[1][1] == board[2][2] == player:

return True

# Kiểm tra đường chéo phụ

if board[0][2] == board[1][1] == board[2][0] == player:

return True

return False

# Chạy trò chơi

def play\_game():

player = 'X' # Người chơi đầu tiên là X

while True:

draw\_board()

# Nhập vị trí đặt ký tự

row = int(input("Nhập hàng: "))

col = int(input("Nhập cột: "))

# Kiểm tra vị trí hợp lệ

if row < 0 or row >= 3 or col < 0 or col >= 3 or board[row][col] != ' ':

print("Vị trí không hợp lệ, vui lòng thử lại!")

continue

# Đặt ký tự vào bàn cờ

board[row][col] = player

# Kiểm tra xem đã có người chiến thắng chưa

if check\_winner(row, col, player):

draw\_board()

print("Người chơi", player, "thắng!")

break

# Chuyển lượt người chơi

player = 'O' if player == 'X' else 'X'

# Bắt đầu trò chơi

play\_game()

### Cắt tỉa Alpha – Beta

from random import choice

from math import inf

board = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]

def Gameboard(board):

chars = {1: 'X', -1: 'O', 0: ' '}

for x in board:

for y in x:

ch = chars[y]

print(ch, end=' ')

print('\n')

def Clearboard(board):

for x, row in enumerate(board):

for y, col in enumerate(row):

board[x][y] = 0

def winningPlayer(board, player):

conditions = [[board[0][0], board[0][1], board[0][2]],

[board[1][0], board[1][1], board[1][2]],

[board[2][0], board[2][1], board[2][2]],

[board[0][0], board[1][0], board[2][0]],

[board[0][1], board[1][1], board[2][1]],

[board[0][2], board[1][2], board[2][2]],

[board[0][0], board[1][1], board[2][2]],

[board[0][2], board[1][1], board[2][0]]]

if [player, player, player] in conditions:

return True

return False

def gameWon(board):

return winningPlayer(board, 1) or winningPlayer(board, -1)

def printResult(board):

if winningPlayer(board, 1):

print('X has won!\n')

elif winningPlayer(board, -1):

print('O has won!\n')

else:

print('Draw\n')

def blanks(board):

blank = []

for x, row in enumerate(board):

for y, col in enumerate(row):

if board[x][y] == 0:

blank.append([x, y])

return blank

def boardFull(board):

if len(blanks(board)) == 0:

return True

return False

def setMove(board, x, y, player):

board[x][y] = player

def playerMove(board):

e = True

moves = {1: [0, 0], 2: [0, 1], 3: [0, 2],

4: [1, 0], 5: [1, 1], 6: [1, 2],

7: [2, 0], 8: [2, 1], 9: [2, 2]}

while e:

try:

move = int(input('Enter a number between 1-9: '))

if move < 1 or move > 9:

print('Invalid Move! Try again!')

elif not (moves[move] in blanks(board)):

print('Invalid Move! Try again!')

else:

setMove(board, moves[move][0], moves[move][1], 1)

Gameboard(board)

e = False

except(KeyError, ValueError):

print('Invalid input! Try again!')

def abminimax(board, depth, alpha, beta, player):

row = -1

col = -1

if depth == 0 or gameWon(board):

if gameWon(board):

if winningPlayer(board, -1):

return (-1, row, col)

elif winningPlayer(board, 1):

return (1, row, col)

return (0, row, col)

else:

return (0, row, col)

elif player == 1:

maxv = -inf

for cell in blanks(board):

setMove(board, cell[0], cell[1], 1)

(m, min\_i, in\_j) = abminimax(board, depth - 1, alpha, beta, -1)

if m > maxv:

maxv = m

row = cell[0]

col = cell[1]

setMove(board, cell[0], cell[1], 0)

if maxv >= beta:

return (maxv, row, col)

if maxv > alpha:

alpha = maxv

return (maxv, row, col)

else:

minv = inf

for cell in blanks(board):

setMove(board, cell[0], cell[1], -1)

(m, max\_i, max\_j) = abminimax(board, depth - 1, alpha, beta, 1)

if m < minv:

minv = m

row = cell[0]

col = cell[1]

setMove(board, cell[0], cell[1], 0)

if minv <= alpha:

return (minv, row, col)

if minv < beta:

beta = minv

return (minv, row, col)

def playGame():

Clearboard(board)

Gameboard(board)

while True:

playerMove(board)

if gameWon(board):

printResult(board)

break

elif boardFull(board):

print("It's a draw!")

break

print("Computer's turn:")

(m, qx, qy) = abminimax(board, 9, -inf, inf, -1)

setMove(board, qx, qy, -1)

Gameboard(board)

if gameWon(board):

printResult(board)

break

playGame()

### Vét cạn định trị:

import numpy as np

class Caro:

def \_\_init\_\_(self):

self.board = np.zeros((3, 3)) # Lưới 3x3 biểu diễn bàn cờ

self.player = 1 # Người chơi hiện tại (1 hoặc -1)

def move(self, x, y):

if self.board[x][y] == 0:

self.board[x][y] = self.player

self.player \*= -1 # Thay đổi người chơi

def is\_leaf(self):

# Một nút là nút lá nếu bàn cờ đã đầy hoặc một người chơi đã thắng

return np.abs(self.board).sum() == 9 or self.winner() != 0

def payoff(self):

# Giả sử người chơi 1 là người chơi 'MAX', người chơi -1 là 'MIN'

return self.winner()

def winner(self):

# Xác định người chiến thắng: 1 cho người chơi 1, -1 cho người chơi -1, 0 nếu chưa ai thắng

for i in range(3):

if abs(sum(self.board[i, :])) == 3: # Kiểm tra hàng

return self.board[i, 0]

if abs(sum(self.board[:, i])) == 3: # Kiểm tra cột

return self.board[0, i]

if abs(sum([self.board[i, i] for i in range(3)])) == 3: # Kiểm tra đường chéo chính

return self.board[0, 0]

if abs(sum([self.board[i, 2 - i] for i in range(3)])) == 3: # Kiểm tra đường chéo phụ

return self.board[0, 2]

return 0 # Chưa ai thắng

def minimax(self):

if self.is\_leaf():

return self.payoff(), None

if self.player == 1: # Người chơi 'MAX'

max\_payoff = -np.inf

action = None

for x, y in self.valid\_moves():

self.move(x, y)

payoff, \_ = self.minimax()

if payoff > max\_payoff:

max\_payoff = payoff

action = (x, y)

self.undo(x, y) # Quay lại trạng thái trước đó

return max\_payoff, action

else: # Người chơi 'MIN'

min\_payoff = np.inf

action = None

for x, y in self.valid\_moves():

self.move(x, y)

payoff, \_ = self.minimax()

if payoff < min\_payoff:

min\_payoff = payoff

action = (x, y)

self.undo(x, y) # Quay lại trạng thái trước đó

return min\_payoff, action

def valid\_moves(self):

return [(i, j) for i in range(3) for j in range(3) if self.board[i, j] == 0]

def undo(self, x, y):

self.board[x][y] = 0

self.player \*= -1

# Tạo một đối tượng game mới

game = Caro()

# Chơi một nước đi

game.move(0, 0)

# Sử dụng thuật toán Minimax để tìm nước đi tốt nhất tiếp theo

payoff, action = game.minimax()

print(f"Payoff: {payoff}, Best action: {action}")

# Chơi nước đi tốt nhất

if action is not None:

game.move(\*action)

# In bàn cờ

print(game.board)

# **KỸ THUẬT TÌM KIẾM ĐỊA PHƯƠNG**

## Tìm số nguyên tố

***Kiểm Tra Số Nguyên Tố Sử Dụng Kỹ Thuật Tìm Kiếm Địa Phương: Viết chương trình Python sử dụng kỹ thuật tìm kiếm địa phương để kiểm tra xem một số nguyên dương n có phải là số nguyên tố không.***

Để kiểm tra xem một số nguyên dương `n` có phải là số nguyên tố không, chúng ta chỉ cần kiểm tra xem `n` có ước số nào trong khoảng từ `2` đến căn bậc hai của `n` không. Nếu không tìm thấy ước số nào trong khoảng này, `n` là số nguyên tố.

Đây là một kỹ thuật tìm kiếm địa phương (local search technique) vì chúng ta chỉ kiểm tra một tập hợp hữu hạn các ứng viên (các số trong khoảng từ `2` đến `sqrt(n)`) thay vì kiểm tra tất cả các số từ `2` đến `n-1`.

Dưới đây là chương trình Python để kiểm tra xem một số nguyên dương có phải là số nguyên tố sử dụng kỹ thuật này:

import math

def is\_prime(n):

# Số nhỏ hơn 2 không phải là số nguyên tố

if n < 2:

return False

# Kiểm tra xem n có ước số nào trong khoảng từ 2 đến căn bậc hai của n không

for i in range(2, int(math.sqrt(n)) + 1):

if n % i == 0:

return False

# Nếu không tìm thấy ước số nào, n là số nguyên tố

return True

# Nhập một số nguyên dương

n = int(input("Nhập một số nguyên dương: "))

# Kiểm tra và in kết quả

if is\_prime(n):

print(f"{n} là số nguyên tố.")

else:

print(f"{n} không phải là số nguyên tố.")

Trong chương trình này, hàm `is\_prime` kiểm tra xem số `n` có phải là số nguyên tố không bằng cách kiểm tra xem nó có ước số nào trong khoảng từ `2` đến căn bậc hai của `n` không.

## Bài toán cây phủ tối thiểu Minimum Spanning Tree (MST)

*# 3. VCT tìm cây phủ tối thiểu bằng kỹ thuật Tìm kiếm địa phương (TKĐP) - Tr76*

class Graph:

def \_\_init\_\_(self, vertices):

self.V = vertices

self.graph = []

self.nodes = {}

def add\_edge(self, u, v, w):

if u not in self.nodes:

self.nodes[u] = len(self.nodes)

if v not in self.nodes:

self.nodes[v] = len(self.nodes)

self.graph.append([self.nodes[u], self.nodes[v], w])

def find(self, parent, i):

if parent[i] == i:

return i

return self.find(parent, parent[i])

def union(self, parent, rank, x, y):

xroot = self.find(parent, x)

yroot = self.find(parent, y)

if rank[xroot] < rank[yroot]:

parent[xroot] = yroot

elif rank[xroot] > rank[yroot]:

parent[yroot] = xroot

else:

parent[yroot] = xroot

rank[xroot] += 1

def local\_search(self):

result = []

self.graph = sorted(self.graph, key=lambda item: item[2])

parent = []

rank = []

for node in range(self.V):

parent.append(node)

rank.append(0)

e = 0

while len(result) < self.V - 1:

u, v, w = self.graph[e]

e += 1

x = self.find(parent, u)

y = self.find(parent, v)

if x != y:

result.append([list(self.nodes.keys())[list(self.nodes.values()).index(u)],

list(self.nodes.keys())[list(self.nodes.values()).index(v)], w])

self.union(parent, rank, x, y)

return result

g = Graph(5)

g.add\_edge('A', 'B', 3)

g.add\_edge('A', 'C', 4)

g.add\_edge('A', 'D', 2)

g.add\_edge('A', 'E', 7)

g.add\_edge('B', 'C', 4)

g.add\_edge('B', 'D', 6)

g.add\_edge('B', 'E', 3)

g.add\_edge('C', 'D', 5)

g.add\_edge('C', 'E', 8)

g.add\_edge('D', 'E', 6)

results = g.local\_search()

total\_weight = 0

for u, v, w in results:

print(f"{u}{v} = {w}")

total\_weight += w

print(f"Giá của chu trình : {total\_weight}")

## Bài toán tìm đường đi ngắn nhất (TSP) của người giao hàng

# VTC tìm đường đi ngắn nhất bằng TKĐP - Tr77

import random

def total\_distance(tour, dist\_matrix):

total = 0

for i in range(len(tour) - 1):

total += dist\_matrix[tour[i]][tour[i + 1]]

total += dist\_matrix[tour[-1]][tour[0]] # return to the start

return total

def local\_search\_tsp(dist\_matrix):

n = len(dist\_matrix)

tour = list(range(n)) # initial tour

random.shuffle(tour) # randomize the initial tour

improved = True

while improved:

improved = False

for i in range(n):

for j in range(i + 2, n - 1):

new\_tour = tour[:i] + tour[i:j][::-1] + tour[j:] # reverse segment i-j

if total\_distance(new\_tour, dist\_matrix) < total\_distance(tour, dist\_matrix):

tour = new\_tour

improved = True

if not improved:

break

return tour

# example distance matrix

dist\_matrix = [

[0, 10, 15, 20],

[10, 0, 35, 25],

[15, 35, 0, 30],

[20, 25, 30, 0]

]

tour = local\_search\_tsp(dist\_matrix)

print("Best tour:", tour)

print("Total distance:", total\_distance(tour, dist\_matrix))

# **TẬP TIN**

## TẬP TIN TUẦN TỰ

### VCT mô phỏng CTDL và các phép toán cơ bản như: tìm, thêm, xóa, sửa cho

### tập tin tuần tự.

1. Tạo file `data.txt` chứa dữ liệu, mỗi dòng là một mục.

2. Viết chương trình Python để đọc và thao tác với file này.

def doc\_tap\_tin(ten\_tap\_tin):

try:

with open(ten\_tap\_tin, 'r') as file:

return file.readlines()

except FileNotFoundError:

return []

def ghi\_tap\_tin(ten\_tap\_tin, du\_lieu):

with open(ten\_tap\_tin, 'w') as file:

file.writelines(du\_lieu)

def them(du\_lieu, noi\_dung):

du\_lieu.append(noi\_dung + '\n')

def tim(du\_lieu, noi\_dung):

return [x for x in du\_lieu if noi\_dung in x]

def xoa(du\_lieu, noi\_dung):

return [x for x in du\_lieu if noi\_dung not in x]

def sua(du\_lieu, cu, moi):

return [moi + '\n' if x.strip() == cu else x for x in du\_lieu]

# Đọc dữ liệu từ tập tin

ten\_tap\_tin = 'data.txt'

du\_lieu = doc\_tap\_tin(ten\_tap\_tin)

# Thêm dữ liệu

them(du\_lieu, 'Dòng mới')

# Tìm dữ liệu

tim\_kiem = tim(du\_lieu, 'Dòng')

print('Kết quả tìm kiếm:', tim\_kiem)

# Xóa dữ liệu

du\_lieu = xoa(du\_lieu, 'Dòng cần xóa')

# Sửa dữ liệu

du\_lieu = sua(du\_lieu, 'Dòng cần sửa', 'Dòng đã sửa')

# Ghi dữ liệu trở lại tập tin

ghi\_tap\_tin(ten\_tap\_tin, du\_lieu)

Giải thích chương trình:

- `doc\_tap\_tin`: Đọc nội dung của tập tin vào một danh sách.

- `ghi\_tap\_tin`: Ghi nội dung từ danh sách ra tập tin.

- `them`: Thêm một dòng mới vào danh sách.

- `tim`: Tìm các dòng chứa nội dung cần tìm và trả về danh sách kết quả.

- `xoa`: Xóa các dòng chứa nội dung cần xóa khỏi danh sách.

- `sua`: Thay thế nội dung cần sửa bằng nội dung mới trong danh sách.

Chú ý: Trước khi chạy chương trình, hãy tạo một tập tin `data.txt` và thêm một số dòng dữ liệu vào đó để thử nghiệm.

## TẬP TIN BẢN BĂM

### VCT mô phỏng CTDL và các phép toán cơ bản như: tìm, thêm, xóa, sửa cho tập tin băm.

import json

def doc\_tap\_tin(ten\_tap\_tin):

try:

with open(ten\_tap\_tin, 'r') as file:

return json.load(file)

except FileNotFoundError:

return {}

def ghi\_tap\_tin(ten\_tap\_tin, du\_lieu):

with open(ten\_tap\_tin, 'w') as file:

json.dump(du\_lieu, file)

def them(du\_lieu, khoa, gia\_tri):

du\_lieu[khoa] = gia\_tri

def tim(du\_lieu, khoa):

return du\_lieu.get(khoa)

def xoa(du\_lieu, khoa):

if khoa in du\_lieu:

del du\_lieu[khoa]

def sua(du\_lieu, khoa, gia\_tri\_moi):

if khoa in du\_lieu:

du\_lieu[khoa] = gia\_tri\_moi

# Đọc dữ liệu từ tập tin

ten\_tap\_tin = 'data.json'

du\_lieu = doc\_tap\_tin(ten\_tap\_tin)

# Thêm dữ liệu

them(du\_lieu, 'khoa1', 'giá trị 1')

# Tìm dữ liệu

gia\_tri = tim(du\_lieu, 'khoa1')

print('Giá trị tìm thấy:', gia\_tri)

# Sửa dữ liệu

sua(du\_lieu, 'khoa1', 'giá trị mới')

# Xóa dữ liệu

xoa(du\_lieu, 'khoa\_xoa')

# Ghi dữ liệu trở lại tập tin

ghi\_tap\_tin(ten\_tap\_tin, du\_lieu)

---

# **ĐỀ THAM KHẢO HỌC THÊM**

## Xây dựng lớp HinhHoc:

Viết một lớp Python tên là HinhHoc để tính diện tích và chu vi của hình tròn, hình vuông, và hình chữ nhật. Lớp nên có các phương thức để thiết lập kích thước và tính toán diện tích, chu vi của các hình.

import math

class HinhHoc:

def \_\_init\_\_(self):

pass

def dien\_tich\_hinh\_tron(self, r):

return math.pi \* r \* r

def chu\_vi\_hinh\_tron(self, r):

return 2 \* math.pi \* r

def dien\_tich\_hinh\_vuong(self, a):

return a \* a

def chu\_vi\_hinh\_vuong(self, a):

return 4 \* a

def dien\_tich\_hinh\_chu\_nhat(self, a, b):

return a \* b

def chu\_vi\_hinh\_chu\_nhat(self, a, b):

return 2 \* (a + b)

hinh = HinhHoc()

print(hinh.dien\_tich\_hinh\_tron(5))

print(hinh.chu\_vi\_hinh\_vuong(4))

## Tìm ước chung lớn nhất bằng thuật toán Euclid:

Viết chương trình Python để tìm ước chung lớn nhất (GCD) của hai số nguyên dương bằng cách sử dụng thuật toán Euclid (có thể sử dụng đệ qui hoặc vòng lặp).

def gcd(a, b):

while b:

a, b = b, a % b

return a

print(gcd(60, 48))

## Tìm các dãy con liên tiếp có tổng lớn nhất trong mảng:

Cho một mảng số nguyên, viết chương trình Python để tìm dãy con liên tiếp có tổng lớn nhất sử dụng kỹ thuật chia để trị.

def max\_subarray\_sum(arr):

max\_so\_far = max\_ending\_here = arr[0]

for x in arr[1:]:

max\_ending\_here = max(x, max\_ending\_here + x)

max\_so\_far = max(max\_so\_far, max\_ending\_here)

return max\_so\_far

print(max\_subarray\_sum([-2, 1, -3, 4, -1, 2, 1, -5, 4]))

## Sắp xếp Shell Sort (Class Sapxep\_TheoKhoangCach kế thừa từ Danhsach):

Cài đặt thuật toán Shell Sort trong Python bằng cách tạo một lớp tên Sapxep\_TheoKhoangCach kế thừa từ lớp Danhsach.

def shell\_sort(arr):

n = len(arr)

gap = n // 2

while gap > 0:

for i in range(gap, n):

temp = arr[i]

j = i

while j >= gap and arr[j - gap] > temp:

arr[j] = arr[j - gap]

j -= gap

arr[j] = temp

gap //= 2

return arr

print(shell\_sort([12, 34, 54, 2, 3]))

## Đếm số lượng đảo ngược trong mảng:

Cho một mảng số nguyên, viết một chương trình Python để đếm số lượng cặp (i, j) sao cho i < j và a[i] > a[j]. Sử dụng kỹ thuật chia để trị để giải quyet bài toán này.

[Code này sẽ khá dài, bạn có thể tìm hiểu "Count Inversions in an array" để có thêm thông tin.]

## Bài toán N-Queens sử dụng kỹ thuật quay lui:

Viết một chương trình Python để giải quyet bài toán N-Queens, trong đó bạn cần đặt N quân hậu trên bàn cờ NxN sao cho không có hai quân hậu nào đe dọa nhau.

[Đây cũng là một bài toán phức tạp, tôi khuyến khích bạn tìm hiểu "N-Queens Problem" để xem các ví dụ cụ thể về cách giải quyet bài toán này.]

## Áp dụng kỹ thuật tham lam để tối ưu hóa việc sử dụng tài nguyên máy tính:

Cho một tập hợp các công việc, mỗi công việc đều yêu cầu một lượng tài nguyên nhất định để thực thi. Viết chương trình Python để lựa chọn tập hợp các công việc sao cho tối đa số lượng công việc được thực thi mà không vượt quá tài nguyên có sẵn.

[Vì đây là một bài toán tổng quát, tôi khuyến khích bạn tìm hiểu "Greedy Algorithms" để xem các ví dụ và cách tiếp cận bài toán này.]

## Phương pháp tìm kiếm Interpolation Search:

Cài đặt thuật toán tìm kiếm nội suy (Interpolation Search) trong Python. So sánh hiệu suất của nó với tìm kiếm tuyến tính và tìm kiếm nhị phân.

def interpolation\_search(arr, x):

lo = 0

hi = len(arr) - 1

while lo <= hi and x >= arr[lo] and x <= arr[hi]:

pos = lo + int(((x - arr[lo]) \* (hi - lo)) / (arr[hi] - arr[lo]))

if arr[pos] == x:

return pos

if arr[pos] < x:

lo = pos + 1

else:

hi = pos - 1

return -1

arr = [10, 12, 13, 16, 18, 19, 20, 21, 22, 23, 24, 33, 35, 42, 47]

print(interpolation\_search(arr, 18))

## Cài đặt B-Tree và thực hiện các thao tác cơ bản:

Xây dựng cấu trúc dữ liệu B-Tree trong Python và cài đặt các thao tác như chèn, xóa, và tìm kiếm.

[Đây là một bài toán phức tạp và cần nhiều code. Tôi khuyến khích bạn tìm hiểu "B-Tree" để xem các ví dụ và cách cài đặt.]

## Tìm chu trình Hamiltonian trong đồ thị:

Cho một đồ thị, viết chương trình Python để tìm một chu trình Hamiltonian (một chu trình đi qua tất cả các đỉnh một lần) sử dụng kỹ thuật quay lui.

[Tương tự như các bài toán phức tạp khác, tôi khuyến khích bạn tìm hiểu "Hamiltonian Cycle" để xem các ví dụ và cách giải quyet bài toán này.]

Để tìm một chu trình Hamiltonian trong đồ thị, chúng ta có thể sử dụng kỹ thuật quay lui. Ý tưởng là đi qua từng đỉnh của đồ thị, và đối với mỗi đỉnh, thử xem nếu thêm nó vào chu trình hiện tại có thỏa mãn các điều kiện của chu trình Hamiltonian hay không (ví dụ, không có đỉnh nào được truy cập hai lần, và đỉnh cuối cùng phải kết nối với đỉnh đầu tiên). Nếu không, chúng ta sẽ "quay lui" và thử một đỉnh khác.

Dưới đây là chương trình Python để tìm chu trình Hamiltonian sử dụng kỹ thuật quay lui:

```python

def is\_valid(v, pos, path, graph):

# Kiểm tra xem có cạnh từ đỉnh hiện tại tới v hay không

# và v chưa xuất hiện trong đường dẫn.

if graph[path[pos-1]][v] == 0:

return False

# Kiểm tra xem v đã xuất hiện trong đường dẫn chưa

if v in path:

return False

return True

def hamiltonian\_cycle\_util(graph, path, pos):

# Điều kiện cơ bản: nếu chu trình chứa tất cả các đỉnh

if pos == len(graph):

# Nếu có cạnh từ đỉnh cuối cùng tới đỉnh đầu tiên

# thì trả về True để báo hiệu tìm thấy chu trình Hamiltonian

return graph[path[pos-1]][path[0]] == 1

# Thử thêm các đỉnh còn lại vào đường dẫn hiện tại

for v in range(1, len(graph)):

if is\_valid(v, pos, path, graph):

path[pos] = v

if hamiltonian\_cycle\_util(graph, path, pos + 1):

return True

# Xóa đỉnh hiện tại khỏi đường dẫn và quay lui

path[pos] = -1

return False

def hamiltonian\_cycle(graph):

path = [-1] \* len(graph)

path[0] = 0 # Bắt đầu từ đỉnh 0

if not hamiltonian\_cycle\_util(graph, path, 1):

print("Không tìm thấy chu trình Hamiltonian")

return

print("Chu trình Hamiltonian:")

for vertex in path:

print(vertex, end=' ')

print(path[0]) # In lại đỉnh đầu tiên để tạo chu trình

# Ví dụ về đồ thị sử dụng ma trận kề

graph = [

[0, 1, 1, 1, 0, 0],

[1, 0, 1, 0, 1, 0],

[1, 1, 0, 1, 0, 1],

[1, 0,

1, 0, 0, 0],

[0, 1, 0, 0, 0, 1],

[0, 0, 1, 0, 1, 0]

]

# Tìm và in chu trình Hamiltonian

hamiltonian\_cycle(graph)

```

Trong chương trình này:

- `is\_valid` kiểm tra xem có thể thêm một đỉnh cụ thể vào đường dẫn hiện tại không.

- `hamiltonian\_cycle\_util` là hàm đệ quy thực hiện việc tìm kiếm chu trình Hamiltonian sử dụng kỹ thuật quay lui.

- `hamiltonian\_cycle` khởi tạo đường dẫn và gọi hàm đệ quy.

---

**CẤU TRÚC DỮ LIỆU**

Dưới đây là một số đoạn mã Python đơn giản để minh họa cho một số cấu trúc dữ liệu bạn đã đề cập. Lưu ý rằng, các ví dụ này chỉ mang tính chất minh họa và không đại diện cho cách thực hiện tốt nhất cho từng cấu trúc.

**1. Vector (Mảng 1 chiều):**

vector = [1, 2, 3, 4, 5]

print("Vector:", vector)

**2. Bảng (Mảng 2 chiều):**

bang = [

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

]

print("Bảng:")

for hang in bang:

print(hang)

**3. Ma trận vuông cấp N:**

n = int(input("Nhập cấp của ma trận vuông: "))

ma\_tran\_vuong = [[0]\*n for \_ in range(n)]

for i in range(n):

for j in range(n):

ma\_tran\_vuong[i][j] = int(input(f"Nhập phần tử ở hàng {i+1}, cột {j+1}: "))

print("Ma trận vuông:")

for hang in ma\_tran\_vuong:

print(hang)

**4. Danh sách liên kết (Linked List):**

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class LinkedList:

def \_\_init\_\_(self):

self.head = None

def them(self, data):

new\_node = Node(data)

new\_node.next = self.head

self.head = new\_node

def in\_danh\_sach(self):

temp = self.head

while temp:

print(temp.data, end=" ")

temp = temp.next

ds = LinkedList()

ds.them(1)

ds.them(2)

ds.them(3)

print("Danh sách liên kết:")

ds.in\_danh\_sach()

**5. Cây nhị phân (Binary Tree):**

class Node:

def \_\_init\_\_(self, key):

self.left = None

self.right = None

self.val = key

def in\_order\_traversal(root):

if root:

in\_order\_traversal(root.left)

print(root.val, end=" ")

in\_order\_traversal(root.right)

r = Node(1)

r.left = Node(2)

r.right = Node(3)

r.left.left = Node(4)

r.left.right = Node(5)

print("Cây nhị phân (in-order traversal):")

in\_order\_traversal(r)

**6. Đồ thị (Graph):**

class Graph:

def \_\_init\_\_(self):

self.graph = {}

def them\_dinh(self, dinh):

if dinh not in self.graph:

self.graph[dinh] = []

def them\_canh(self, dinh1, dinh2):

self.graph[dinh1].append(dinh2)

self.graph[dinh2].append(dinh1)

def in\_do\_thi(self):

for dinh, ke in self.graph.items():

print(f"{dinh}: {ke}")

g = Graph()

g.them\_dinh(1)

g.them\_dinh(2)

g.them\_dinh(3)

g.them\_canh(1, 2)

g.them\_canh(2, 3)

print("Đồ thị:")

g.in\_do\_thi()

**7. Stack:**

stack = []

stack.append(1)

stack.append(2)

stack.append(3)

print("Stack sau khi thêm phần tử:")

print(stack)

print("Pop phần tử từ stack:")

print(stack.pop())

print("Stack sau khi pop:")

print(stack)

**8. Queue:**

queue = []

queue.append(1)

queue.append(2)

queue.append(3)

print("Queue sau khi thêm phần tử:")

print(queue)

print("Dequeue phần tử từ queue:")

print(queue.pop(0))

print("Queue sau khi dequeue:")

print(queue)

**9. Hash Table/Hash Map:**

hash\_table = {}

hash\_table['one'] = 1

hash\_table['two'] = 2

hash\_table['three'] = 3

print("Hash Table:")

for key, value in hash\_table.items():

print(f"{key}: {value}")

Lưu ý rằng, ví dụ trên chỉ minh họa cách sử dụng cơ bản của các cấu trúc dữ liệu và không đại diện cho những cách thực hiện tối ưu nhất. Trong thực tế, bạn sẽ cần tìm hiểu sâu hơn về từng cấu trúc dữ liệu và cách sử dụng chúng một cách hiệu quả trong các vấn đề cụ thể.

Dưới đây là một ví dụ cụ thể về việc sử dụng cấu trúc dữ liệu Queue để giải quyết vấn đề hàng đợi của người dùng:

### Vấn đề:

Một ngân hàng có nhiều quầy giao dịch. Người dùng sẽ đến ngân hàng và xếp hàng đợi để được giao dịch. Bạn cần mô phỏng hàng đợi của người dùng và xử lý hàng đợi để biết ai sẽ được giao dịch tiếp theo.

### Giải pháp sử dụng Queue:

Sử dụng cấu trúc Queue để giữ hàng đợi của người dùng. Khi một người dùng mới đến, họ sẽ được thêm vào cuối hàng đợi. Người dùng ở đầu hàng đợi sẽ là người tiếp theo được giao dịch.

```python

class Queue:

def \_\_init\_\_(self):

self.queue = []

def enqueue(self, item):

self.queue.append(item)

def dequeue(self):

if len(self.queue) < 1:

return None

return self.queue.pop(0)

def size(self):

return len(self.queue)

# Khởi tạo hàng đợi

hang\_doi = Queue()

# Người dùng đến và xếp hàng

hang\_doi.enqueue("Người dùng 1")

hang\_doi.enqueue("Người dùng 2")

hang\_doi.enqueue("Người dùng 3")

# Xử lý hàng đợi

while hang\_doi.size() > 0:

nguoi\_dung = hang\_doi.dequeue()

print(f"{nguoi\_dung} đang được giao dịch.")

# Tại đây có thể thêm mã giả lập thời gian giao dịch ví dụ: import time; time.sleep(2)

Ở ví dụ trên, chúng ta tạo ra một class `Queue` để mô phỏng cấu trúc hàng đợi. Người dùng sẽ được thêm vào cuối hàng đợi khi họ đến ngân hàng (`enqueue`). Người dùng ở đầu hàng đợi sẽ được gọi ra để giao dịch và được xóa khỏi hàng đợi (`dequeue`). Chương trình sẽ tiếp tục xử lý cho đến khi không còn người dùng nào trong hàng đợi.

Ví dụ này minh họa cách sử dụng cấu trúc Queue để giải quyết một vấn đề thực tế cụ thể. Tương tự, bạn có thể sử dụng các cấu trúc dữ liệu khác như Stack, Tree, Graph, Hash Table để giải quyết các vấn đề khác nhau trong lập trình.

[**CVT Giải phương trình đệ qui dạng tổng quát với hàm tiến triển là hàm nhân**](#_Toc139466337)

[**Xây dựng lớp Danhsach – Code ok**](#_Toc139466338)

[**Xây dựng một lớp trong ngôn ngữ lập trình của bạn lựa chọn để quản lý một danh sách liên kết các số nguyên. Trong lớp này, bạn phải xây dựng các phương thức cho việc thêm, tìm kiếm, sửa, xóa và xuất các phần tử.**](#_Toc139466339)

[Merge Sort (Class Sapxep kethua Danhsach) - Ok](#_Toc139466340)

[**CÁC PHƯƠNG PHÁP SẮP XẾP THUỘC LỚP HÀM N2**](#_Toc139466341)

[Phương pháp chọn (Selection Sort) (Class Sapxep\_N^2 kethua Danhsach)](#_Toc139466342)

[Phương pháp chèn (Insertion Sort) (Class Sapxep\_N^2 kethua Danhsach)](#_Toc139466343)

[Phương pháp nổi bọt (Bubble Sort) (Class Sapxep\_N^2 kethua Danhsach)](#_Toc139466344)

[**CÁC PHƯƠNG PHÁP SẮP XẾP THUỘC LỚP HÀM NLogN**](#_Toc139466345)

[Phương pháp Heap Sort (Class Sapxep\_NLogN kethua Danhsach) - Ok](#_Toc139466346)

[Phương pháp Quick Sort (Class Sapxep\_NLogN kethua Danhsach) - Ok](#_Toc139466347)

[**CÁC PHƯƠNG PHÁP SẮP XẾP THUỘC LỚP HÀM N**](#_Toc139466348)

[Phương pháp Radix Sort (Class Sapxep\_N kethua Danhsach) - Ok](#_Toc139466349)

[Phương pháp Bin Sort (Class Sapxep\_N kethua Danhsach)](#_Toc139466350)

[Trường hợp đơn giản](#_Toc139466351)

[Trường hợp tổng quát](#_Toc139466352)

[Trường hợp tập giá trị lớn](#_Toc139466353)

[**CÁC GIẢI THUẬT TÌM KIẾM NỘI (Xây dựng lớp TimKiem kế thừa từ lớp DanhSach)**](#_Toc139466354)

[Tìm kiếm tuyến tính (Linear Search)](#_Toc139466355)

[Không đệ qui](#_Toc139466356)

[Đệ qui - Ok](#_Toc139466357)

[Tìm kiếm nhị phân (Binary Search)](#_Toc139466358)

[Không đệ qui](#_Toc139466359)

[Đệ qui](#_Toc139466360)

[**KỸ THUẬT CHIA ĐỂ TRỊ**](#_Toc139466361)

[Bài toán nhân các số nguyên lớn](#_Toc139466362)

[Thuật toán thông thường](#_Toc139466363)

[Thuật toán áp dụng kỹ thuật chia để trị](#_Toc139466364)

[Nhân hai ma trận:](#_Toc139466365)

[Viết một chương trình cho phép nhân hai ma trận vuông cấp N lớn sử dụng thuật toán chia để trị.](#_Toc139466366)

[Nhân hai ma trận nhiều giá trị:](#_Toc139466367)

[Tìm UCLN và BCNN sử dụng kỹ thuật chia để trị:](#_Toc139466368)

[Tìm giá trị nhỏ nhất trong ma trận vuông cấp N bằng kỹ thuật chia để trị](#_Toc139466369)

[TH1: Đệ qui, chia hàng](#_Toc139466370)

[Trường hợp nhận số chẵn](#_Toc139466371)

[Có trường hợp nếu nhập số lẻ, chẵn:](#_Toc139466372)

[TH2 (Khó): Đệ qui, Chia thành 4 ô vuông](#_Toc139466373)

[Trường hợp số chẵn](#_Toc139466374)

[Trường hợp số lẽ, chẵn:](#_Toc139466375)

[**KỸ THUẬT THAM ĂN**](#_Toc139466376)

[Thuật toán thông thường](#_Toc139466377)

[Thuật toán áp dụng kỹ thuật tham ăn](#_Toc139466378)

[ATM:](#_Toc139466379)

[-----Không giới hạn số tờ-----](#_Toc139466380)

[----Giới hạn số tờ là 1-----](#_Toc139466381)

[-----Số tờ tùy chọn -----](#_Toc139466382)

[Bài toán tìm đường đi ngắn nhất (TSP) của người giao hàng](#_Toc139466383)

[Bài toán cá ba lô (knapsack) - Kỹ thuật tham ăn](#_Toc139466384)

[Cái ba lô:](#_Toc139466385)

[----Không giới hạn-----](#_Toc139466386)

[-------Tùy chọn theo số lượng-----](#_Toc139466387)

[-----Mỗi đồ vật chỉ chọn 1---](#_Toc139466388)

[**KỸ THUẬT QUI HOẠCH ĐỘNG (Dynamic programming)**](#_Toc139466389)

[Tính số tổ hợp](#_Toc139466390)

[Bằng thuật toán đệ qui, áp dụng kỹ thuật qui hoạch động với CTDL là bảng](#_Toc139466391)

[Week7\_Baitap2\_Mang2chieu:](#_Toc139466392)

[Bằng thuật toán đệ qui, áp dụng kỹ thuật qui hoạch động với CTDL là vector](#_Toc139466393)

[Week7\_Baitap2\_Mang1Chieu:](#_Toc139466394)

[Bài toán cái ba lô - Week7\_Baitap3:](#_Toc139466395)

[TH1: Chọn đồ vắn có đơn giá cao nhất cho đến khi không thể chọn thêm đồ vật nào class Do\_vat:](#_Toc139466396)

[TH2: Có số lượng](#_Toc139466397)

[TH3: Chọn tối đa một đồ vật của từng loại](#_Toc139466398)

[**KỸ THUẬT QUAY LUI**](#_Toc139466399)

[Định trị cây biểu thức số học](#_Toc139466400)

[Mô phỏng trò chơi ca rô](#_Toc139466401)

[Cắt tỉa Alpha – Beta](#_Toc139466402)

[Vét cạn định trị:](#_Toc139466403)

[**KỸ THUẬT TÌM KIẾM ĐỊA PHƯƠNG**](#_Toc139466404)

[Tìm số nguyên tố](#_Toc139466405)

[Bài toán cây phủ tối thiểu Minimum Spanning Tree (MST)](#_Toc139466406)

[Bài toán tìm đường đi ngắn nhất (TSP) của người giao hàng](#_Toc139466407)

[**TẬP TIN**](#_Toc139466408)

[TẬP TIN TUẦN TỰ](#_Toc139466409)

[VCT mô phỏng CTDL và các phép toán cơ bản như: tìm, thêm, xóa, sửa cho](#_Toc139466410)

[tập tin tuần tự.](#_Toc139466411)

[TẬP TIN BẢN BĂM](#_Toc139466412)

[VCT mô phỏng CTDL và các phép toán cơ bản như: tìm, thêm, xóa, sửa cho tập tin băm.](#_Toc139466413)

[**ĐỀ THAM KHẢO HỌC THÊM**](#_Toc139466414)

[Xây dựng lớp HinhHoc:](#_Toc139466415)

[Tìm ước chung lớn nhất bằng thuật toán Euclid:](#_Toc139466416)

[Tìm các dãy con liên tiếp có tổng lớn nhất trong mảng:](#_Toc139466417)

[Sắp xếp Shell Sort (Class Sapxep\_TheoKhoangCach kế thừa từ Danhsach):](#_Toc139466418)

[Đếm số lượng đảo ngược trong mảng:](#_Toc139466419)

[Bài toán N-Queens sử dụng kỹ thuật quay lui:](#_Toc139466420)

[Áp dụng kỹ thuật tham lam để tối ưu hóa việc sử dụng tài nguyên máy tính:](#_Toc139466421)

[Phương pháp tìm kiếm Interpolation Search:](#_Toc139466422)

[Cài đặt B-Tree và thực hiện các thao tác cơ bản:](#_Toc139466423)

[Tìm chu trình Hamiltonian trong đồ thị:](#_Toc139466424)

**Hàm nhân**

import math  
def f(b, n):  
 return b \*\* n  
def d(n):  
 return n # Hàm d(n) = n  
def ktraHamNhan(d, m, n):  
 return d(m \* n) == d(m) \* d(n)  
def T(a, b, n):  
 if ktraHamNhan(d, a, b):  
 if a > f(b, n):  
 n = int(math.log(a, b))  
 print(f"T(n) = O(n^{n})")  
 elif a < f(b, n):  
 n = int(math.log(f(b, n), b))  
 print(f"T(n) = O(n^{n})")  
 else:  
 n = int(math.log(a, b))  
 print(f"T(n) = O((n^{n})log{b}n)")  
 else:  
 print("d(n) không phải là hàm nhân. Không thể giải phương trình.")

a = int(input("Nhâp a = "))  
b = int(input("Nhập b = "))  
somu\_n = int(input("Nhập số mũ của n = "))  
T(a, b, somu\_n)

**Xây dụng lớp DanhSach**

class Danhsach:  
 def \_\_init\_\_(self):  
 self.ds = []  
 def nhap(self):  
 n = int(input("Nhập số lượng phần tử: "))  
 for i in range(n):  
 self.ds.append(int(input(f"Nhập phần tử thứ {i + 1}: ")))  
 def xuat(self):  
 for i in self.ds:  
 print(i, end=' ')  
 print()  
 def tim(self, x):  
 if x in self.ds:  
 return self.ds.index(x)  
 else:  
 return -1  
 def them(self, x):  
 self.ds.append(x)  
  
 def xoa(self, x):  
 if x in self.ds:  
 self.ds.remove(x)  
 def sua(self, x, y):  
 for i in range(len(self.ds)):  
 if self.ds[i] == x:  
 self.ds[i] = y

**MergeSort:**

class SapXep(Danhsach):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 @staticmethod  
 def **merge\_sort**(arr):  
 n = len(arr)  
 if n <= 1:  
 return arr  
 mid = n // 2  
 left = arr[:mid]  
 right = arr[mid:]  
 return SapXep().merge(SapXep.merge\_sort(left), SapXep.merge\_sort(right))  
 def **merge**(self, left, right):  
 result = []  
 i = j = 0  
 while i < len(left) and j < len(right):  
 if left[i] < right[j]:  
 result.append(left[i])  
 i += 1  
 else:  
 result.append(right[j])  
 j += 1  
 result.extend(left[i:])  
 result.extend(right[j:])  
 return result  
 def **SapXepMerge**(self):  
 self.ds = self.merge\_sort(self.ds)

**Selection Sort , Insertion Sort , Bubble Sort**

def **binary\_search**(self, val, start, end):  
 if start == end:  
 return start  
 mid = (start + end) // 2  
 if self.ds[mid] < val:  
 return self.binary\_search(val, mid + 1, end)  
 else:  
 return self.binary\_search(val, start, mid)  
def **InsertionSort2**(self):  
 for i in range(1, len(self.ds)):  
 val = self.ds[i]  
 pos = self.binary\_search(val, 0, i)  
 j = i  
 while j > pos:  
 self.ds[j] = self.ds[j - 1]  
 j -= 1  
 self.ds[pos] = val  
  
def **BubbleSort**(self):  
 n = len(self.ds)  
 for i in range(n):  
 for j in range(n - 1, i, -1):  
 if self.ds[j] < self.ds[j - 1]:  
 self.ds[j], self.ds[j - 1] = self.ds[j - 1], self.ds[j]  
  
def **BubbleSort1**(self):  
 n = len(self.ds)  
 for i in range(n):  
 swapped = False  
 for j in range(n - 1, i, -1):  
 if self.ds[j] < self.ds[j - 1]:  
 self.ds[j], self.ds[j - 1] = self.ds[j - 1], self.ds[j]  
 swapped = True  
 if not swapped:  
 break  
  
def **BubbleSort2**(self):  
 n = len(self.ds)  
 swapped = True  
 start = 0  
 end = n - 1  
 while swapped == True:  
 swapped = False  
 for i in range(start, end):  
 if self.ds[i] > self.ds[i + 1]:  
 self.ds[i], self.ds[i + 1] = self.ds[i + 1], self.ds[i]  
 swapped = True  
 if swapped == False:  
 break  
  
 swapped = False  
 end -= 1  
 for i in range(end - 1, start - 1, -1):  
 if self.ds[i] > self.ds[i + 1]:  
 self.ds[i], self.ds[i + 1] = self.ds[i + 1], self.ds[i]  
 swapped = True  
 start += 1

**Heap sort:**

def **create\_heap**(self, n):  
 for t in range(n // 2, -1, -1):  
 i = t  
 j = 2 \* i  
 while j <= n:  
 if j < n and self.ds[j] < self.ds[j + 1]:  
 j = j + 1  
 if self.ds[i] < self.ds[j]:  
 self.ds[i], self.ds[j] = self.ds[j], self.ds[i]  
 i = j  
 j = 2 \* i  
 else:  
 break  
  
def **heap\_sort**(self):  
 p = len(self.ds) - 1  
 while p > 0:  
 self.create\_heap(p)  
 self.ds[0], self.ds[p] = self.ds[p], self.ds[0]  
 p=p-1

**Quick sort:**

def **PartitionSort**(self, first, last):  
 if first >= last:  
 return  
 x = self.ds[(first + last) // 2]  
 i = first  
 j = last  
 while i <= j:  
 while self.ds[i] < x:  
 i += 1  
 while self.ds[j] > x:  
 j -= 1  
 if i <= j:  
 self.ds[i], self.ds[j] = self.ds[j], self.ds[i]  
 i += 1  
 j -= 1  
 self.PartitionSort(first, j)  
 self.PartitionSort(i, last)  
  
def **QuickSort**(self):  
 self.PartitionSort(0, len(self.ds) - 1)

**Radix sort:**

def **counting\_sort**(self, exp1):  
 n = len(self.ds)  
 output = [0] \* n  
 count = [0] \* 10  
 for i in range(0, n):  
 index = (self.ds[i] // exp1)  
 count[(index % 10)] += 1  
 for i in range(1, 10):  
 count[i] += count[i - 1]  
 i = n - 1  
 while i >= 0: # Giảm dần while i < n  
 index = (self.ds[i] // exp1)  
 output[count[(index % 10)] - 1] = self.ds[i]  
 # Giảm dần: output[n - count[(index % 10)]] = self.ds[i]  
 count[(index % 10)] -= 1  
 i -= 1  
 i = 0  
 for i in range(0, len(self.ds)):  
 self.ds[i] = output[i]  
def **RadixSort**(self):  
 max1 = max(self.ds)  
 exp = 1  
 while max1 // exp > 0:  
 self.counting\_sort(exp)  
 exp \*= 10

**Bin sort:**

def **Bin\_Sort\_DG**(self):  
 B = [None] \* len(self.ds)  
 for i in range(len(self.ds)):  
 B[self.ds[i] - 1] = self.ds[i]  
 self.ds = B  
 return self.ds  
  
  
def **Bin\_Sort\_TQ**(self):  
 m = max(self.ds)  
 B = [[] for \_ in range(m + 1)] # Tạo m bins  
  
 # Phân loại các phần tử vào các bin tương ứng  
 for i in self.ds:  
 B[i].append(i) # Giảm dần : B[i].insert(0,num) chèn vào đầu thay vì cuối  
  
 # Nối tất cả các bin lại với nhau  
 self.ds = []  
 for bin in B: # Giảm dần: for bin in reversed(B)  
 self.ds += bin  
  
 return self.ds

def **Bin\_Sort\_TQ1**(self):  
 n = len(self.ds)  
 bins = [[] for \_ in range(n)]  
  
 # Kỳ 1: Phân phối các phần tử vào các bin theo key % n  
 for i in range(n):  
 index = self.ds[i] % n  
 bins[index].append(self.ds[i])  
  
 # Kỳ 2: Phân phối các phần tử trong mỗi bin vào các bin mới theo key / n  
 new\_bins = [[] for \_ in range(n)]  
 for bin in bins:  
 for num in bin:  
 index = num // n  
 new\_bins[index].append(num)  
  
 # Concatenate các bin lại với nhau để tạo ra danh sách được sắp xếp  
 self.ds = [num for bin in new\_bins for num in bin]  
  
 return self.ds

**Tìm kiếm tuyến tính**

def **LinearSearch\_Better**(self, X):  
 N = len(self.ds)  
 self.ds.append(X)  
 k = 0  
 while self.ds[k] != X:  
 k += 1  
 self.ds.pop()  
 if k < N:  
 return k  
 else:  
 return -1  
  
  
def **LinearSearch\_DQ**(self, X, index=0):  
 if index == len(self.ds):  
 return -1  
 elif self.ds[index] == X:  
 return index  
 else:  
 return self.LinearSearch\_DQ(X, index + 1)  
  
def **RecBinarySearch**(self, X, first, last):  
 if first > last:  
 return -1  
 mid = (first + last) // 2  
 if X == self.ds[mid]:  
 return mid  
 elif X < self.ds[mid]:  
 return self.RecBinarySearch(X, first, mid - 1)  
 else:  
 return self.RecBinarySearch(X, mid + 1, last)  
  
  
def **BinarySearch\_DQ**(self, X):  
 return self.RecBinarySearch(X, 0, len(self.ds) - 1)  
  
  
def **BinarySearch**(self, X):  
 first = 0  
 last = len(self.ds) - 1  
 while first <= last:  
 mid = (first + last) // 2  
 if X == self.ds[mid]:  
 return mid  
 elif X < self.ds[mid]:  
 last = mid - 1  
 else:  
 first = mid + 1  
 return -1

**BigMult\_ChiaDeTri:**

def **Big\_int\_mult**(X, Y):  
 if X < 10 or Y < 10:  
 return X \* Y  
 else:  
 n = max(len(str(X)), len(str(Y)))  
 m = n // 2  
  
 a = X // 10\*\*(m)  
 b = X % 10\*\*(m)  
 c = Y // 10\*\*(m)  
 d = Y % 10\*\*(m)  
  
 ac = Big\_int\_mult(a, c)  
 bd = Big\_int\_mult(b, d)  
 ad\_plus\_bc = Big\_int\_mult(a+b, c+d) - ac - bd  
  
 return ac \* 10\*\*(2\*m) + (ad\_plus\_bc \* 10\*\*m) + bd  
X = 123456789012345678901234567890  
Y = 1234567890  
result = Big\_int\_mult(X,Y)  
print("Kết quả là: ",result)  
print(X\*Y)

**ATM:**

def **Chon**(X, N):  
 X = sorted(X, reverse=True)  
 count = 0  
 total = 0  
 phuongan = []  
 for i in X:  
 while total <= N:  
 count += 1  
 total += i  
 if total > N:  
 total -= i  
 count -= 1  
 break  
 phuongan.append((i, count))  
 count = 0  
 return phuongan  
  
N = 3450000  
X = [500000, 200000, 100000, 50000]  
results = Chon(X, N)  
for menh\_gia, so\_to in results:  
 print(f"Mệnh giá {menh\_gia}: {so\_to} tờ")  
  
def **Chon**(X, N):  
 X = sorted(X, reverse=True)  
 count = 0  
 total = 0  
 so\_to = 1  
 phuongan = []  
 for i in X:  
 while so\_to > 0:  
 so\_to -= 1  
 count += 1  
 total += i  
 if total > N:  
 total -= i  
 count -= 1  
 phuongan.append((i, count))  
 count = 0  
 so\_to = 1  
 return phuongan  
  
X = [500000, 200000, 100000, 50000]  
  
N = 3450000  
results = Chon(X, N)  
  
for menh\_gia, so\_to in results:  
 print(f"Mệnh giá {menh\_gia}: {so\_to} tờ")  
S = 5  
  
def **Chon**(X, N):  
 X = sorted(X, reverse=True)  
 count = 0  
 total = 0  
 so\_to = S  
 phuongan = []  
 for i in X:  
 while so\_to > 0:  
 so\_to -= 1  
 count += 1  
 total += i  
 if total > N:  
 total -= i  
 count -= 1  
 break  
 phuongan.append((i, count))  
 count = 0  
 so\_to = S  
 return phuongan  
  
X = [500000, 200000, 100000, 50000]  
N = 3450000  
results = Chon(X, N)  
for menh\_gia, so\_to in results:  
 print(f"Mệnh giá {menh\_gia}: {so\_to} tờ")

**TSP:**

class TSP:  
 def \_\_init\_\_(self, matrix):  
 self.matrix = matrix  
 self.n = len(matrix)  
  
 def **greedy\_tsp**(self):  
 n = len(self.matrix)  
 path = [0] # starting point  
 unvisited = list(range(1, n))  
 city\_names = ['A', 'B', 'C', 'D', 'E', 'F']  
 while unvisited:  
 last\_city = path[-1]  
 next\_city = min(unvisited, key=lambda city: self.matrix[last\_city][city])  
 path.append(next\_city)  
 unvisited.remove(next\_city)  
 total\_length = sum(self.matrix[path[i - 1]][path[i]] for i in range(1, len(path)))  
 total\_length += self.matrix[path[0]][path[-1]]  
  
 path = [city\_names[i] for i in path]  
 path\_string = "-".join(path) + "-" + path[0]  
 return path\_string, total\_length  
matrix = [  
 [0, 5, 7.07, 16.55, 15.52, 18], # Khoảng cách từ A đến A, B, C, D, E, F  
 [5, 0, 5, 11.7, 11.05, 14.32], # Khoảng cách từ B đến A, B, C, D, E, F  
 [7.07, 5, 0, 14, 14.32, 18.38], # Khoảng cách từ C đến A, B, C, D, E, F  
 [16.55, 11.7, 14, 0, 3, 7.62], # Khoảng cách từ D đến A, B, C, D, E, F  
 [15.52, 11.05, 14.32, 3, 0, 5], # Khoảng cách từ E đến A, B, C, D, E, F  
 [18, 14.32, 18.38, 7.62, 5, 0], # Khoảng cách từ F đến A, B, C, D, E, F  
]  
tsp = TSP(matrix)  
path, length = tsp.greedy\_tsp()  
print(f"Chu trình: {path}")  
print(f"Tổng độ dài: {length}")

**Cái ba lô:**

class Do\_vat:  
 def \_\_init\_\_(self, ten, w, gt):  
 self.ten = ten  
 self.trong\_luong = w  
 self.gia\_tri = gt  
 self.don\_gia = gt / w  
 self.phuong\_an = 0  
  
def **greedy**(dsdv, w):  
 dsdv = list(dsdv)  
 dsdv.sort(key=lambda x: x.don\_gia, reverse=True)  
 for i in range(len(dsdv)):  
 while w >= dsdv[i].trong\_luong:  
 dsdv[i].phuong\_an = w // dsdv[i].trong\_luong  
 w -= dsdv[i].phuong\_an \* dsdv[i].trong\_luong  
 # Tính tổng trọng lượng và tổng giá trị  
 tong\_trong\_luong = sum([dv.trong\_luong \* dv.phuong\_an for dv in dsdv])  
 tong\_gia\_tri = sum([dv.gia\_tri \* dv.phuong\_an for dv in dsdv])  
 return dsdv, tong\_trong\_luong, tong\_gia\_tri  
  
def **in\_ket\_qua**(dsdv, tong\_trong\_luong, tong\_gia\_tri):  
 print("Danh sách đồ vật được chọn:")  
 for dv in dsdv:  
 if dv.phuong\_an > 0:  
 print("Tên: ", dv.ten, " - Trọng lượng: ", dv.trong\_luong, " - Giá trị: ", dv.gia\_tri, " - Đơn giá:",  
 dv.don\_gia, "- Phương án:", dv.phuong\_an)  
 print("\nTổng trọng lượng: ", tong\_trong\_luong)  
 print("Tổng giá trị: ", tong\_gia\_tri)  
dsdv = [Do\_vat("A", 15, 30), Do\_vat("B", 10, 25), Do\_vat("C", 2, 2), Do\_vat("D", 4, 6)]  
w = 37  
dsdv, tong\_trong\_luong, tong\_gia\_tri = greedy(dsdv, w)  
in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri)  
  
class **Do\_vat**:  
 def \_\_init\_\_(self, ten, w, gt, S):  
 self.ten = ten  
 self.trong\_luong = w  
 self.gia\_tri = gt  
 self.don\_gia = gt / w  
 self.phuong\_an = 0  
 self.so\_luong = S  
  
def **greedy**(dsdv, w):  
 dsdv = sorted(dsdv, key=lambda x: x.don\_gia, reverse=True)  
 for dv in dsdv:  
 while w >= dv.trong\_luong and dv.so\_luong > 0:  
 dv.phuong\_an += 1  
 dv.so\_luong -= 1  
 w -= dv.trong\_luong  
  
 # Tính tổng trọng lượng và tổng giá trị  
 tong\_trong\_luong = sum([dv.trong\_luong \* dv.phuong\_an for dv in dsdv])  
 tong\_gia\_tri = sum([dv.gia\_tri \* dv.phuong\_an for dv in dsdv])  
 return dsdv, tong\_trong\_luong, tong\_gia\_tri  
  
dsdv = [Do\_vat("A", 15, 30, 2), Do\_vat("B", 10, 25, 3), Do\_vat("C", 2, 2, 4), Do\_vat("D", 4, 6, 5)]  
  
def **in\_ket\_qua**(dsdv, tong\_trong\_luong, tong\_gia\_tri):  
 print("Danh sách đồ vật được chọn:")  
 for dv in dsdv:  
 if dv.phuong\_an > 0:  
 print("Tên: ", dv.ten, " - Trọng lượng: ", dv.trong\_luong, " - Giá trị: ", dv.gia\_tri,"- Số lượng còn lại:", dv.so\_luong, " - Đơn giá:", dv.don\_gia, "-Phương án:", dv.phuong\_an)  
  
 print("\nTổng trọng lượng: ", tong\_trong\_luong)  
 print("Tổng giá trị: ", tong\_gia\_tri)  
  
w = 37  
dsdv, tong\_trong\_luong, tong\_gia\_tri = greedy(dsdv, w)  
in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri)  
  
  
class **Do\_vat**:  
 def \_\_init\_\_(self, ten, w, gt):  
 self.ten = ten  
 self.trong\_luong = w  
 self.gia\_tri = gt  
 self.don\_gia = gt / w  
 self.phuong\_an = 0  
  
  
def **greedy**(dsdv, w):  
 dsdv = list(dsdv)  
 dsdv.sort(key=lambda x: x.don\_gia, reverse=True)  
 for i in range(len(dsdv)):  
 dsdv[i].phuong\_an = min((w // dsdv[i].trong\_luong), 1)  
 w -= dsdv[i].phuong\_an \* dsdv[i].trong\_luong  
 # Tính tổng trọng lượng và tổng giá trị  
 tong\_trong\_luong = sum([dv.trong\_luong \* dv.phuong\_an for dv in dsdv])  
 tong\_gia\_tri = sum([dv.gia\_tri \* dv.phuong\_an for dv in dsdv])  
  
 return dsdv, tong\_trong\_luong, tong\_gia\_tri  
  
  
dsdv = [Do\_vat("A", 15, 30), Do\_vat("B", 10, 25), Do\_vat("C", 2, 2), Do\_vat("D", 4, 6)]  
  
  
def in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri):  
 print("Danh sách đồ vật được chọn:")  
 for dv in dsdv:  
 if dv.phuong\_an > 0:  
 print("Tên: ", dv.ten, " - Trọng lượng: ", dv.trong\_luong, " - Giá trị: ", dv.gia\_tri, " - Đơn giá:",  
 dv.don\_gia, "-Phương án:", dv.phuong\_an)  
  
 print("\nTổng trọng lượng: ", tong\_trong\_luong)  
 print("Tổng giá trị: ", tong\_gia\_tri)  
  
  
w = 37  
dsdv, tong\_trong\_luong, tong\_gia\_tri = greedy(dsdv, w)  
in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri)

**Tổ hợp Bảng:**

def **Comb**(n, k):  
 C = [[0 for \_ in range(n + 1)] for \_ in range(n + 1)]  
 C[0][0] = 1  
 for i in range(1, n + 1):  
 C[i][0] = 1  
 C[i][i] = 1  
 for j in range(1, i):  
 C[i][j] = C[i - 1][j - 1] + C[i - 1][j]  
 return C[n][k]  
  
  
n = 10  
k = 3  
print(Comb(n, k))

**Vector**

def **Comb**(n, k):  
 V = [0 for \_ in range(n + 1)]  
 V[0] = 1  
 V[1] = 1  
 for i in range(2, n + 1):  
 p1 = V[0]  
 for j in range(1, i):  
 p2 = V[j]  
 V[j] = p1 + p2  
 p1 = p2  
 V[i] = 1  
 return V[k]  
  
n = 10  
k = 3  
print(Comb(n, k))

**Balo Quy Hoạch Động:**

class DoVat:  
 def \_\_init\_\_(self, ten, tl, gt, so\_dv\_duoc\_chon=0):  
 self.ten = ten  
 self.gt = gt  
 self.tl = tl  
 self.so\_dv\_duoc\_chon = so\_dv\_duoc\_chon  
  
  
def **tao\_bang**(dsdv, n, W):  
 F = [[0 for \_ in range(W + 1)] for \_ in range(n)]  
 X = [[0 for \_ in range(W + 1)] for \_ in range(n)]  
  
 for V in range(dsdv[0].tl, W + 1):  
 X[0][V] = V // dsdv[0].tl  
 F[0][V] = X[0][V] \* dsdv[0].gt  
  
 for k in range(1, n):  
 for V in range(W + 1):  
 FMax = F[k - 1][V]  
 XMax = 0  
 yk = V // dsdv[k].tl  
 # Ba lo 2: yk = min(1,V // dsdv[k].tl)  
 # Ba lo 3: yk = min(V // dsdv[k].tl,dsdv[k].sl)  
 for xk in range(1, yk + 1):  
 if F[k - 1][V - xk \* dsdv[k].tl] + xk \* dsdv[k].gt > FMax:  
 FMax = F[k - 1][V - xk \* dsdv[k].tl] + xk \* dsdv[k].gt  
 XMax = xk  
 F[k][V] = FMax  
 X[k][V] = XMax  
  
 return F, X  
  
  
def **tra\_bang**(dsdv, n, W, X):  
 k = n - 1  
 V = W  
 while k >= 0:  
 dsdv[k].so\_dv\_duoc\_chon = X[k][V]  
 V -= X[k][V] \* dsdv[k].tl  
 k -= 1  
  
  
def **in\_bang**(n, W, F, X):  
 print("Bảng F và X:")  
 for k in range(n):  
 for V in range(W + 1):  
 print(f"{F[k][V]:.1f} {X[k][V]:2d}", end="||")  
 print()  
  
  
def **in\_kq**(dsdv, W):  
 print("Phương án thu được từ kỹ thuật QUY HOẠCH ĐỘNG như sau:")  
 tong\_tl = tong\_gt = 0  
 for dv in dsdv:  
 if dv.so\_dv\_duoc\_chon > 0:  
 tong\_tl += dv.so\_dv\_duoc\_chon \* dv.tl  
 tong\_gt += dv.so\_dv\_duoc\_chon \* dv.gt  
 print(f"Tổng trọng lượng của ba lô: {tong\_tl}")  
 print(f"Tổng giá trị của ba lô: {tong\_gt}")  
  
  
# Khởi tạo danh sách đồ vật  
dsdv = [  
 DoVat("1", 3, 4),  
 DoVat("2", 5, 5),  
 DoVat("3", 5, 6),  
 DoVat("4", 2, 3),  
 DoVat("5", 1, 1)  
]  
  
W = 9 # Khối lượng tối đa của ba lô  
n = len(dsdv) # Số lượng đồ vật  
  
F, X = tao\_bang(dsdv, n, W)  
tra\_bang(dsdv, n, W, X)  
in\_bang(n, W, F, X)  
in\_kq(dsdv, W)

**Định trị biểu thức số học:**

class Node:  
 def \_\_init\_\_(self, val, left=None, right=None):  
 self.val = val  
 self.right = right  
 self.left = left  
  
  
def **Dinh\_Tri**(root):  
 if root is None:  
 return 0  
  
 if root.left is None and root.right is None:  
 return float(root.val)  
  
 if root.val == '+':  
 return Dinh\_Tri(root.left) + Dinh\_Tri(root.right)  
 elif root.val == '-':  
 return Dinh\_Tri(root.left) - Dinh\_Tri(root.right)  
 elif root.val == '\*':  
 return Dinh\_Tri(root.left) \* Dinh\_Tri(root.right)  
 elif root.val == '/':  
 try:  
 return Dinh\_Tri(root.left) / Dinh\_Tri(root.right)  
 except ZeroDivisionError:  
 print("Không thể chia cho 0!")  
 return 0  
 else:  
 return 0  
  
  
def **inorder**(root):  
 if root is not None:  
 inorder(root.left)  
 print(root.val, end=' ')  
 inorder(root.right)  
  
  
def **read\_tree**():  
 x = input("Nhập giá trị cho nút (nhập '0' để bỏ qua): ")  
  
 if x == '0':  
 return None  
 else:  
 node = Node(x)  
 print(f"Nhập con trái của {x}: ")  
 node.left = read\_tree()  
 print(f"Nhập con phải của {x}: ")  
 node.right = read\_tree()  
 return node  
  
  
node = read\_tree()  
print("Cây biểu thức sau khi duyệt LNR là:")  
inorder(node)  
print("\nTrị của biểu thức là: ", Dinh\_Tri(node))

**Mine của tui :3**

class Danhsach:  
 def \_\_init\_\_(self,list=[]):  
 self.PhanTu=list  
 def Nhap(self):  
 a=int(input())  
 while a != 0:  
 self.PhanTu.append(a)  
 a = int(input())  
 return  
 def Xuat(self):  
 for i in self.PhanTu:  
 print(i,end=' ')  
 print()  
 def Tim(self,X):  
 for i in range(len(self.PhanTu)):  
 if self.PhanTu[i]==X:  
 return i  
 return -1  
 def Xoa(self,X):  
 for i in range(len(self.PhanTu)):  
 if self.PhanTu[i] == X:  
 self.PhanTu= self.PhanTu[:i]+self.PhanTu[i+1:]  
 return  
 def Them(self,X,i):  
 i= 0 if i>len(self.PhanTu) else len(self.PhanTu)  
 for j in range(len(self.PhanTu)):  
 if j==i:  
 self.PhanTu=self.PhanTu[:i]+[X]+self.PhanTu[i:]  
 return  
 def Xua(self,X,Y):  
 for i in range(len(self.PhanTu)):  
 if self.PhanTu[i]==X:  
 self.PhanTu[i]=Y  
 return  
 @staticmethod  
 def **merge**(left,right):  
 result = []  
 i = j = 0  
 while i < len(left) and j < len(right):  
 if left[i] < right[j]:  
 result.append(left[i])  
 i += 1  
 else:  
 result.append(right[j])  
 j += 1  
 result=result+left[i:]  
 result=result+right[j:]  
 return result  
 @staticmethod  
 def **MergeSort**(Arr):  
 n=len(Arr)  
 if n>1:  
 return Danhsach.merge(Danhsach.MergeSort(Arr[:n//2]),Danhsach.MergeSort(Arr[n//2:]))  
 return Arr  
 def **SapXep\_MergeSort**(self):  
 self.PhanTu=self.MergeSort(self.PhanTu)  
 def **SapXep\_Selection**(self):  
 for i in range(len(self.PhanTu)-1):  
 for j in range(i,len(self.PhanTu)):  
 if self.PhanTu[i]>self.PhanTu[j]:  
 self.PhanTu[i],self.PhanTu[j]=self.PhanTu[j],self.PhanTu[i]  
 def **SapXep\_Insertion**(self):  
 for i in range(1, len(self.PhanTu)):  
 key = self.PhanTu[i]  
 j = i - 1  
 while j >= 0 and key < self.PhanTu[j]:  
 self.PhanTu[j + 1] = self.PhanTu[j]  
 j -= 1  
 self.PhanTu[j + 1] = key  
  
 def **SapXep\_Bubble**(self):  
 n = len(self.PhanTu)  
 for i in range(n - 1):  
 for j in range(0, n - i - 1):  
 if self.PhanTu[j] > self.PhanTu[j + 1]:  
 self.PhanTu[j], self.PhanTu[j + 1] = self.PhanTu[j + 1], self.PhanTu[j]  
 def **SapXep\_BubbleCT**(self):  
 n = len(self.PhanTu)  
 for i in range(n - 1):  
 min\_index = i  
 for j in range(i, n - i - 1):  
 if self.PhanTu[j] > self.PhanTu[j + 1]:  
 self.PhanTu[j], self.PhanTu[j + 1] = self.PhanTu[j + 1], self.PhanTu[j]  
 if self.PhanTu[j] < self.PhanTu[min\_index]:  
 min\_index = j  
 self.PhanTu[min\_index],self.PhanTu[i]=self.PhanTu[i],self.PhanTu[min\_index]  
 @staticmethod  
 def **heapify**(arr, n, i):  
 largest = i  
 l = 2 \* i + 1  
 r = 2 \* i + 2  
 if l < n and arr[i] < arr[l]:  
 largest = l  
 if r < n and arr[largest] < arr[r]:  
 largest = r  
 if largest != i:  
 arr[i], arr[largest] = arr[largest], arr[i]  
 Danhsach.heapify(arr, n, largest)  
  
 def **heapSort**(self):  
 n = len(self.PhanTu)  
  
 for i in range(n // 2, -1, -1):  
 Danhsach.heapify(self.PhanTu, n, i)  
  
 for i in range(n - 1, 0, -1):  
 self.PhanTu[i], self.PhanTu[0] = self.PhanTu[0], self.PhanTu[i]  
  
 Danhsach.heapify(self.PhanTu, i, 0)  
 @staticmethod  
 def **partition**(array, start, end):  
 pivot = array[start]  
 low = start + 1  
 high = end  
 while True:  
 while low <= high and array[high] >= pivot:  
 high = high - 1  
 while low <= high and array[low] <= pivot:  
 low = low + 1  
 if low <= high:  
 array[low], array[high] = array[high], array[low]  
 else:  
 break  
 array[start], array[high] = array[high], array[start]  
 return high  
 def **quick\_sort**(self, start, end):  
 if start >= end:  
 return  
 p = self.partition(self.PhanTu, start, end)  
 self.quick\_sort(start, p - 1)  
 self.quick\_sort(p + 1, end)  
 @staticmethod  
 def **countingSort**(arr, exp1):  
 n = len(arr)  
 output = [0] \* (n)  
 count = [0] \* (10)  
 for i in range(0, n):  
 index = (arr[i] / exp1)  
 count[int(index % 10)] += 1  
 for i in range(1, 10):  
 count[i] += count[i - 1]  
 i = n - 1  
 while i >= 0:  
 index = (arr[i] / exp1)  
 output[count[int(index % 10)] - 1] = arr[i]  
 count[int(index % 10)] -= 1  
 i -= 1  
 i = 0  
 for i in range(0, len(arr)):  
 arr[i] = output[i]  
 def **radixSort**(self):  
 max1 = max(self.PhanTu)  
 exp = 1  
 while max1 / exp > 0:  
 self.countingSort(self.PhanTu, exp)  
 exp \*= 10  
a=Danhsach([7,8,6,5,4,3,9,2,1])  
a.Xuat()  
a.radixSort()  
a.Xuat()

**THANG\_TAM\_HUYHOANG**

**1/ Hàm nhân**

import math

def f(b, n):

return b\*\*n

def d(n):

return n # Hàm d(n) = n

def ktraHamNhan(d, m, n):

return d(m\*n) == d(m) \* d(n)

def T(a, b, n):

if ktraHamNhan(d, a, b):

if a > f(b, n):

n = int(math.log(a, b))

print(f"T(n) = O(n^{n})")

elif a < f(b, n):

n = int(math.log(f(b, n), b))

print(f"T(n) = O(n^{n})")

else:

n = int(math.log(a, b))

print(f"T(n) = O((n^{n})log{b}n)")

else:

print("d(n) không phải là hàm nhân. Không thể giải phương trình.")

a = int(input("Nhâp a = "))

b = int(input("Nhập b = "))

somu\_n = int(input("Nhập số mũ của n = "))

T(a, b, somu\_n)

**2/ Xây dụng lớp DanhSach**

class Danhsach:

def \_\_init\_\_(self):

self.ds = []

def nhap(self):

n = int(input("Nhập số lượng phần tử: "))

for i in range(n):

self.ds.append(int(input(f"Nhập phần tử thứ {i+1}: ")))

def xuat(self):

for i in self.ds:

print(i, end=' ')

print()

def tim(self, x):

if x in self.ds:

return self.ds.index(x)

else:

return -1

def them(self, x):

self.ds.append(x)

def xoa(self, x):

if x in self.ds:

self.ds.remove(x)

def sua(self, x, y):

for i in range(len(self.ds)):

if self.ds[i] == x:

self.ds[i] = y

**MergeSort:**

class SapXep(Danhsach):

def \_\_init\_\_(self):

super().\_\_init\_\_()

@staticmethod

def merge\_sort(arr):

n = len(arr)

if n <= 1:

return arr

mid = n // 2

left = arr[:mid]

right = arr[mid:]

return SapXep().merge(SapXep.merge\_sort(left),SapXep.merge\_sort(right))

def merge(self,left,right):

result = []

i = j = 0

while i < len(left) and j < len(right):

if left[i] < right[j]:

result.append(left[i])

i += 1

else:

result.append(right[j])

j += 1

result.extend(left[i:])

result.extend(right[j:])

return result

def SapXepMerge(self):

self.ds = self.merge\_sort(self.ds)

**Selection Sort , Insertion Sort , Bubble Sort**

class SapXepN2(Danhsach):

def \_\_init\_\_(self):

super().\_\_init\_\_()

def SelectionSort(self):

n = len(self.ds)

for i in range(n):

min\_temp = i

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

if self.ds[j] < self.ds[min\_temp]:

min\_temp = j

if min\_temp != i:

self.ds[i],self.ds[min\_temp] = self.ds[min\_temp],self.ds[i]

def InsertionSort(self):

n = len(self.ds)

k = 1

while k < n and self.ds[k-1] <= self.ds[k]:

k += 1

while k < n:

x = self.ds[k]

pos = k - 1

while pos >= 0 and x < self.ds[pos]:

self.ds[pos + 1] = self.ds[pos]

pos -= 1

self.ds[pos + 1] = x

k += 1

def binary\_search(self,val,start,end):

if start == end:

return start

mid = (start + end) // 2

if self.ds[mid] < val:

return self.binary\_search(val,mid + 1,end)

else:

return self.binary\_search(val,start,mid)

def InsertionSort2(self):

for i in range(1, len(self.ds)):

val = self.ds[i]

pos = self.binary\_search(val, 0, i)

j = i

while j > pos:

self.ds[j] = self.ds[j - 1]

j -= 1

self.ds[pos] = val

def BubbleSort(self):

n = len(self.ds)

for i in range(n):

for j in range(n-1,i,-1):

if self.ds[j] < self.ds[j-1]:

self.ds[j],self.ds[j-1] = self.ds[j-1],self.ds[j]

def BubbleSort1(self):

n = len(self.ds)

for i in range(n):

swapped = False

for j in range(n-1,i,-1):

if self.ds[j] < self.ds[j-1]:

self.ds[j],self.ds[j-1] = self.ds[j-1],self.ds[j]

swapped = True

if not swapped:

break

def BubbleSort2(self):

n = len(self.ds)

swapped = True

start = 0

end = n - 1

while swapped == True:

swapped = False

for i in range(start,end):

if self.ds [i] > self.ds[i+1]:

self.ds[i],self.ds[i+1] = self.ds[i+1],self.ds[i]

swapped = True

if swapped == False:

break

swapped = False

end -= 1

for i in range(end-1,start-1,-1):

if self.ds [i] > self.ds[i+1]:

self.ds[i],self.ds[i+1] = self.ds[i+1],self.ds[i]

swapped = True

start += 1

**Heap sort:**

def create\_heap(self, n):

for t in range(n//2, -1, -1):

i = t

j = 2 \* i

while j <= n:

if j < n and self.ds[j] < self.ds[j+1]:

j = j + 1

if self.ds[i] < self.ds[j]:

self.ds[i], self.ds[j] = self.ds[j], self.ds[i]

i = j

j = 2 \* i

else:

break

def heap\_sort(self):

p = len(self.ds) - 1

while p > 0:

self.create\_heap(p)

self.ds[0], self.ds[p] = self.ds[p], self.ds[0]

p = p – 1

**Quick sort:**

def PartitionSort(self,first,last):

if first >= last:

return

x = self.ds[(first + last)//2]

i = first

j = last

while i <= j:

while self.ds[i] < x:

i += 1

while self.ds[j] > x:

j -= 1

if i <= j:

self.ds[i],self.ds[j] = self.ds[j],self.ds[i]

i += 1

j -= 1

self.PartitionSort(first,j)

self.PartitionSort(i,last)

def QuickSort(self):

self.PartitionSort(0,len(self.ds)-1)

**Radix sort:**

def counting\_sort(self, exp1):

n = len(self.ds)

output = [0] \* n

count = [0] \* 10

for i in range(0, n):

index = (self.ds[i] // exp1)

count[(index % 10)] += 1

for i in range(1, 10):

count[i] += count[i - 1]

i = n - 1

while i >= 0: # Giảm dần while i < n

index = (self.ds[i] // exp1)

output[count[(index % 10)] - 1] = self.ds[i] # Giảm dần: output[n - count[(index % 10)]] = self.ds[i]

count[(index % 10)] -= 1

i -= 1

i = 0

for i in range(0, len(self.ds)):

self.ds[i] = output[i]

def RadixSort(self):

max1 = max(self.ds)

exp = 1

while max1 // exp > 0:

self.counting\_sort(exp)

exp \*= 10

**Bin sort:**

def Bin\_Sort\_DG(self):

B = [None] \* len(self.ds)

for i in range(len(self.ds)):

B[self.ds[i] - 1] = self.ds[i]

self.ds = B

return self.ds

def Bin\_Sort\_TQ(self):

m = max(self.ds)

B = [[] for \_ in range(m+1)] # Tạo m bins

# Phân loại các phần tử vào các bin tương ứng

for i in self.ds:

B[i].append(i) #Giảm dần : B[i].insert(0,num) chèn vào đầu thay vì cuối

# Nối tất cả các bin lại với nhau

self.ds = []

for bin in B: # Giảm dần: for bin in reversed(B)

self.ds += bin

return self.ds

def Bin\_Sort\_TQ1(self):

n = len(self.ds)

bins = [[] for \_ in range(n)]

# Kỳ 1: Phân phối các phần tử vào các bin theo key % n

for i in range(n):

index = self.ds[i] % n

bins[index].append(self.ds[i])

# Kỳ 2: Phân phối các phần tử trong mỗi bin vào các bin mới theo key / n

new\_bins = [[] for \_ in range(n)]

for bin in bins:

for num in bin:

index = num // n

new\_bins[index].append(num)

# Concatenate các bin lại với nhau để tạo ra danh sách được sắp xếp

self.ds = [num for bin in new\_bins for num in bin]

return self.ds

**Tìm kiếm tuyến tính**

class Timkiem(Danhsach):

def \_\_init\_\_(self):

super().\_\_init\_\_()

def LinearSearch\_Better(self, X):

N = len(self.ds)

self.ds.append(X)

k = 0

while self.ds[k] != X:

k += 1

self.ds.pop()

if k < N:

return k

else:

return -1

def LinearSearch\_DQ(self, X, index=0):

if index == len(self.ds):

return -1

elif self.ds[index] == X:

return index

else:

return self.LinearSearch\_DQ(X, index + 1)

**Binary search:**

class TimKiem(Danhsach):

def \_\_init\_\_(self):

super().\_\_init\_\_()

def RecBinarySearch(self, X, first, last):

if first > last:

return -1

mid = (first + last) // 2

if X == self.ds[mid]:

return mid

elif X < self.ds[mid]:

return self.RecBinarySearch(X, first, mid - 1)

else:

return self.RecBinarySearch(X, mid + 1, last)

def BinarySearch\_DQ(self, X):

return self.RecBinarySearch(X, 0, len(self.ds) - 1)

def BinarySearch(self, X):

first = 0

last = len(self.ds) - 1

while first <= last:

mid = (first + last) // 2

if X == self.ds[mid]:

return mid

elif X < self.ds[mid]:

last = mid - 1

else:

first = mid + 1

return -1

**BigMult\_ChiaDeTri:**

def Big\_int\_mult(X,Y,n):

s = Sign(X) \* Sign(Y)

x = abs(X)

y = abs(Y)

if n == 1:

return x \* y \* s

else:

A = Left(x,n//2)

B = Right(x,n//2)

C = Left(y,n//2)

D = Right(y,n//2)

m1 = Big\_int\_mult(A,C,n//2)

m2 = Big\_int\_mult(A-B,D-C,n//2)

m3 = Big\_int\_mult(B,D,n//2)

return (s \* (m1 \* (10\*\*n) + (m1 + m2 + m3)\*(10\*\*(n//2)) + m3))

def Sign(X):

if X > 0:

return 1

elif X < 0:

return -1

else:

return 0

def Left(X,n):

return X // (10\*\*n)

def Right(X,n):

return X % (10\*\*n)

X = 1234

Y = -1234

n = max(len(str(abs(X))),len(str(abs(Y))))

result = Big\_int\_mult(X,Y,n)

print("Kết quả là: ",result)

**ATM:**

**-----Không giới hạn số tờ-----**

def Chon(X,N):

X = sorted(X, reverse=True)

count = 0

total = 0

phuongan =[]

for i in X:

while total <= N:

count +=1

total += i

if total > N:

total -= i

count -= 1

break

phuongan.append((i,count))

count = 0

return phuongan

N = 3450000

X = [500000,200000,100000,50000]

results=Chon(X,N)

for menh\_gia, so\_to in results:

print(f"Mệnh giá {menh\_gia}: {so\_to} tờ")

**----Giới hạn số tờ là 1-----**

def Chon(X,N):

X = sorted(X, reverse=True)

count = 0

total = 0

so\_to = 1

phuongan = []

for i in X:

while so\_to > 0:

so\_to -= 1

count += 1

total += i

if total > N:

total -= i

count -= 1

phuongan.append((i,count))

count = 0

so\_to = 1

return phuongan

X = [500000,200000,100000,50000]

N = 3450000

results=Chon(X,N)

for menh\_gia, so\_to in results:

print(f"Mệnh giá {menh\_gia}: {so\_to} tờ")

**-----Số tờ tùy chọn -----**

S = 5

def Chon(X,N):

X = sorted(X, reverse=True)

count = 0

total = 0

so\_to = S

phuongan = []

for i in X:

while so\_to > 0:

so\_to -= 1

count += 1

total += i

if total > N:

total -= i

count -= 1

break

phuongan.append((i,count))

count = 0

so\_to = S

return phuongan

X = [500000,200000,100000,50000]

N = 3450000

results=Chon(X,N)

for menh\_gia, so\_to in results:

**TSP:**

class TSP:

def \_\_init\_\_(self,matrix):

self.matrix = matrix

self.n = len(matrix)

def greedy\_tsp(self):

n = len(self.matrix)

path = [0] # starting point

unvisited = list(range(1, n))

city\_names = ['A', 'B', 'C', 'D', 'E', 'F']

while unvisited:

last\_city = path[-1]

next\_city = min(unvisited, key=lambda city: self.matrix[last\_city][city])

path.append(next\_city)

unvisited.remove(next\_city)

total\_length = sum(self.matrix[path[i - 1]][path[i]] for i in range(1, len(path)))

total\_length += self.matrix[path[0]][path[-1]]

path = [city\_names[i] for i in path]

path\_string = "-".join(path) + "-" + path[0]

return path\_string, total\_length

matrix = [

[0, 5, 7.07, 16.55, 15.52, 18], # Khoảng cách từ A đến A, B, C, D, E, F

[5, 0, 5, 11.7, 11.05, 14.32], # Khoảng cách từ B đến A, B, C, D, E, F

[7.07, 5, 0, 14, 14.32, 18.38], # Khoảng cách từ C đến A, B, C, D, E, F

[16.55, 11.7, 14, 0, 3, 7.62], # Khoảng cách từ D đến A, B, C, D, E, F

[15.52, 11.05, 14.32, 3, 0, 5], # Khoảng cách từ E đến A, B, C, D, E, F

[18, 14.32, 18.38, 7.62, 5, 0], # Khoảng cách từ F đến A, B, C, D, E, F

]

tsp = TSP(matrix)

path, length = tsp.greedy\_tsp()

print(f"Chu trình: {path}")

print(f"Tổng độ dài: {length}")

**Cái ba lô:**

**----Không giới hạn-----**

class Do\_vat:

def \_\_init\_\_(self,ten,w,gt):

self.ten = ten

self.trong\_luong = w

self.gia\_tri = gt

self.don\_gia = gt / w

self.phuong\_an = 0

def greedy(dsdv, w):

dsdv = list(dsdv)

dsdv.sort(key = lambda x: x.don\_gia, reverse=True)

for i in range(len(dsdv)):

while w >= dsdv[i].trong\_luong:

dsdv[i].phuong\_an = w // dsdv[i].trong\_luong

w -= dsdv[i].phuong\_an \* dsdv[i].trong\_luong

# Tính tổng trọng lượng và tổng giá trị

tong\_trong\_luong = sum([dv.trong\_luong \* dv.phuong\_an for dv in dsdv])

tong\_gia\_tri = sum([dv.gia\_tri \* dv.phuong\_an for dv in dsdv])

return dsdv, tong\_trong\_luong, tong\_gia\_tri

def in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri):

print("Danh sách đồ vật được chọn:")

for dv in dsdv:

if dv.phuong\_an > 0:

print("Tên: ", dv.ten, " - Trọng lượng: ", dv.trong\_luong, " - Giá trị: ", dv.gia\_tri," - Đơn giá:",dv.don\_gia,"- Phương án:",dv.phuong\_an)

print("\nTổng trọng lượng: ", tong\_trong\_luong)

print("Tổng giá trị: ", tong\_gia\_tri)

dsdv = [Do\_vat("A",15,30),Do\_vat("B",10,25),Do\_vat("C",2,2),Do\_vat("D",4,6)]

w = 37

dsdv, tong\_trong\_luong, tong\_gia\_tri = greedy(dsdv, w)

in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri)

**-------Tùy chọn theo số lượng-----**

class Do\_vat:

def \_\_init\_\_(self,ten,w,gt,S):

self.ten = ten

self.trong\_luong = w

self.gia\_tri = gt

self.don\_gia = gt / w

self.phuong\_an = 0

self.so\_luong = S

def greedy(dsdv, w):

dsdv = sorted(dsdv, key = lambda x: x.don\_gia, reverse=True)

for dv in dsdv:

while w >= dv.trong\_luong and dv.so\_luong > 0:

dv.phuong\_an += 1

dv.so\_luong -= 1

w -= dv.trong\_luong

# Tính tổng trọng lượng và tổng giá trị

tong\_trong\_luong = sum([dv.trong\_luong \* dv.phuong\_an for dv in dsdv])

tong\_gia\_tri = sum([dv.gia\_tri \* dv.phuong\_an for dv in dsdv])

return dsdv, tong\_trong\_luong, tong\_gia\_tri

dsdv = [Do\_vat("A",15,30,2),Do\_vat("B",10,25,3),Do\_vat("C",2,2,4),Do\_vat("D",4,6,5)]

def in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri):

print("Danh sách đồ vật được chọn:")

for dv in dsdv:

if dv.phuong\_an > 0:

print("Tên: ", dv.ten, " - Trọng lượng: ", dv.trong\_luong, " - Giá trị: ", dv.gia\_tri,"- Số lượng còn lại:",dv.so\_luong," - Đơn giá:",dv.don\_gia,"-Phương án:",dv.phuong\_an)

print("\nTổng trọng lượng: ", tong\_trong\_luong)

print("Tổng giá trị: ", tong\_gia\_tri)

w = 37

dsdv, tong\_trong\_luong, tong\_gia\_tri = greedy(dsdv, w)

in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri)

**-----Mỗi đồ vật chỉ chọn 1---**

class Do\_vat:

def \_\_init\_\_(self,ten,w,gt):

self.ten = ten

self.trong\_luong = w

self.gia\_tri = gt

self.don\_gia = gt / w

self.phuong\_an = 0

def greedy(dsdv,w):

dsdv = list(dsdv)

dsdv.sort(key = lambda x:x.don\_gia, reverse=True)

for i in range(len(dsdv)):

dsdv[i].phuong\_an = min((w//dsdv[i].trong\_luong),1)

w -= dsdv[i].phuong\_an \* dsdv[i].trong\_luong

# Tính tổng trọng lượng và tổng giá trị

tong\_trong\_luong = sum([dv.trong\_luong \* dv.phuong\_an for dv in dsdv])

tong\_gia\_tri = sum([dv.gia\_tri \* dv.phuong\_an for dv in dsdv])

return dsdv, tong\_trong\_luong, tong\_gia\_tri

dsdv = [Do\_vat("A",15,30),Do\_vat("B",10,25),Do\_vat("C",2,2),Do\_vat("D",4,6)]

def in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri):

print("Danh sách đồ vật được chọn:")

for dv in dsdv:

if dv.phuong\_an > 0:

print("Tên: ", dv.ten, " - Trọng lượng: ", dv.trong\_luong, " - Giá trị: ", dv.gia\_tri," - Đơn giá:",dv.don\_gia,"-Phương án:",dv.phuong\_an)

print("\nTổng trọng lượng: ", tong\_trong\_luong)

print("Tổng giá trị: ", tong\_gia\_tri)

w = 37

dsdv, tong\_trong\_luong, tong\_gia\_tri = greedy(dsdv, w)

in\_ket\_qua(dsdv, tong\_trong\_luong, tong\_gia\_tri)

**Tổ hợp**

**Bảng:**

def Comb(n,k):

C = [[0 for \_ in range(n+1)] for \_ in range(n+1)]

C[0][0] = 1

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

C[i][0] = 1

C[i][i] = 1

for j in range(1,i):

C[i][j] = C[i-1][j-1] + C[i-1][j]

return C[n][k]

n = 10

k = 3

print(Comb(n,k))

**Vector**

def Comb(n,k):

V = [0 for \_ in range(n+1)]

V[0] = 1

V[1] = 1

for i in range(2,n+1):

p1 = V[0]

for j in range(1,i):

p2 = V[j]

V[j] = p1 +p2

p1 = p2

V[i] = 1

return V[k]

n = 10

k = 3

print(Comb(n,k))

**Balo Quy Hoạch Động:**

class DoVat:

def \_\_init\_\_(self, ten, tl, gt, so\_dv\_duoc\_chon=0):

self.ten = ten

self.gt = gt

self.tl = tl

self.so\_dv\_duoc\_chon = so\_dv\_duoc\_chon

def tao\_bang(dsdv, n, W):

F = [[0 for \_ in range(W+1)] for \_ in range(n)]

X = [[0 for \_ in range(W+1)] for \_ in range(n)]

for V in range(dsdv[0].tl, W+1):

X[0][V] = V // dsdv[0].tl

F[0][V] = X[0][V] \* dsdv[0].gt

for k in range(1, n):

for V in range(W+1):

FMax = F[k-1][V]

XMax = 0

yk = V // dsdv[k].tl

#Ba lo 2: yk = min(1,V // dsdv[k].tl)

#Ba lo 3: yk = min(V // dsdv[k].tl,dsdv[k].sl)

for xk in range(1, yk+1):

if F[k-1][V-xk\*dsdv[k].tl] + xk\*dsdv[k].gt > FMax:

FMax = F[k-1][V-xk\*dsdv[k].tl] + xk\*dsdv[k].gt

XMax = xk

F[k][V] = FMax

X[k][V] = XMax

return F, X

def tra\_bang(dsdv, n, W, X):

k = n-1

V = W

while k >= 0:

dsdv[k].so\_dv\_duoc\_chon = X[k][V]

V -= X[k][V] \* dsdv[k].tl

k -= 1

def in\_bang(n, W, F, X):

print("Bảng F và X:")

for k in range(n):

for V in range(W+1):

print(f"{F[k][V]:.1f} {X[k][V]:2d}", end="||")

print()

def in\_kq(dsdv, W):

print("Phương án thu được từ kỹ thuật QUY HOẠCH ĐỘNG như sau:")

tong\_tl = tong\_gt = 0

for dv in dsdv:

if dv.so\_dv\_duoc\_chon > 0:

tong\_tl += dv.so\_dv\_duoc\_chon \* dv.tl

tong\_gt += dv.so\_dv\_duoc\_chon \* dv.gt

print(f"Tổng trọng lượng của ba lô: {tong\_tl}")

print(f"Tổng giá trị của ba lô: {tong\_gt}")

# Khởi tạo danh sách đồ vật

dsdv = [

DoVat("1", 3, 4),

DoVat("2", 5, 5),

DoVat("3", 5, 6),

DoVat("4", 2, 3),

DoVat("5", 1, 1)

]

W = 9 # Khối lượng tối đa của ba lô

n = len(dsdv) # Số lượng đồ vật

F, X = tao\_bang(dsdv, n, W)

tra\_bang(dsdv, n, W, X)

in\_bang(n, W, F, X)

in\_kq(dsdv, W)

**Định trị biểu thức số học:**

class Node:

def \_\_init\_\_(self,val,left = None, right = None):

self.val = val

self.right = right

self.left = left

def Dinh\_Tri(root):

if root is None:

return 0

if root.left is None and root.right is None:

return float(root.val)

if root.val == '+':

return Dinh\_Tri(root.left) + Dinh\_Tri(root.right)

elif root.val == '-':

return Dinh\_Tri(root.left) - Dinh\_Tri(root.right)

elif root.val == '\*':

return Dinh\_Tri(root.left) \* Dinh\_Tri(root.right)

elif root.val == '/':

try:

return Dinh\_Tri(root.left) / Dinh\_Tri(root.right)

except ZeroDivisionError:

print("Không thể chia cho 0!")

return 0

else: return 0

def inorder(root):

if root is not None:

inorder(root.left)

print(root.val, end=' ')

inorder(root.right)

def read\_tree():

x = input("Nhập giá trị cho nút (nhập '0' để bỏ qua): ")

if x == '0':

return None

else:

node = Node(x)

print(f"Nhập con trái của {x}: ")

node.left = read\_tree()

print(f"Nhập con phải của {x}: ")

node.right = read\_tree()

return node

node = read\_tree()

print("Cây biểu thức sau khi duyệt LNR là:")

inorder(node)

print("\nTrị của biểu thức là: ",Dinh\_Tri(node))

**Bài toán mô phỏng caro**

# Khởi tạo bàn cờ

board = [[' ' for \_ in range(3)] for \_ in range(3)]

# Vẽ bàn cờ

def draw\_board():

for row in board:

print('|'.join(row))

print('-' \* 7)

# Kiểm tra xem đã có người chiến thắng chưa

def check\_winner(row, col, player):

# Kiểm tra hàng ngang

if board[row][0] == board[row][1] == board[row][2] == player:

return True

# Kiểm tra hàng dọc

if board[0][col] == board[1][col] == board[2][col] == player:

return True

# Kiểm tra đường chéo chính

if board[0][0] == board[1][1] == board[2][2] == player:

return True

# Kiểm tra đường chéo phụ

if board[0][2] == board[1][1] == board[2][0] == player:

return True

return False

# Chạy trò chơi

def play\_game():

player = 'X' # Người chơi đầu tiên là X

while True:

draw\_board()

# Nhập vị trí đặt ký tự

row = int(input("Nhập hàng: "))

col = int(input("Nhập cột: "))

# Kiểm tra vị trí hợp lệ

if row < 0 or row >= 3 or col < 0 or col >= 3 or board[row][col] != ' ':

print("Vị trí không hợp lệ, vui lòng thử lại!")

continue

# Đặt ký tự vào bàn cờ

board[row][col] = player

# Kiểm tra xem đã có người chiến thắng chưa

if check\_winner(row, col, player):

draw\_board()

print("Người chơi", player, "thắng!")

break

# Chuyển lượt người chơi

player = 'O' if player == 'X' else 'X'

# Bắt đầu trò chơi

play\_game()

**Cây phủ**

class Graph:

def \_\_init\_\_(self, vertices):

self.V = vertices

self.graph = []

self.nodes = {}

def add\_edge(self, u, v, w):

if u not in self.nodes:

self.nodes[u] = len(self.nodes)

if v not in self.nodes:

self.nodes[v] = len(self.nodes)

self.graph.append([self.nodes[u], self.nodes[v], w])

def find(self, parent, i):

if parent[i] == i:

return i

return self.find(parent, parent[i])

def union(self, parent, rank, x, y):

xroot = self.find(parent, x)

yroot = self.find(parent, y)

if rank[xroot] < rank[yroot]:

parent[xroot] = yroot

elif rank[xroot] > rank[yroot]:

parent[yroot] = xroot

else:

parent[yroot] = xroot

rank[xroot] += 1

def local\_search(self):

result = []

self.graph = sorted(self.graph, key=lambda item: item[2])

parent = []

rank = []

for node in range(self.V):

parent.append(node)

rank.append(0)

e = 0

while len(result) < self.V - 1:

u, v, w = self.graph[e]

e += 1

x = self.find(parent, u)

y = self.find(parent, v)

if x != y:

result.append([list(self.nodes.keys())[list(self.nodes.values()).index(u)],

list(self.nodes.keys())[list(self.nodes.values()).index(v)], w])

self.union(parent, rank, x, y)

return result

g = Graph(5)

g.add\_edge('A', 'B', 3)

g.add\_edge('A', 'C', 4)

g.add\_edge('A', 'D', 2)

g.add\_edge('A', 'E', 7)

g.add\_edge('B', 'C', 4)

g.add\_edge('B', 'D', 6)

g.add\_edge('B', 'E', 3)

g.add\_edge('C', 'D', 5)

g.add\_edge('C', 'E', 8)

g.add\_edge('D', 'E', 6)

results = g.local\_search()

total\_weight = 0

for u, v, w in results:

print(f"{u}{v} = {w}")

total\_weight += w

print(f"Giá của chu trình : {total\_weight}")

**Nhập ma trận và tìm phần tử nhỏ nhất**

**Cách 1**

matrix = [  
 [5,2,7,4],  
 [3,2,-4,8],  
 [9,7,3,5],  
 [6,4,1,-10]  
]  
  
def Min\_CDT(matrix,row\_start,row\_end,column\_start,column\_end):  
 if row\_start == row\_end and column\_start == column\_end:  
 return matrix[row\_start][column\_start]  
  
 row\_mid = (row\_start + row\_end) // 2  
 column\_mid = (column\_start + column\_end) // 2  
 m1= Min\_CDT(matrix,row\_start,row\_mid,column\_start,column\_mid)  
 m2 = Min\_CDT(matrix,row\_start,row\_mid,column\_mid+1,column\_end)  
 m3 = Min\_CDT(matrix,row\_mid+1,row\_end,column\_start,column\_mid)  
 m4 = Min\_CDT(matrix,row\_mid+1,row\_end,column\_mid+1,column\_end)  
 return min(m1,m2,m3,m4)  
n = len(matrix)-1  
a= Min\_CDT(matrix,0,n,0,n)  
print(a)

**Cách 2**

def khoi\_tao\_ma\_tran(so\_hang, so\_cot):  
 ma\_tran = []  
 for i in range(so\_hang):  
 hang = []  
 for j in range(so\_cot):  
 gia\_tri = int(input(f"Nhập giá trị tại vị trí ({i+1},{j+1}): "))  
 hang.append(gia\_tri)  
 ma\_tran.append(hang)  
 return ma\_tran  
  
def tim\_gia\_tri\_nho\_nhat(ma\_tran):  
 n = len(ma\_tran)  
 if n == 1:  
 return ma\_tran[0][0]  
 else:  
 giua = n // 2  
 nua\_tren = [hang[:giua] for hang in ma\_tran[:giua]]  
 nua\_duoi = [hang[giua:] for hang in ma\_tran[giua:]]  
 min\_nua\_tren = tim\_gia\_tri\_nho\_nhat(nua\_tren)  
 min\_nua\_duoi = tim\_gia\_tri\_nho\_nhat(nua\_duoi)  
 return min(min\_nua\_tren, min\_nua\_duoi)  
  
# Nhập số hàng và số cột của ma trận  
so\_hang = int(input("Nhập số hàng của ma trận: "))  
so\_cot = int(input("Nhập số cột của ma trận: "))  
ma\_tran = khoi\_tao\_ma\_tran(so\_hang, so\_cot)  
  
# Tìm giá trị nhỏ nhất bằng kỹ thuật chia để trị  
gia\_tri\_nho\_nhat = tim\_gia\_tri\_nho\_nhat(ma\_tran)  
  
# In kết quả  
print("Giá trị nhỏ nhất:", gia\_tri\_nho\_nhat)

**hàm Xóa trùng**

def XoaTrung(self):  
 n = len(self.ds)-1  
 a =[]  
 for i in range(0,n):  
 for i in self.ds:  
 if i not in a:  
 a.append(i)  
 self.ds = a

**KHANHDUY**

class ATM:

def \_\_init\_\_(self):

self.menhgia = [500000, 200000, 100000, 50000, 20000, 10000, 5000, 2000, 1000]

self.ds\_menh\_gia = []

def Nhap(self):

n = input("Nhập số tiền của bạn: ")

while n.isdigit()==False:

n=input("Nhập lại số tiền của bạn: ")

return n

def DoiMenhGia(self,n):

print("Cách mệnh giá bạn có thể chọn:")# có hoặc không cũng được

print(self.menhgia) # có hoặc không cũng được

x = input("Nhập mệnh giá tối đa bạn muốn chọn: ")

while x.isdigit()==False:

print("Mệnh giá không phù hợp!")

x=input("Nhập lại số mệnh giá tối đa bạn muôn chọn: ")

while int(x) > int(n):

print("Số mệnh giá tối đa vượt quá số tiền hiện có. Vui lòng nhập lại!")

x = input("Nhập số mệnh giá tối đa bạn muốn chọn: ")

while x.isdigit() == False:

print("Mệnh giá không phù hợp!")

x = input("Nhập lại số mệnh giá tối đa bạn muôn chọn: ")

print("Các mệnh giá bạn có thể đổi: ", self.MenhGiaCoTheDoi(int(x)))

def MenhGiaCoTheDoi(self, x):

self.ds\_menh\_gia = []

for menh\_gia in self.menhgia:

if menh\_gia <= int(x):

self.ds\_menh\_gia.append(menh\_gia)

return self.ds\_menh\_gia

def ChonKhongGH(self,x,n):

x=self.ds\_menh\_gia

count=0

total=0

phuongan=[]

for i in x:

while total<=int(n):

count+=1

total+=i

if total>int(n):

total-=i

count-=1

break

phuongan.append((i,count))

count=0

return phuongan

def ChonGH1To(self,x,n):

x=self.ds\_menh\_gia

count=0

toal=0

so\_to=1

phuong\_an=[]

for i in x:

while so\_to>0:

so\_to-=1

count+=1

toal+=i

if toal>int(n):

toal-=i

count-=1

phuong\_an.append((i,count))

count=0

so\_to=1

return phuong\_an

def ChonTuyY(self,x,n):

s=input("Nhập số tờ bạn muốn:")

while s.isdigit()==False:

print("Không hợp lệ. Vui lòng nhập lại!")

s=input("Nhập lại số tờ bạn muốn:")

so\_to = int(s)

x=self.ds\_menh\_gia

count=0

toal=0

phuong\_an=[]

for i in x:

while int(so\_to) >0:

so\_to-=1

count+=1

toal+=i

if toal>int(n):

toal-=i

count-=1

break

phuong\_an.append((i,count))

count=0

so\_to=int(s)

return phuong\_an

a = ATM()

n=a.Nhap()

x=a.DoiMenhGia(n)

print("Đổi không giới hạn số tờ:")

results=a.ChonKhongGH(x,n)

for i,j in results:

print(f"{i}:{j} tờ")

print("Đổi số tờ giới hạn là 1:")

results=a.ChonGH1To(x,n)

for i,j in results:

print(f"{i}:{j} tờ")

print("Đổi số tờ tùy ý:")

results=a.ChonTuyY(x,n)

for i,j in results:

print(f"{i}:{j} tờ")

+++++++++++++++++++++++++++++++++++++

# tìm giá tr lớn nhất và nhỏ chia để trị

def khoi\_tao\_ma\_tran(so\_hang, so\_cot):

ma\_tran = []

for i in range(so\_hang):

hang = []

for j in range(so\_cot):

gia\_tri = int(input(f"Nhập giá trị tại vị trí ({i+1},{j+1}): "))

hang.append(gia\_tri)

ma\_tran.append(hang)

return ma\_tran

def tim\_gia\_tri\_nho\_nhat(ma\_tran):

def tim\_gia\_tri\_nho\_nhat\_trong\_hang(hang):

min\_gia\_tri = float('inf') # Giả định giá trị nhỏ nhất là vô cùng lớn

for gia\_tri in hang:

if gia\_tri < min\_gia\_tri:

min\_gia\_tri = gia\_tri

return min\_gia\_tri

min\_gia\_tri = float('inf') # Giả định giá trị nhỏ nhất là vô cùng lớn

for hang in ma\_tran:

min\_hang = tim\_gia\_tri\_nho\_nhat\_trong\_hang(hang)

if min\_hang < min\_gia\_tri:

min\_gia\_tri = min\_hang

return min\_gia\_tri

def tim\_gia\_tri\_lon\_nhat(ma\_tran):

def tim\_gia\_tri\_lon\_nhat\_trong\_hang(hang):

max\_gia\_tri = float('-inf') # Giả định giá trị lớn nhất là âm vô cùng

for gia\_tri in hang:

if gia\_tri > max\_gia\_tri:

max\_gia\_tri = gia\_tri

return max\_gia\_tri

max\_gia\_tri = float('-inf') # Giả định giá trị lớn nhất là âm vô cùng

for hang in ma\_tran:

max\_hang = tim\_gia\_tri\_lon\_nhat\_trong\_hang(hang)

if max\_hang > max\_gia\_tri:

max\_gia\_tri = max\_hang

return max\_gia\_tri

so\_hang = int(input("Nhập số hàng của ma trận: "))

so\_cot = int(input("Nhập số cột của ma trận: "))

ma\_tran = khoi\_tao\_ma\_tran(so\_hang, so\_cot)

gia\_tri\_nho\_nhat = tim\_gia\_tri\_nho\_nhat(ma\_tran)

gia\_tri\_lon\_nhat = tim\_gia\_tri\_lon\_nhat(ma\_tran)

print("Giá trị nhỏ nhất:", gia\_tri\_nho\_nhat)

print("Giá trị lớn nhất:", gia\_tri\_lon\_nhat)

++++++++++++++++++++++++++++++++++

#mảng 1 chiều chia dề trị

def khoi\_tao\_mang(so\_phan\_tu):

mang = []

for i in range(so\_phan\_tu):

gia\_tri = int(input(f"Nhập giá trị phần tử thứ {i+1}: "))

mang.append(gia\_tri)

return mang

def tim\_gia\_tri\_nho\_nhat(mang):

if len(mang) == 1:

return mang[0]

else:

giua = len(mang) // 2

nua\_tren = mang[:giua]

nua\_duoi = mang[giua:]

min\_nua\_tren = tim\_gia\_tri\_nho\_nhat(nua\_tren)

min\_nua\_duoi = tim\_gia\_tri\_nho\_nhat(nua\_duoi)

return min(min\_nua\_tren, min\_nua\_duoi)

def tim\_gia\_tri\_lon\_nhat(mang):

if len(mang) == 1:

return mang[0]

else:

giua = len(mang) // 2

nua\_tren = mang[:giua]

nua\_duoi = mang[giua:]

max\_nua\_tren = tim\_gia\_tri\_lon\_nhat(nua\_tren)

max\_nua\_duoi = tim\_gia\_tri\_lon\_nhat(nua\_duoi)

return max(max\_nua\_tren, max\_nua\_duoi)

# Nhập số phần tử của mảng

so\_phan\_tu = int(input("Nhập số phần tử của mảng: "))

mang = khoi\_tao\_mang(so\_phan\_tu)

# Tìm giá trị nhỏ nhất bằng kỹ thuật chia để trị

gia\_tri\_nho\_nhat = tim\_gia\_tri\_nho\_nhat(mang)

# Tìm giá trị lớn nhất bằng kỹ thuật chia để trị

gia\_tri\_lon\_nhat = tim\_gia\_tri\_lon\_nhat(mang)

# In kết quả

print("Giá trị nhỏ nhất:", gia\_tri\_nho\_nhat)

print("Giá trị lớn nhất:", gia\_tri\_lon\_nhat)

+++++++++++++++++++++++++++++++++++

2 chiều tham an

#tim gia tri lon nhất và nhỏ nhất bằng kĩ thuật tham ăn

def khoi\_tao\_ma\_tran(so\_hang, so\_cot):

ma\_tran = []

for i in range(so\_hang):

hang = []

for j in range(so\_cot):

gia\_tri = int(input(f"Nhập giá trị tại vị trí ({i+1},{j+1}): "))

hang.append(gia\_tri)

ma\_tran.append(hang)

return ma\_tran

def tim\_gia\_tri\_nho\_nhat(ma\_tran):

gia\_tri\_nho\_nhat = float('inf')

for hang in ma\_tran:

for gia\_tri in hang:

if gia\_tri < gia\_tri\_nho\_nhat:

gia\_tri\_nho\_nhat = gia\_tri

return gia\_tri\_nho\_nhat

def tim\_gia\_tri\_lon\_nhat(ma\_tran):

gia\_tri\_lon\_nhat = float('-inf')

for hang in ma\_tran:

for gia\_tri in hang:

if gia\_tri > gia\_tri\_lon\_nhat:

gia\_tri\_lon\_nhat = gia\_tri

return gia\_tri\_lon\_nhat

def tim\_va\_in\_gia\_tri\_nho\_nhat(ma\_tran):

gia\_tri\_nho\_nhat = tim\_gia\_tri\_nho\_nhat(ma\_tran)

print("Giá trị nhỏ nhất:", gia\_tri\_nho\_nhat)

def tim\_va\_in\_gia\_tri\_lon\_nhat(ma\_tran):

gia\_tri\_lon\_nhat = tim\_gia\_tri\_lon\_nhat(ma\_tran)

print("Giá trị lớn nhất:", gia\_tri\_lon\_nhat)

# Nhập số hàng và số cột của ma trận

so\_hang = int(input("Nhập số hàng của ma trận: "))

so\_cot = int(input("Nhập số cột của ma trận: "))

ma\_tran = khoi\_tao\_ma\_tran(so\_hang, so\_cot)

# Tìm và in giá trị nhỏ nhất

tim\_va\_in\_gia\_tri\_nho\_nhat(ma\_tran)

# Tìm và in giá trị lớn nhất

tim\_va\_in\_gia\_tri\_lon\_nhat(ma\_tran)

+++++++++++++++++++++++++

mảng 1 chiều tham an c1

def khoi\_tao\_mang(so\_phan\_tu):

mang = []

for i in range(so\_phan\_tu):

gia\_tri = int(input(f"Nhập giá trị phần tử thứ {i+1}: "))

mang.append(gia\_tri)

return mang

def tim\_gia\_tri\_nho\_nhat(mang):

gia\_tri\_nho\_nhat = float('inf') # Khởi tạo giá trị nhỏ nhất ban đầu với giá trị dương vô cùng

for gia\_tri in mang:

if gia\_tri < gia\_tri\_nho\_nhat:

gia\_tri\_nho\_nhat = gia\_tri

return gia\_tri\_nho\_nhat

def tim\_gia\_tri\_lon\_nhat(mang):

gia\_tri\_lon\_nhat = float('-inf') # Khởi tạo giá trị lớn nhất ban đầu với giá trị âm vô cùng

for gia\_tri in mang:

if gia\_tri > gia\_tri\_lon\_nhat:

gia\_tri\_lon\_nhat = gia\_tri

return gia\_tri\_lon\_nhat

# Nhập số phần tử của mảng

so\_phan\_tu = int(input("Nhập số phần tử của mảng: "))

mang = khoi\_tao\_mang(so\_phan\_tu)

# Tìm giá trị nhỏ nhất bằng kỹ thuật tham ăn

gia\_tri\_nho\_nhat = tim\_gia\_tri\_nho\_nhat(mang)

# Tìm giá trị lớn nhất bằng kỹ thuật tham ăn

gia\_tri\_lon\_nhat = tim\_gia\_tri\_lon\_nhat(mang)

# In kết quả

print("Giá trị nhỏ nhất:", gia\_tri\_nho\_nhat)

print("Giá trị lớn nhất:", gia\_tri\_lon\_nhat)

+++++++++++++++++++++++++++++++

Mảng 1 chiều của tham an

def khoi\_tao\_mang(so\_phan\_tu):

mang = []

for i in range(so\_phan\_tu):

gia\_tri = int(input(f"Nhập giá trị phần tử thứ {i+1}: "))

mang.append(gia\_tri)

return mang

def tim\_gia\_tri\_nho\_nhat(mang):

gia\_tri\_nho\_nhat = mang[0]

for gia\_tri in mang:

if gia\_tri < gia\_tri\_nho\_nhat:

gia\_tri\_nho\_nhat = gia\_tri

return gia\_tri\_nho\_nhat

def tim\_gia\_tri\_lon\_nhat(mang):

gia\_tri\_lon\_nhat = mang[0]

for gia\_tri in mang:

if gia\_tri > gia\_tri\_lon\_nhat:

gia\_tri\_lon\_nhat = gia\_tri

return gia\_tri\_lon\_nhat

# Nhập số phần tử của mảng

so\_phan\_tu = int(input("Nhập số phần tử của mảng: "))

mang = khoi\_tao\_mang(so\_phan\_tu)

# Tìm giá trị nhỏ nhất bằng kỹ thuật tham ăn

gia\_tri\_nho\_nhat = tim\_gia\_tri\_nho\_nhat(mang)

# Tìm giá trị lớn nhất bằng kỹ thuật tham ăn

gia\_tri\_lon\_nhat = tim\_gia\_tri\_lon\_nhat(mang)

# In kết quả

print("Giá trị nhỏ nhất:", gia\_tri\_nho\_nhat)

print("Giá trị lớn nhất:", gia\_tri\_lon\_nhat)

+++++++++++++++++++

#Kĩ thuật quay lui

def khoi\_tao\_ma\_tran(so\_hang, so\_cot):

ma\_tran = []

for i in range(so\_hang):

hang = []

for j in range(so\_cot):

gia\_tri = int(input(f"Nhập giá trị tại vị trí ({i+1},{j+1}): "))

hang.append(gia\_tri)

ma\_tran.append(hang)

return ma\_tran

def tim\_gia\_tri\_nho\_nhat(ma\_tran):

min\_gia\_tri = float('inf')

def backtrack(i, j, cur\_min):

if i >= len(ma\_tran) or j >= len(ma\_tran[0]):

return

cur\_min = min(cur\_min, ma\_tran[i][j])

if i == len(ma\_tran) - 1 and j == len(ma\_tran[0]) - 1:

nonlocal min\_gia\_tri

min\_gia\_tri = min(min\_gia\_tri, cur\_min)

return

backtrack(i + 1, j, cur\_min)

backtrack(i, j + 1, cur\_min)

backtrack(0, 0, float('inf'))

return min\_gia\_tri

def tim\_gia\_tri\_lon\_nhat(ma\_tran):

max\_gia\_tri = float('-inf')

def backtrack(i, j, cur\_max):

if i >= len(ma\_tran) or j >= len(ma\_tran[0]):

return

cur\_max = max(cur\_max, ma\_tran[i][j])

if i == len(ma\_tran) - 1 and j == len(ma\_tran[0]) - 1:

nonlocal max\_gia\_tri

max\_gia\_tri = max(max\_gia\_tri, cur\_max)

return

backtrack(i + 1, j, cur\_max)

backtrack(i, j + 1, cur\_max)

backtrack(0, 0, float('-inf'))

return max\_gia\_tri

# Nhập số hàng và số cột của ma trận

so\_hang = int(input("Nhập số hàng của ma trận: "))

so\_cot = int(input("Nhập số cột của ma trận: "))

ma\_tran = khoi\_tao\_ma\_tran(so\_hang, so\_cot)

# Tìm giá trị nhỏ nhất bằng kỹ thuật quay lui

gia\_tri\_nho\_nhat = tim\_gia\_tri\_nho\_nhat(ma\_tran)

# Tìm giá trị lớn nhất bằng kỹ thuật quay lui

gia\_tri\_lon\_nhat = tim\_gia\_tri\_lon\_nhat(ma\_tran)

# In kết quả

print("Giá trị nhỏ nhất:", gia\_tri\_nho\_nhat)

print("Giá trị lớn nhất:", gia\_tri\_lon\_nhat)

++++++++++++++++++++++++++++++++++++

#1 chiều qui lui

def khoi\_tao\_mang(so\_phan\_tu):

mang = []

for i in range(so\_phan\_tu):

gia\_tri = int(input(f"Nhập giá trị phần tử thứ {i+1}: "))

mang.append(gia\_tri)

return mang

def tim\_gia\_tri\_nho\_nhat(mang):

min\_gia\_tri = float('inf')

def backtrack(index, cur\_min):

nonlocal min\_gia\_tri

if index >= len(mang):

min\_gia\_tri = min(min\_gia\_tri, cur\_min)

return

cur\_min = min(cur\_min, mang[index])

backtrack(index + 1, cur\_min)

backtrack(0, float('inf'))

return min\_gia\_tri

def tim\_gia\_tri\_lon\_nhat(mang):

max\_gia\_tri = float('-inf')

def backtrack(index, cur\_max):

nonlocal max\_gia\_tri

if index >= len(mang):

max\_gia\_tri = max(max\_gia\_tri, cur\_max)

return

cur\_max = max(cur\_max, mang[index])

backtrack(index + 1, cur\_max)

backtrack(0, float('-inf'))

return max\_gia\_tri

# Nhập số phần tử của mảng

so\_phan\_tu = int(input("Nhập số phần tử của mảng: "))

mang = khoi\_tao\_mang(so\_phan\_tu)

# Tìm giá trị nhỏ nhất bằng kỹ thuật quay lui

gia\_tri\_nho\_nhat = tim\_gia\_tri\_nho\_nhat(mang)

# Tìm giá trị lớn nhất bằng kỹ thuật quay lui

gia\_tri\_lon\_nhat = tim\_gia\_tri\_lon\_nhat(mang)

# In kết quả

print("Giá trị nhỏ nhất:", gia\_tri\_nho\_nhat)

print("Giá trị lớn nhất:", gia\_tri\_lon\_nhat)

+++++++++++++++++++++

#kĩ thuật qui hoạt động

def khoi\_tao\_ma\_tran(so\_hang, so\_cot):

ma\_tran = []

for i in range(so\_hang):

hang = []

for j in range(so\_cot):

gia\_tri = int(input(f"Nhập giá trị tại vị trí ({i+1},{j+1}): "))

hang.append(gia\_tri)

ma\_tran.append(hang)

return ma\_tran

def tim\_gia\_tri\_nho\_nhat(ma\_tran):

so\_hang = len(ma\_tran)

so\_cot = len(ma\_tran[0])

dp = [[0] \* so\_cot for \_ in range(so\_hang)]

dp[0][0] = ma\_tran[0][0]

for i in range(1, so\_hang):

dp[i][0] = min(dp[i-1][0], ma\_tran[i][0])

for j in range(1, so\_cot):

dp[0][j] = min(dp[0][j-1], ma\_tran[0][j])

for i in range(1, so\_hang):

for j in range(1, so\_cot):

dp[i][j] = min(dp[i-1][j], dp[i][j-1], ma\_tran[i][j])

return dp[so\_hang-1][so\_cot-1]

def tim\_gia\_tri\_lon\_nhat(ma\_tran):

so\_hang = len(ma\_tran)

so\_cot = len(ma\_tran[0])

dp = [[0] \* so\_cot for \_ in range(so\_hang)]

dp[0][0] = ma\_tran[0][0]

for i in range(1, so\_hang):

dp[i][0] = max(dp[i-1][0], ma\_tran[i][0])

for j in range(1, so\_cot):

dp[0][j] = max(dp[0][j-1], ma\_tran[0][j])

for i in range(1, so\_hang):

for j in range(1, so\_cot):

dp[i][j] = max(dp[i-1][j], dp[i][j-1], ma\_tran[i][j])

return dp[so\_hang-1][so\_cot-1]

# Nhập số hàng và số cột của ma trận

so\_hang = int(input("Nhập số hàng của ma trận: "))

so\_cot = int(input("Nhập số cột của ma trận: "))

ma\_tran = khoi\_tao\_ma\_tran(so\_hang, so\_cot)

# Tìm giá trị nhỏ nhất bằng kỹ thuật quy hoạch động

gia\_tri\_nho\_nhat = tim\_gia\_tri\_nho\_nhat(ma\_tran)

# Tìm giá trị lớn nhất bằng kỹ thuật quy hoạch động

gia\_tri\_lon\_nhat = tim\_gia\_tri\_lon\_nhat(ma\_tran)

# In kết quả

print("Giá trị nhỏ nhất:", gia\_tri\_nho\_nhat)

print("Giá trị lớn nhất:", gia\_tri\_lon\_nhat)

+++++++++++++++++++++++++++++++++

#Mảng 1 chiều của qui hoạt dộng   
def khoi\_tao\_mang(so\_phan\_tu):

mang = []

for i in range(so\_phan\_tu):

gia\_tri = int(input(f"Nhập giá trị phần tử thứ {i+1}: "))

mang.append(gia\_tri)

return mang

def tim\_gia\_tri\_nho\_nhat(mang):

so\_phan\_tu = len(mang)

dp = [0] \* so\_phan\_tu

dp[0] = mang[0]

for i in range(1, so\_phan\_tu):

dp[i] = min(dp[i-1], mang[i])

return dp[so\_phan\_tu-1]

def tim\_gia\_tri\_lon\_nhat(mang):

so\_phan\_tu = len(mang)

dp = [0] \* so\_phan\_tu

dp[0] = mang[0]

for i in range(1, so\_phan\_tu):

dp[i] = max(dp[i-1], mang[i])

return dp[so\_phan\_tu-1]

# Nhập số phần tử của mảng

so\_phan\_tu = int(input("Nhập số phần tử của mảng: "))

mang = khoi\_tao\_mang(so\_phan\_tu)

# Tìm giá trị nhỏ nhất bằng kỹ thuật quy hoạch động

gia\_tri\_nho\_nhat = tim\_gia\_tri\_nho\_nhat(mang)

# Tìm giá trị lớn nhất bằng kỹ thuật quy hoạch động

gia\_tri\_lon\_nhat = tim\_gia\_tri\_lon\_nhat(mang)

# In kết quả

print("Giá trị nhỏ nhất:", gia\_tri\_nho\_nhat)

print("Giá trị lớn nhất:", gia\_tri\_lon\_nhat)