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

```
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("Nhập 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("Nhập 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 N^2

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): # Để 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 = 12345678901234567890123456789O
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 vẫn 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

Xây dựng lớp Danhsach – Code ok

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.

Merge Sort (Class Sapxep kethua Danhsach) - Ok

CÁC PHƯƠNG PHÁP SẮP XẾP THUỘC LỚP HÀM N^2

Phương pháp chọn (Selection Sort) (Class Sapxep_N^2 kethua Danhsach)

Phương pháp chèn (Insertion Sort) (Class Sapxep_N^2 kethua Danhsach)

Phương pháp nổi bọt (Bubble Sort) (Class Sapxep_N^2 kethua Danhsach)

CÁC PHƯƠNG PHÁP SẮP XẾP THUỘC LỚP HÀM NLogN

Phương pháp Heap Sort (Class Sapxep_NLogN kethua Danhsach) - Ok

Phương pháp Quick Sort (Class Sapxep_NLogN kethua Danhsach) - Ok

CÁC PHƯƠNG PHÁP SẮP XẾP THUỘC LỚP HÀM N

Phương pháp Radix Sort (Class Sapxep_N kethua Danhsach) - Ok

Phương pháp Bin Sort (Class Sapxep_N kethua Danhsach)

Trường hợp đơn giản

Trường hợp tổng quát

Trường hợp tập giá trị lớn

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

Tìm kiếm tuyến tính (Linear Search)

Không đệ qui

Đệ qui - Ok

Tìm kiếm nhị phân (Binary Search)

Không đệ qui

Đệ qui

KỸ THUẬT CHIA ĐỂ TRỊ

Bài toán nhân các số nguyên lớn

Thuật toán thông thường

Thuật toán áp dụng kỹ thuật chia để trị

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.

Nhân hai ma trận nhiều giá trị:

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

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

Có trường hợp nếu nhập số lẻ, chẵn:

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

Trường hợp số chẵn

Trường hợp số lẽ, chẵn:

KỸ THUẬT THAM ĂN

Thuật toán thông thường

Thuật toán áp dụng kỹ thuật tham ăn

ATM:

-----Không giới hạn số tờ-----

----Giới hạn số tờ là 1-----

-----Số tờ tùy chọn -----

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

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

Cái ba lô:

----Không giới hạn-----

-------Tùy chọn theo số lượng-----

-----Mỗi đồ vật chỉ chọn 1---

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

Week7_Baitap2_Mang2chieu:

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

Week7_Baitap2_Mang1Chieu:

Bài toán cái ba lô - Week7_Baitap3:

TH1: Chọn đồ vấn có đơn giá cao nhất cho đến khi không thể chọn thêm đồ vật nào class Do_vat:

TH2: Có số lượng

TH3: Chọn tối đa một đồ vật của từng loại

KỸ THUẬT QUAY LUI

Định trị cây biểu thức số học

Mô phỏng trò chơi ca rô

Cắt tỉa Alpha – Beta

Vét cạn định trị:

KỸ THUẬT TÌM KIẾM ĐỊA PHƯƠNG

Tìm số nguyên tố

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

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

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.

TẬP TIN BẢN BĂM

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

ĐỀ THAM KHẢO HỌC THÊM

Xây dựng lớp HinhHoc:

Tìm ước chung lớn nhất bằng thuật toán Euclid:

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

Sắp xếp Shell Sort (Class Sapxep_TheoKhoangCach kế thừa từ Danhsach):

Đếm số lượng đảo ngược trong mảng:

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

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

Phương pháp tìm kiếm Interpolation Search:

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

Tìm chu trình Hamiltonian trong đồ thị:

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("Nhập a = "))
b = int(input("Nhập b = "))
somu_n = int(input("Nhập số mũ của n = "))
T(a, b, somu_n)
```

Xây dựng 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 = 12345678901234567890123456789O
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("Nhập a = "))
b = int(input("Nhập b = "))
somu_n = int(input("Nhập số mũ của n = "))

T(a, b, somu_n)
```

2/ Xây dựng 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_e
nd):
    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_m
id)
    m2 =
Min_CDT(matrix,row_start,row_mid,column_mid+1,column_e
nd)
    m3 =
Min_CDT(matrix,row_mid+1,row_end,column_start,column_m
id)
    m4 =
Min_CDT(matrix,row_mid+1,row_end,column_mid+1,column_e
nd)
    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 động

```
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)
```