**1/ Hàm nhân**

import math

def f(b, n):

return b\*\*n

def d(n):

return n # Hàm d(n) = n

def ktraHamNhan(d, m, n):

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

def T(a, b, n):

if ktraHamNhan(d, a, b):

if a > f(b, n):

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

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

elif a < f(b, n):

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

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

else:

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

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

else:

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

a = int(input("Nhâp a = "))

b = int(input("Nhập b = "))

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

T(a, b, somu\_n)

**2/ Xây dụng lớp DanhSach**

class Danhsach:

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

self.ds = []

def nhap(self):

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

for i in range(n):

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

def xuat(self):

for i in self.ds:

print(i, end=' ')

print()

def tim(self, x):

if x in self.ds:

return self.ds.index(x)

else:

return -1

def them(self, x):

self.ds.append(x)

def xoa(self, x):

if x in self.ds:

self.ds.remove(x)

def sua(self, x, y):

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

if self.ds[i] == x:

self.ds[i] = y

**MergeSort:**

class SapXep(Danhsach):

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

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

@staticmethod

def merge\_sort(arr):

n = len(arr)

if n <= 1:

return arr

mid = n // 2

left = arr[:mid]

right = arr[mid:]

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

def merge(self,left,right):

result = []

i = j = 0

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

if left[i] < right[j]:

result.append(left[i])

i += 1

else:

result.append(right[j])

j += 1

result.extend(left[i:])

result.extend(right[j:])

return result

def SapXepMerge(self):

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

**Selection Sort , Insertion Sort , Bubble Sort**

class SapXepN2(Danhsach):

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

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

def SelectionSort(self):

n = len(self.ds)

for i in range(n):

min\_temp = i

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

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

min\_temp = j

if min\_temp != i:

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

def InsertionSort(self):

n = len(self.ds)

k = 1

while k < n and self.ds[k-1] <= self.ds[k]:

k += 1

while k < n:

x = self.ds[k]

pos = k - 1

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

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

pos -= 1

self.ds[pos + 1] = x

k += 1

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

if start == end:

return start

mid = (start + end) // 2

if self.ds[mid] < val:

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

else:

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

def InsertionSort2(self):

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

val = self.ds[i]

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

j = i

while j > pos:

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

j -= 1

self.ds[pos] = val

def BubbleSort(self):

n = len(self.ds)

for i in range(n):

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

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

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

def BubbleSort1(self):

n = len(self.ds)

for i in range(n):

swapped = False

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

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

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

swapped = True

if not swapped:

break

def BubbleSort2(self):

n = len(self.ds)

swapped = True

start = 0

end = n - 1

while swapped == True:

swapped = False

for i in range(start,end):

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

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

swapped = True

if swapped == False:

break

swapped = False

end -= 1

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

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

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

swapped = True

start += 1

**Heap sort:**

def create\_heap(self, n):

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

i = t

j = 2 \* i

while j <= n:

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

j = j + 1

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

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

i = j

j = 2 \* i

else:

break

def heap\_sort(self):

p = len(self.ds) - 1

while p > 0:

self.create\_heap(p)

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

p = p – 1

**Quick sort:**

def PartitionSort(self,first,last):

if first >= last:

return

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

i = first

j = last

while i <= j:

while self.ds[i] < x:

i += 1

while self.ds[j] > x:

j -= 1

if i <= j:

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

i += 1

j -= 1

self.PartitionSort(first,j)

self.PartitionSort(i,last)

def QuickSort(self):

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

**Radix sort:**

def counting\_sort(self, exp1):

n = len(self.ds)

output = [0] \* n

count = [0] \* 10

for i in range(0, n):

index = (self.ds[i] // exp1)

count[(index % 10)] += 1

for i in range(1, 10):

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

i = n - 1

while i >= 0: # Giảm dần while i < n

index = (self.ds[i] // exp1)

output[count[(index % 10)] - 1] = self.ds[i] # Giảm dần: output[n - count[(index % 10)]] = self.ds[i]

count[(index % 10)] -= 1

i -= 1

i = 0

for i in range(0, len(self.ds)):

self.ds[i] = output[i]

def RadixSort(self):

max1 = max(self.ds)

exp = 1

while max1 // exp > 0:

self.counting\_sort(exp)

exp \*= 10

**Bin sort:**

def Bin\_Sort\_DG(self):

B = [None] \* len(self.ds)

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

B[self.ds[i] - 1] = self.ds[i]

self.ds = B

return self.ds

def Bin\_Sort\_TQ(self):

m = max(self.ds)

B = [[] for \_ in range(m+1)] # Tạo m bins

# Phân loại các phần tử vào các bin tương ứng

for i in self.ds:

B[i].append(i) #Giảm dần : B[i].insert(0,num) chèn vào đầu thay vì cuối

# Nối tất cả các bin lại với nhau

self.ds = []

for bin in B: # Giảm dần: for bin in reversed(B)

self.ds += bin

return self.ds

def Bin\_Sort\_TQ1(self):

n = len(self.ds)

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

# Kỳ 1: Phân phối các phần tử vào các bin theo key % n

for i in range(n):

index = self.ds[i] % n

bins[index].append(self.ds[i])

# Kỳ 2: Phân phối các phần tử trong mỗi bin vào các bin mới theo key / n

new\_bins = [[] for \_ in range(n)]

for bin in bins:

for num in bin:

index = num // n

new\_bins[index].append(num)

# Concatenate các bin lại với nhau để tạo ra danh sách được sắp xếp

self.ds = [num for bin in new\_bins for num in bin]

return self.ds

**Tìm kiếm tuyến tính**

class Timkiem(Danhsach):

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

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

def LinearSearch\_Better(self, X):

N = len(self.ds)

self.ds.append(X)

k = 0

while self.ds[k] != X:

k += 1

self.ds.pop()

if k < N:

return k

else:

return -1

def LinearSearch\_DQ(self, X, index=0):

if index == len(self.ds):

return -1

elif self.ds[index] == X:

return index

else:

return self.LinearSearch\_DQ(X, index + 1)

**Binary search:**

class TimKiem(Danhsach):

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

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

def RecBinarySearch(self, X, first, last):

if first > last:

return -1

mid = (first + last) // 2

if X == self.ds[mid]:

return mid

elif X < self.ds[mid]:

return self.RecBinarySearch(X, first, mid - 1)

else:

return self.RecBinarySearch(X, mid + 1, last)

def BinarySearch\_DQ(self, X):

return self.RecBinarySearch(X, 0, len(self.ds) - 1)

def BinarySearch(self, X):

first = 0

last = len(self.ds) - 1

while first <= last:

mid = (first + last) // 2

if X == self.ds[mid]:

return mid

elif X < self.ds[mid]:

last = mid - 1

else:

first = mid + 1

return -1

**BigMult\_ChiaDeTri:**

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

s = Sign(X) \* Sign(Y)

x = abs(X)

y = abs(Y)

if n == 1:

return x \* y \* s

else:

A = Left(x,n//2)

B = Right(x,n//2)

C = Left(y,n//2)

D = Right(y,n//2)

m1 = Big\_int\_mult(A,C,n//2)

m2 = Big\_int\_mult(A-B,D-C,n//2)

m3 = Big\_int\_mult(B,D,n//2)

return (s \* (m1 \* (10\*\*n) + (m1 + m2 + m3)\*(10\*\*(n//2)) + m3))

def Sign(X):

if X > 0:

return 1

elif X < 0:

return -1

else:

return 0

def Left(X,n):

return X // (10\*\*n)

def Right(X,n):

return X % (10\*\*n)

X = 1234

Y = -1234

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

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

print("Kết quả là: ",result)

**ATM:**

**-----Không giới hạn số tờ-----**

def Chon(X,N):

X = sorted(X, reverse=True)

count = 0

total = 0

phuongan =[]

for i in X:

while total <= N:

count +=1

total += i

if total > N:

total -= i

count -= 1

break

phuongan.append((i,count))

count = 0

return phuongan

N = 3450000

X = [500000,200000,100000,50000]

results=Chon(X,N)

for menh\_gia, so\_to in results:

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

**----Giới hạn số tờ là 1-----**

def Chon(X,N):

X = sorted(X, reverse=True)

count = 0

total = 0

so\_to = 1

phuongan = []

for i in X:

while so\_to > 0:

so\_to -= 1

count += 1

total += i

if total > N:

total -= i

count -= 1

phuongan.append((i,count))

count = 0

so\_to = 1

return phuongan

X = [500000,200000,100000,50000]

N = 3450000

results=Chon(X,N)

for menh\_gia, so\_to in results:

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

**-----Số tờ tùy chọn -----**

S = 5

def Chon(X,N):

X = sorted(X, reverse=True)

count = 0

total = 0

so\_to = S

phuongan = []

for i in X:

while so\_to > 0:

so\_to -= 1

count += 1

total += i

if total > N:

total -= i

count -= 1

break

phuongan.append((i,count))

count = 0

so\_to = S

return phuongan

X = [500000,200000,100000,50000]

N = 3450000

results=Chon(X,N)

for menh\_gia, so\_to in results:

**TSP:**

class TSP:

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

self.matrix = matrix

self.n = len(matrix)

def greedy\_tsp(self):

n = len(self.matrix)

path = [0] # starting point

unvisited = list(range(1, n))

city\_names = ['A', 'B', 'C', 'D', 'E', 'F']

while unvisited:

last\_city = path[-1]

next\_city = min(unvisited, key=lambda city: self.matrix[last\_city][city])

path.append(next\_city)

unvisited.remove(next\_city)

total\_length = sum(self.matrix[path[i - 1]][path[i]] for i in range(1, len(path)))

total\_length += self.matrix[path[0]][path[-1]]

path = [city\_names[i] for i in path]

path\_string = "-".join(path) + "-" + path[0]

return path\_string, total\_length

matrix = [

[0, 5, 7.07, 16.55, 15.52, 18], # Khoảng cách từ A đến A, B, C, D, E, F

[5, 0, 5, 11.7, 11.05, 14.32], # Khoảng cách từ B đến A, B, C, D, E, F

[7.07, 5, 0, 14, 14.32, 18.38], # Khoảng cách từ C đến A, B, C, D, E, F

[16.55, 11.7, 14, 0, 3, 7.62], # Khoảng cách từ D đến A, B, C, D, E, F

[15.52, 11.05, 14.32, 3, 0, 5], # Khoảng cách từ E đến A, B, C, D, E, F

[18, 14.32, 18.38, 7.62, 5, 0], # Khoảng cách từ F đến A, B, C, D, E, F

]

tsp = TSP(matrix)

path, length = tsp.greedy\_tsp()

print(f"Chu trình: {path}")

print(f"Tổng độ dài: {length}")

**Cái ba lô:**

**----Không giới hạn-----**

class Do\_vat:

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

self.ten = ten

self.trong\_luong = w

self.gia\_tri = gt

self.don\_gia = gt / w

self.phuong\_an = 0

def greedy(dsdv, w):

dsdv = list(dsdv)

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

for i in range(len(dsdv)):

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

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

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

# Tính tổng trọng lượng và tổng giá trị

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

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

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

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

print("Danh sách đồ vật được chọn:")

for dv in dsdv:

if dv.phuong\_an > 0:

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

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

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

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

w = 37

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

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

**-------Tùy chọn theo số lượng-----**

class Do\_vat:

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

self.ten = ten

self.trong\_luong = w

self.gia\_tri = gt

self.don\_gia = gt / w

self.phuong\_an = 0

self.so\_luong = S

def greedy(dsdv, w):

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

for dv in dsdv:

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

dv.phuong\_an += 1

dv.so\_luong -= 1

w -= dv.trong\_luong

# Tính tổng trọng lượng và tổng giá trị

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

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

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

dsdv = [Do\_vat("A",15,30,2),Do\_vat("B",10,25,3),Do\_vat("C",2,2,4),Do\_vat("D",4,6,5)]

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

print("Danh sách đồ vật được chọn:")

for dv in dsdv:

if dv.phuong\_an > 0:

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

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

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

w = 37

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

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

**-----Mỗi đồ vật chỉ chọn 1---**

class Do\_vat:

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

self.ten = ten

self.trong\_luong = w

self.gia\_tri = gt

self.don\_gia = gt / w

self.phuong\_an = 0

def greedy(dsdv,w):

dsdv = list(dsdv)

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

for i in range(len(dsdv)):

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

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

# Tính tổng trọng lượng và tổng giá trị

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

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

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

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

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

print("Danh sách đồ vật được chọn:")

for dv in dsdv:

if dv.phuong\_an > 0:

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

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

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

w = 37

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

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

**Tổ hợp**

**Bảng:**

def Comb(n,k):

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

C[0][0] = 1

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

C[i][0] = 1

C[i][i] = 1

for j in range(1,i):

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

return C[n][k]

n = 10

k = 3

print(Comb(n,k))

**Vector**

def Comb(n,k):

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

V[0] = 1

V[1] = 1

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

p1 = V[0]

for j in range(1,i):

p2 = V[j]

V[j] = p1 +p2

p1 = p2

V[i] = 1

return V[k]

n = 10

k = 3

print(Comb(n,k))

**Balo Quy Hoạch Động:**

class DoVat:

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

self.ten = ten

self.gt = gt

self.tl = tl

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

def tao\_bang(dsdv, n, W):

F = [[0 for \_ in range(W+1)] for \_ in range(n)]

X = [[0 for \_ in range(W+1)] for \_ in range(n)]

for V in range(dsdv[0].tl, W+1):

X[0][V] = V // dsdv[0].tl

F[0][V] = X[0][V] \* dsdv[0].gt

for k in range(1, n):

for V in range(W+1):

FMax = F[k-1][V]

XMax = 0

yk = V // dsdv[k].tl

#Ba lo 2: yk = min(1,V // dsdv[k].tl)

#Ba lo 3: yk = min(V // dsdv[k].tl,dsdv[k].sl)

for xk in range(1, yk+1):

if F[k-1][V-xk\*dsdv[k].tl] + xk\*dsdv[k].gt > FMax:

FMax = F[k-1][V-xk\*dsdv[k].tl] + xk\*dsdv[k].gt

XMax = xk

F[k][V] = FMax

X[k][V] = XMax

return F, X

def tra\_bang(dsdv, n, W, X):

k = n-1

V = W

while k >= 0:

dsdv[k].so\_dv\_duoc\_chon = X[k][V]

V -= X[k][V] \* dsdv[k].tl

k -= 1

def in\_bang(n, W, F, X):

print("Bảng F và X:")

for k in range(n):

for V in range(W+1):

print(f"{F[k][V]:.1f} {X[k][V]:2d}", end="||")

print()

def in\_kq(dsdv, W):

print("Phương án thu được từ kỹ thuật QUY HOẠCH ĐỘNG như sau:")

tong\_tl = tong\_gt = 0

for dv in dsdv:

if dv.so\_dv\_duoc\_chon > 0:

tong\_tl += dv.so\_dv\_duoc\_chon \* dv.tl

tong\_gt += dv.so\_dv\_duoc\_chon \* dv.gt

print(f"Tổng trọng lượng của ba lô: {tong\_tl}")

print(f"Tổng giá trị của ba lô: {tong\_gt}")

# Khởi tạo danh sách đồ vật

dsdv = [

DoVat("1", 3, 4),

DoVat("2", 5, 5),

DoVat("3", 5, 6),

DoVat("4", 2, 3),

DoVat("5", 1, 1)

]

W = 9 # Khối lượng tối đa của ba lô

n = len(dsdv) # Số lượng đồ vật

F, X = tao\_bang(dsdv, n, W)

tra\_bang(dsdv, n, W, X)

in\_bang(n, W, F, X)

in\_kq(dsdv, W)

**Định trị biểu thức số học:**

class Node:

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

self.val = val

self.right = right

self.left = left

def Dinh\_Tri(root):

if root is None:

return 0

if root.left is None and root.right is None:

return float(root.val)

if root.val == '+':

return Dinh\_Tri(root.left) + Dinh\_Tri(root.right)

elif root.val == '-':

return Dinh\_Tri(root.left) - Dinh\_Tri(root.right)

elif root.val == '\*':

return Dinh\_Tri(root.left) \* Dinh\_Tri(root.right)

elif root.val == '/':

try:

return Dinh\_Tri(root.left) / Dinh\_Tri(root.right)

except ZeroDivisionError:

print("Không thể chia cho 0!")

return 0

else:

return 0

def inorder(root):

if root is not None:

inorder(root.left)

print(root.val, end=' ')

inorder(root.right)

def read\_tree():

x = input("Nhập giá trị cho nút (nhập '0' để bỏ qua): ")

if x == '0':

return None

else:

node = Node(x)

print(f"Nhập con trái của {x}: ")

node.left = read\_tree()

print(f"Nhập con phải của {x}: ")

node.right = read\_tree()

return node

node = read\_tree()

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

inorder(node)

print("\nTrị của biểu thức là: ",Dinh\_Tri(node))

**Bài toán mô phỏng caro**

# Khởi tạo bàn cờ

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

# Vẽ bàn cờ

def draw\_board():

for row in board:

print('|'.join(row))

print('-' \* 7)

# Kiểm tra xem đã có người chiến thắng chưa

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

# Kiểm tra hàng ngang

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

return True

# Kiểm tra hàng dọc

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

return True

# Kiểm tra đường chéo chính

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

return True

# Kiểm tra đường chéo phụ

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

return True

return False

# Chạy trò chơi

def play\_game():

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

while True:

draw\_board()

# Nhập vị trí đặt ký tự

row = int(input("Nhập hàng: "))

col = int(input("Nhập cột: "))

# Kiểm tra vị trí hợp lệ

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

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

continue

# Đặt ký tự vào bàn cờ

board[row][col] = player

# Kiểm tra xem đã có người chiến thắng chưa

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

draw\_board()

print("Người chơi", player, "thắng!")

break

# Chuyển lượt người chơi

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

# Bắt đầu trò chơi

play\_game()

**Cây phủ**

class Graph:

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

self.V = vertices

self.graph = []

self.nodes = {}

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

if u not in self.nodes:

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

if v not in self.nodes:

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

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

def find(self, parent, i):

if parent[i] == i:

return i

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

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

xroot = self.find(parent, x)

yroot = self.find(parent, y)

if rank[xroot] < rank[yroot]:

parent[xroot] = yroot

elif rank[xroot] > rank[yroot]:

parent[yroot] = xroot

else:

parent[yroot] = xroot

rank[xroot] += 1

def local\_search(self):

result = []

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

parent = []

rank = []

for node in range(self.V):

parent.append(node)

rank.append(0)

e = 0

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

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

e += 1

x = self.find(parent, u)

y = self.find(parent, v)

if x != y:

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

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

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

return result

g = Graph(5)

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

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

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

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

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

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

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

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

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

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

results = g.local\_search()

total\_weight = 0

for u, v, w in results:

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

total\_weight += w

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

**Nhập ma trận và tìm phần tử nhỏ nhất**

matrix = [  
 [5,2,7,4],  
 [3,2,-4,8],  
 [9,7,3,5],  
 [6,4,1,-10]  
]  
  
def Min\_CDT(matrix,row\_start,row\_end,column\_start,column\_end):  
 if row\_start == row\_end and column\_start == column\_end:  
 return matrix[row\_start][column\_start]  
  
 row\_mid = (row\_start + row\_end) // 2  
 column\_mid = (column\_start + column\_end) // 2  
 m1= Min\_CDT(matrix,row\_start,row\_mid,column\_start,column\_mid)  
 m2 = Min\_CDT(matrix,row\_start,row\_mid,column\_mid+1,column\_end)  
 m3 = Min\_CDT(matrix,row\_mid+1,row\_end,column\_start,column\_mid)  
 m4 = Min\_CDT(matrix,row\_mid+1,row\_end,column\_mid+1,column\_end)  
 return min(m1,m2,m3,m4)  
n = len(matrix)-1  
a= Min\_CDT(matrix,0,n,0,n)  
print(a)

**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