**N-Queen Problem**

https://youtu.be/Ph95IHmRp5M?si=V78ICgGzX9n6Y9dz  
class Solution:

def solveNQueen(self, n: int) -> list[list[str]]:

cols = set()

negDiag = set() # (r - c)

posDiag = set() # (r + c)

res = []

board = [["."] \* n for \_ in range(n)]

def backtracking(r):

if r == n: # placed queens in all rows

copy = ["".join(row) for row in board]

res.append(copy)

return

for c in range(n):

if c in cols or (r - c) in negDiag or (r + c) in posDiag:

continue

# place queen

cols.add(c)

negDiag.add(r - c)

posDiag.add(r + c)

board[r][c] = "Q"

# go to next row

backtracking(r + 1)

# remove queen (backtrack)

cols.remove(c)

negDiag.remove(r - c)

posDiag.remove(r + c)

board[r][c] = "."

backtracking(0)

return res

# Example run

n = 4

solution = Solution()

output = solution.solveNQueen(n)

for sol in output:

for row in sol:

print(row)

print()

**Tower OF Hanoi**

https://youtu.be/usOtwunz0oM?si=\_lMizFy-IumaQ\_J7  
def tower\_of\_hanoi(n, a, b, c):

if n == 1:

print(f"Move 1st disk from {a} to {c}")

return

tower\_of\_hanoi(n - 1, a, c, b)

print(f"Move {n}th disk from {a} to {c}")

tower\_of\_hanoi(n - 1, b, a, c)

# Example run

tower\_of\_hanoi(2, "A", "B", "C")  
  
  
**Water jug problem**

from collections import defaultdict

jug1,jug2, aim = 4 , 3 ,2

visited = defaultdict(lambda: False)

def waterjugsolver(amt1, amt2):

if(amt1 == aim and amt2== 0 ) or (amt2 == aim and amt1== 0):

print(amt1, amt2)

return True

if visited[(amt1, amt2)] == False:

print(amt1, amt2)

visited[(amt1, amt2)] = True

return(waterjugsolver(amt1, 0) or

waterjugsolver(0, amt2)or

waterjugsolver(jug1, amt2) or

waterjugsolver(amt1, jug2) or

waterjugsolver(amt1+ min(amt2, (jug1 - amt1)), amt2-(min(amt2, (jug1-amt1))))

or waterjugsolver(amt1+ min(amt1, (jug2 - amt2)), amt2-(min(amt1, (jug2-amt2)))))

else:

return False

print("Steps: ")

waterjugsolver(0, 0)  
  
  
**travelling sales man problem**

py -3.10 -m pip install python-tsp  
pip install numpy

import numpy as np

from python\_tsp.exact import solve\_tsp\_dynamic\_programming

# Cost matrix (square matrix)

distance\_matrix = np.array([

[0, 2, 9, 10],

[1, 0, 6, 4],

[15, 7, 0, 8],

[6, 3, 12, 0]

])

# Solve TSP

permutation, distance = solve\_tsp\_dynamic\_programming(distance\_matrix)

print("Shortest Path:", permutation)

print("Total Distance:", distance)

**alpha beta pruning**

maximum, minimum = 1000 , -1000

def fun\_alphabeta(d, node, maxP , v, A, B):

if d==3:

return v[node]

if maxP:

best = minimum

for i in range(0,2):

value = fun\_alphabeta(d+1, node\*2+i, False, v, A, B)

best= max(best, value)

A = max(A, best)

if B<= A:

break

return best

else:

best = maximum

for i in range(0,2):

value = fun\_alphabeta(d+1, node\*2+i, True, v,A,B)

best = min(best, value)

B = min(B, best)

if B<= A:

break

return best

scr=[]

x= int(input("Enter total number of leaf node: "))

for i in range(x):

y=int(input("enter node value: "))

scr.append(y)

d= int(input("Enter dept value: "))

node= int(input("Enter node value: "))

print("the optimal value is :", fun\_alphabeta(d, node, True, scr , minimum, maximum) )

**a\_star**

graph = {

"S": {"A": 5, "B": 4},

"A": {"C": 6, "D": 8},

"B": {"E": 2, "G": 3},

"C": {}, "D": {"G": 4}, "E": {}, "G": {}

}

h = {"S": 10, "A": 8, "B": 2, "C": 4, "D": 5, "E": 6, "G": 0}

def a\_star(start, goal):

open\_list = {start: (h[start], 0)} # {node: (heuristic, g\_cost)}

cost = {start: 0} # best g\_cost found so far

path = {start: [start]} # paths from start

while open\_list:

node = min(open\_list, key=lambda n: sum(open\_list[n]))

h\_val, g\_val = open\_list.pop(node)

if node == goal:

return path[node]

for neigh, w in graph[node].items():

new\_cost = g\_val + w

if neigh not in cost or new\_cost < cost[neigh]:

cost[neigh] = new\_cost

open\_list[neigh] = (h[neigh], new\_cost)

path[neigh] = path[node] + [neigh]

return []

print("Path:", a\_star("S", "G"))

**greedy**

graph={

'F':({'C':8,'D':14},0),'E':({'B':11,'C':11},10),'D':({'A':5,'C':6,'F':14},5),

'C':({'A':12,'E':11,'F':8},5),'B':({'A':10,'E':11},15),'A':({'B':10,'C':12,'D':5},10)

}

def greedy(g,dst,path,q):

for n in g[path[-1]][0]:

if n not in path: q[n]=g[n][1]

if not q: return []

mn=min(q,key=q.get); del q[mn]

return path+[mn] if mn==dst else greedy(g,dst,path+[mn],q)

print("Greedy Path:", "->".join(greedy(graph,"F",["A"],{})) or "Not found")

**travelling salesman n hill climb**  
import random

dist=[[0,2,9,10],[2,0,6,4],[9,6,0,3],[10,4,3,0]]

def cost(tour): return sum(dist[tour[i-1]][tour[i]] for i in range(len(tour)))

def neighbour(t): a,b=random.sample(range(len(t)),2); t[a],t[b]=t[b],t[a]; return t

def hill\_climb():

cur=random.sample(range(4),4); cur\_cost=cost(cur)

for \_ in range(10):

n=neighbour(cur[:]); n\_cost=cost(n)

if n\_cost<cur\_cost: cur,cur\_cost=n,n\_cost

return cur, cur\_cost

t,c=hill\_climb()

print("Best tour:",t,"Cost:",c)

**missionaries n cannebels**

from collections import deque

# Each state: (M\_left, C\_left, Boat\_side, M\_right, C\_right)

# Boat\_side: 0 = left, 1 = right

def is\_valid(m\_left, c\_left, m\_right, c\_right):

# No negative values

if m\_left < 0 or c\_left < 0 or m\_right < 0 or c\_right < 0:

return False

# Missionaries never outnumbered by cannibals

if m\_left > 0 and m\_left < c\_left:

return False

if m\_right > 0 and m\_right < c\_right:

return False

return True

def bfs():

start = (3, 3, 0, 0, 0) # 3 missionaries, 3 cannibals, boat left

goal = (0, 0, 1, 3, 3) # all on right

q = deque()

q.append((start, [start]))

visited = set([start])

moves = [(1,0),(2,0),(0,1),(0,2),(1,1)] # possible boat moves

while q:

(m\_left, c\_left, boat, m\_right, c\_right), path = q.popleft()

if (m\_left, c\_left, boat, m\_right, c\_right) == goal:

return path

for m, c in moves:

if boat == 0: # Boat on left

new\_state = (m\_left - m, c\_left - c, 1, m\_right + m, c\_right + c)

else: # Boat on right

new\_state = (m\_left + m, c\_left + c, 0, m\_right - m, c\_right - c)

if is\_valid(new\_state[0], new\_state[1], new\_state[3], new\_state[4]) and new\_state not in visited:

visited.add(new\_state)

q.append((new\_state, path + [new\_state]))

return None

def print\_solution(solution):

print("\nMissionaries and Cannibals Solution:\n")

for step, (m\_left, c\_left, boat, m\_right, c\_right) in enumerate(solution):

boat\_side = "Left" if boat == 0 else "Right"

print(f"Step {step}:")

print(f" Left Bank : M={m\_left}, C={c\_left} {'<--Boat' if boat==0 else ''}")

print(f" Right Bank: M={m\_right}, C={c\_right} {'<--Boat' if boat==1 else ''}")

print()

# Run BFS

solution = bfs()

if solution:

print\_solution(solution)

else:

print("No solution.")

**number puzzle**

from collections import deque

# Moves for empty tile

def moves(pos):

row, col = divmod(pos, 3)

return [pos-3 if row>0 else None, pos+3 if row<2 else None,

pos-1 if col>0 else None, pos+1 if col<2 else None]

# BFS solver

def bfs(start, goal):

q = deque([(start, [])])

visited = set([start])

while q:

state, path = q.popleft()

if state == goal: return path + [state]

zero = state.index(0)

for m in moves(zero):

if m is not None:

new = list(state)

new[zero], new[m] = new[m], new[zero]

t = tuple(new)

if t not in visited:

visited.add(t)

q.append((t, path + [state]))

return None

# Print nicely

def print\_sol(sol):

for i, s in enumerate(sol):

print(f"Step {i}:")

for j in range(0,9,3): print(s[j:j+3])

print()

# Example

start = (1,2,3,4,0,5,6,7,8)

goal = (1,2,3,4,5,6,7,8,0)

solution = bfs(start, goal)

solution and print\_sol(solution)

**Shuffle the deck of cards**  
  
import random

deck=[f"{r} of {s}" for s in ['hearts','diamonds','clubs','spades'] for r in ['2','3','4','5','6','7','8','9','10','jack','queen','king','ace']]

random.shuffle(deck)

print("Shuffled deck:\n"+"\n".join(deck)  
)

?**TIC TAC TOE**   
board=[' ']\*9

player='X'

def show(): print('\n'.join([f"| {board[i]} | {board[i+1]} | {board[i+2]} |" for i in range(0,9,3)]), end='\n---------\n')

wins=[(0,1,2),(3,4,5),(6,7,8),(0,3,6),(1,4,7),(2,5,8),(0,4,8),(2,4,6)]

def winner(p): return any(all(board[i]==p for i in t) for t in wins)

def tie(): return ' ' not in board

while True:

show()

try:

m=int(input(f'player {player} enter 0-8: '))

if 0<=m<=8 and board[m]==' ':

board[m]=player

if winner(player): show(); print(f'player {player} wins! 🎉'); break

if tie(): show(); print('its a tie! 🤝'); break

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

else: print('Invalid move!')

except: print('Enter 0-8!')

ROCK PAPER SCISSORS

import random

u = input("rock, paper, scissors? ").lower()

c = random.choice(['rock','paper','scissors'])

print(f"You: {u}, Computer: {c}")

if u not in ['rock','paper','scissors']: print("Invalid!")

elif u==c: print("Tie!")

elif (u,c) in [('rock','scissors'),('paper','rock'),('scissors','paper')]: print("You win!")

else: print("Computer wins!")