**Monkey Banana**

def max\_bananas(triangle):

for i in range(len(triangle) - 2, -1, -1):

for j in range(len(triangle[i])):

triangle[i][j] += max(triangle[i+1][j], triangle[i+1][j+1])

return triangle[0][0]

triangle = [

[7],

[3, 8],

[8, 1, 0],

[2, 7, 4, 4],

[4, 5, 2, 6, 5]

]

print(max\_bananas(triangle))

**Alpha Beta Pruning**

def ab(depth, idx, maxp, vals, a, b):

if depth == 3: return vals[idx]

if maxp:

for i in range(2):

a = max(a, ab(depth+1, idx\*2+i, 0, vals, a, b))

if b <= a: break

return a

else:

for i in range(2):

b = min(b, ab(depth+1, idx\*2+i, 1, vals, a, b))

if b <= a: break

return b

vals = [3, 5, 6, 9, 1, 2, 0, -1]

print (ab (0, 0, 1, vals, float('-inf'), float('inf')))

**TSP**

from itertools import permutations

def tsp(graph):

n = len(graph)

cities = range(n)

min\_cost = float('inf')

for path in permutations(cities[1:]):

cost = graph[0][path[0]] + sum(graph[path[i]][path[i+1]] for i in range(n-2)) + graph[path[-1]][0]

min\_cost = min(min\_cost, cost)

return min\_cost

graph = [

[0, 10, 15, 20],

[10, 0, 35, 25],

[15, 35, 0, 30],

[20, 25, 30, 0]

]

print(tsp(graph))

**Tower of Hanoi**

def hanoi(n, source, target, auxiliary):

if n == 1:

print(f"Move disk 1 from {source} to {target}")

return

hanoi(n-1, source, auxiliary, target)

print(f"Move disk {n} from {source} to {target}")

hanoi(n-1, auxiliary, target, source)

# Example: 3 disks

hanoi(3, 'A', 'C', 'B')

**8-Puzzle**

import heapq

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

def manhattan(p): # Heuristic

return sum(abs((v-1)//3 - i) + abs((v-1)%3 - j)

for i,row in enumerate(p)

for j,v in enumerate(row) if v)

def neighbors(p):

i, j = [(i,j) for i in range(3) for j in range(3) if p[i][j] == 0][0]

moves = [(-1,0),(1,0),(0,-1),(0,1)]

for dx,dy in moves:

x,y = i+dx,j+dy

if 0<=x<3 and 0<=y<3:

np = [list(r) for r in p]

np[i][j], np[x][y] = np[x][y], np[i][j]

yield tuple(map(tuple, np))

def solve(start):

heap = [(manhattan(start), 0, start, [])]

seen = set()

while heap:

\_, g, state, path = heapq.heappop(heap)

if state == goal:

return path + [state]

if state in seen: continue

seen.add(state)

for n in neighbors(state):

heapq.heappush(heap, (g+1+manhattan(n), g+1, n, path + [state]))

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

for step in solve(start):

for row in step: print(row)

    print()