- *Depth first search*

```
graph = {'5': ['3', '7'], '3': ['2', '4'], '7': ['8'], '2': [], '4': ['8'], '8': []}
visited = set()
def dfs(visted, graph, node):
  if node not in visited:
    print(node)
    visted.add(node)
    for neighbour in graph[node]:
      dfs(visted, graph, neighbour)
print("Following is the Depth-First Search")
dfs(visited,graph,'5')
```

- *Breadth first search*

```
graph = {'5': ['3', '7'], '3': ['2', '4'], '7': ['8'], '2': [], '4': ['8'], '8': []}
visited = []
queue = []
def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)
  while queue:
    m = queue.pop(0)
    print(m, end="  ")
    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)
print("Following is the Breadth-First Search")
bfs(visited,graph,'5')
```

- *Towers of Honai*

```
def TowerOfHanoi(n, from_rod, to_rod, aux_rod):
  if n == 1:
```

```
        print("Move disk 1 from rod ", from_rod, "to rod", to_rod)
        return
    TowerOfHanoi(n - 1, from_rod, aux_rod, to_rod)
    print("Move disk ", n, "from rod ", from_rod, "to rod", to_rod)
    TowerOfHanoi(n - 1, aux_rod, to_rod, from_rod)
n = int(input("Enter the number : "))
TowerOfHanoi(n, 'A', 'C', 'B')
```

- *Monkey Banana problema*

```
def move(box_position, action):
    if action == 'left':
        return max(0, box_position - 1)
    elif action == 'right':
        return min(4, box_position + 1)
    else:
        return box_position
def monkey_banana():
    print("Monkey Banana Problem:")
    print("The monkey needs to reach the bananas using a box.")
    box_position = 2
    monkey_position = 0
    while monkey_position != 4:
        print(f"The monkey is at position {monkey_position}.")
        print(f"The box is at position {box_position}.")
        action = input("Enter your action ('left', 'right', 'up', 'down'): ")
        box_position = move(box_position, action)
        if monkey_position < box_position:
```

```
        monkey_position += 1
    elif monkey_position > box_position:
        monkey_position -= 1
    if monkey_position == 4:
        print("Congratulations! The monkey reached the bananas.")
    else:
        print("The monkey is still trying to reach the bananas.")
monkey_banana()
```

- *Travelling salesman problem*

```
from sys import maxsize
from itertools import permutations
V = 4
def travellingSalesManProblem(graph, s):
  vertex = []
  for i in range(V):
    if i != s:
      vertex.append(i)
  min_path = maxsize
  next_permutation = permutations(vertex)
  for i in next_permutation:
    current_pathweight = 0
    k = s
    for j in i:
      current_pathweight += graph[k][j]
      k = j
    current_pathweight += graph[k][s]
```

```
        min_path = min(min_path, current_pathweight)
    return min_path
if __name__ == "__main__":
    graph = [[0, 10, 15, 20], [10, 0, 35, 25],
             [15, 35, 0, 30], [20, 25, 30, 0]]
    s = 0
    print(travellingSalesManProblem(graph, s))
```