DFS

```
graph = {
   'A': ['B', 'C'],
   'B': ['A', 'D', 'E'],
   'C': ['A', 'F'],
   'D': ['B'],
   'E': ['B'],
   'F': ['C']
}

def dfs(v, visited=set()):
   if v not in visited:
      print(v, end=' ')
      visited.add(v)
      for n in graph[v]:
         dfs(n, visited)

dfs('A')
```

BFS

```
from collections import deque

graph = {
   'A': ['B', 'C'],
   'B': ['A', 'D', 'E'],
   'C': ['A', 'F'],
   'D': ['B'],
   'E': ['B'],
   'F': ['C']
}

def bfs(start):
   visited = set([start])
   q = deque([start])
   while q:
      v = q.popleft()
      print(v, end=' ')
      for n in graph[v]:
         if n not in visited:
            visited.add(n)
            q.append(n)
```

```
bfs('A')
```

A* algorithm

```
import heapq

def astar(s, g, grid):
    H = lambda a: abs(a[0]-g[0]) + abs(a[1]-g[1])
    open = [(H(s), 0, s)]
    seen = set()
    while open:
        _, cost, pos = heapq.heappop(open)
        if pos == g: return cost
        if pos in seen: continue
        seen.add(pos)
        for d in [(0,1),(1,0),(-1,0),(0,-1)]:
            x, y = pos[0]+d[0], pos[1]+d[1]
            if 0<=x<len(grid) and 0<=y<len(grid[0]) and not grid[x][y]:
                heapq.heappush(open, (cost+1+H((x,y)), cost+1, (x,y)))

grid = [[0,1,0],[0,0,0],[1,0,0]]
print(astar((0,0), (2,2), grid))
```

Selection sort

```
def selection_sort(arr):
    for i in range(len(arr)):
        m = i
        for j in range(i+1, len(arr)):
            if arr[j] < arr[m]: m = j
        arr[i], arr[m] = arr[m], arr[i]
    return arr

print(selection_sort([5, 2, 8, 1, 3]))
```

Prims algorithm

```
import heapq

def prim(graph, start):
    visited, mst, heap = set(), [], [(0, start, None)]
    while heap:
        cost, u, prev = heapq.heappop(heap)
        if u in visited: continue
```

```
        visited.add(u)
        if prev is not None: mst.append((prev, u, cost))
        for v, w in graph[u]:
            if v not in visited:
                heapq.heappush(heap, (w, v, u))
    return mst

graph = {
    'A': [('B', 1), ('C', 3)],
    'B': [('A', 1), ('C', 1), ('D', 6)],
    'C': [('A', 3), ('B', 1), ('D', 4)],
    'D': [('B', 6), ('C', 4)]
}

print(prim(graph, 'A'))
```

Graph coloring

```
def is_safe(v, graph, color, c):
    return all(color[i] != c for i in graph[v])

def graph_coloring(graph, m):
    color = [0] * len(graph)
    def solve(v=0):
        if v == len(graph): return color
        for c in range(1, m+1):
            if is_safe(v, graph, color, c):
                color[v] = c
                if solve(v + 1): return color
                color[v] = 0
        return None
    return solve()

# Example graph represented as adjacency list (0-indexed)
graph = [[1, 2], [0, 2], [0, 1]]

# Try to color the graph with 3 colors
result = graph_coloring(graph, 3)
print("Coloring solution:", result)
```

Chatbot

```
def chatbot():
    print("Hello! How can I assist you today?")
```

```
    while True:
        user_input = input("You: ").lower()

        if "hours" in user_input:
            print("Chatbot: We are open from 9 AM to 9 PM.")
        elif "order" in user_input and "status" in user_input:
            print("Chatbot: Please provide your order number to check the status.")
        elif "product" in user_input:
            print("Chatbot: We have a variety of products. What are you looking for?")
        elif "bye" in user_input:
            print("Chatbot: Goodbye! Have a great day.")
            break
        else:
            print("Chatbot: Sorry, I didn't understand that. Can you rephrase?")

chatbot()
```

Expert system

```
def diagnose_patient():
    print("Welcome to the Health Diagnosis Expert System")

    # Collecting symptoms
    fever = input("Do you have a fever? (yes/no): ").lower()
    cough = input("Do you have a cough? (yes/no): ").lower()
    fatigue = input("Are you feeling fatigued? (yes/no): ").lower()

    # Diagnosing based on symptoms
    if fever == "yes" and cough == "yes" and fatigue == "yes":
        diagnosis = "You may have the flu. Please consult a doctor."
    elif fever == "yes" and cough == "yes":
        diagnosis = "You may have a cold or flu. Please monitor your symptoms."
    elif fatigue == "yes" and cough == "no" and fever == "no":
        diagnosis = "You may be stressed or overworked. Rest well."
    else:
        diagnosis = "Symptoms are not conclusive. Please seek medical advice."

    print(f"Diagnosis: {diagnosis}")

# Run the expert system
diagnose_patient()
```