```
#Prim's Algorithm
import heap
def make_graph():
    # tuple = (cost, n1, n2)
    return {
        'A': [(3, 'D', 'A'), (4, 'B', 'A'), (5, 'E', 'A')],
        'B': [(4, 'A', 'B'), (2, 'C', 'B')],
        'C': [(2, 'B', 'C'),(1, 'D', 'C')],
        'D': [(3, 'A', 'D'), (1, 'C', 'D')],
        'E': [(5, 'A', 'E')],
    }

def prims(G, start='A'):
    unvisited = list(G.keys())
    visited = []
    total_cost = 0
    MST = []
    unvisited.remove(start)
    visited.append(start)
    heap = G[start]
    heapq.heapify(heap)
    while unvisited:
        (cost, n2, n1) = heapq.heappop(heap)
        new_node = None
        if n1 in unvisited and n2 in visited:
            new_node = n1
            MST.append((n2, n1, cost))
        elif n1 in visited and n2 in unvisited:
            new_node = n2
            MST.append((n1, n2, cost))
        if new_node != None:
            unvisited.remove(new_node)
            visited.append(new_node)
            total_cost += cost

            for node in G[new_node]:
                heapq.heappush(heap, node)

    return MST, total_cost



def main():
    G = make_graph()
    MST, total_cost = prims(G, 'A')

    print(f'Minimum spanning tree: {MST}')
    print(f'Total cost: {total_cost}')

main()
```

OutPut :

```
Minimum spanning tree: [('A', 'D', 3), ('D', 'C', 1), ('C', 'B', 2), ('A', 'E', 5)]
Total cost: 11
```

```
#N-Queens problem
def main():
    N = int(input("Enter No of Queens: "))
    board = [[0 for _ in range(N)] for _ in range(N)]
    for i in range(N):
        for j in range(N):
            board[i][j] = 0
    if helper(board, 0, N):
        print_board(board, N)
    else:
        print("Solution does not exist")


def helper(board, col, N):
    if col >= N:
        return True
    for i in range(N):
        if safe(board, col, i, N):
            board[i][col] = 1
            if helper(board, col + 1, N):
                return True
            board[i][col] = 0
    return False


def safe(board, col, row, N):
    for i in range(col):
        if board[row][i] == 1:
            return False
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False
    for i, j in zip(range(row, N), range(col, -1, -1)):
        if board[i][j] == 1:
            return False
    return True


def print_board(board, N):
    for i in range(N):
        for j in range(N):
            if board[i][j] == 1:
                print(" Q ", end=" ")
            else:
                print(" _ ", end=" ")
        print()
if __name__ == "__main__":
    main()
```

Output :

```
Enter No of Queens: 7
 Q  _  _  _  _  _  _
 _  _  _  _  Q  _  _
 _  Q  _  _  _  _  _
 _  _  _  _  _  Q  _
 _  _  Q  _  _  _  _
 _  _  _  _  _  _  Q
 _  _  _  Q  _  _  _
```

```
#Chat-Bot System
def chat_bot_system():
   print("Enter Your Name : ", end="")
   name = input()
   print("Hello", name, "Welcome to AASWAD-Restaurent\n")
   print("What would you like to order", name, "\n")
   menu_options = ["Rice-Plate", "Samosa", "Vada-Pav", "Chole-Bhature", "Pohe"]
   q_count = [0] * len(menu_options)

   while True:
      for i, option in enumerate(menu_options):
         print("Option", i + 1, ":", option)
      print("\nI would like to have option : ", end="")
      opt = int(input()) - 1
      if opt >= len(menu_options):
         print("Display relevant query")
         continue
      print("\nYou Confirm order :", menu_options[opt])
      q_count[opt] += 1
      if q_count[opt] >= 5:
         break
      order = input("Do you want anything else (yes/no): ").strip().upper()
      print()
      if order == "YES":
         continue
      else:
         break
   your_order(menu_options, q_count)
   print("\nYour total bill is", total_bill(q_count))
   print("\nThanks for your order!")
```

```
def total_bill(q_count):
    ans = 0
    prize = [50, 25, 25, 55, 25]
    for i in range(len(q_count)):
        ans += q_count[i] * prize[i]
    return ans

def your_order(menu_options, q_count):
    print("Your Order is : ")
    for i in range(len(q_count)):
        if q_count[i] > 0:
            print(menu_options[i], q_count[i])

def main():
    chat_bot_system()
if __name__ == "__main__":
    main()
```

Output :

```
Enter Your Name : Chaitanya
Hello Chaitanya Welcome to AASWAD-Restaurent

What would you like to order Chaitanya

Option 1 : Rice-Plate
Option 2 : Samosa
Option 3 : Vada-Pav
Option 4 : Chole-Bhature
Option 5 : Pohe
```

```
I would like to have option : 3


You Confirm order : Vada-Pav

Do you want anything else (yes/no): yes


Option 1 : Rice-Plate

Option 2 : Samosa

Option 3 : Vada-Pav

Option 4 : Chole-Bhature

Option 5 : Pohe


I would like to have option : 2


You Confirm order : Samosa

Do you want anything else (yes/no): no


Your Order is :

Samosa 1

Vada-Pav 1


Your total bill is 50


Thanks for your order!
```

```
#ExpertSystem

def main():
    print("Welcome to the Stock Market Trading System!")
    print("Please answer the following questions:")
    trend = ask_question("What is the current market trend? (Upwards/Downwards): ")
    fundamentals = ask_question("How are the fundamentals of the company? (strong/weak): ")
    indicators = ask_question("What do the technical indicators suggest? (positive/negative): ")
    should_buy = evaluate(trend, fundamentals, indicators)
    print_result(should_buy)

def ask_question(question):
    return input(question).strip()

def evaluate(trend, fundamentals, indicators):
    return trend.lower() == "upwards" and fundamentals.lower() == "strong" and indicators.lower() == "positive"

def print_result(should_buy):
    if should_buy:
        print("Recommendation: Buy the stock!")
    else:
        print("Recommendation: Do not buy the stock.")
if __name__ == "__main__":
    main()
```

Output :

```
Welcome to the Stock Market Trading System!
Please answer the following questions:
What is the current market trend? (Upwards/Downwards): upwards
How are the fundamentals of the company? (strong/weak): strong
```

What do the technical indicators suggest? (positive/negative): positive

Recommendation: Buy the stock!