STOP AND WAIT

```
import time
import random
print("Enter the number of packets:")
a=int(input())
c=1
def send(a):
    for i in range(a):
        print("Packet",i+1,"sent")
        time.sleep(2)
        x=random.randint(0,2)
        if x==1:
            print("Packet Received")
            print("Received-Ack",i+2)
        elif x==0:
            time.sleep(2)
            print("Ack lost")
            print("Send packet",i+1)
            print("Packet Received")
            print("Packet",i+1,"already present so discard packet",i+1)
            print("Received-Ack",i+2)
        elif x==2:
            print("Packet lost")
            print("No Ack received")
            print("Send packet",i+1)
            print("Packet Received")
            print("Received-Ack",i+2)
send(a)
```

```
##GO BACK N
import random
a=list(map(int,input()))
print("Enter window size:")
size=int(input())
def send(a,size):
    x=random.randint(0,1)
    if x==1:
        for i in range(size):
            print("Send packet",a[i])
            if i==3:
                print("ACK",a[i]+1,"Received")
        for i in range(size):
            a.pop(0)
    elif x==0:
        for i in range(3):
            print("Time:",i)
        print("Timer timed out")
        print("No ACK received")
        print("Retransmit the packets")
        for i in range(size):
            print("Send packet",a[i])
            if i==3:
                print("ACK",a[i]+1,"Received")
        for i in range(size):
                a.pop(0)
    if len(a)!=0:
        send(a,size)
send(a,size)
```

DIJKSTRA

```
import heapq

def dijkstra(graph, start):
    vertices = len(graph)

    distance = [float('inf')] * vertices
    distance[start] = 0

    priority_queue = [(0, start)]

    while priority_queue:
        current_distance, current_vertex = heapq.heappop(priority_queue)
        if current_distance > distance[current_vertex]:
            continue
        for v in range(vertices):
            if graph[current_vertex][v] > 0:
                new_distance = distance[current_vertex] + graph[current_vertex][v]
                if new_distance < distance[v]:
                    distance[v] = new_distance
                    heapq.heappush(priority_queue, (new_distance, v))
    return distance
graph = [
    [0, 4, 0, 0, 0, 0, 0, 8, 0],
    [4, 0, 8, 0, 0, 0, 0, 11, 0],
    [0, 8, 0, 7, 0, 4, 0, 0, 2],
    [0, 0, 7, 0, 9, 14, 0, 0, 0],
    [0, 0, 0, 9, 0, 10, 0, 0, 0],
    [0, 0, 4, 14, 10, 0, 2, 0, 0],
    [0, 0, 0, 0, 0, 2, 0, 1, 6],
    [8, 11, 0, 0, 0, 0, 1, 0, 7],
    [0, 0, 2, 0, 0, 0, 6, 7, 0]
]

start_vertex = 0
result = dijkstra(graph, start_vertex)

print("Shortest distances from vertex {}:".format(start_vertex))
for i, distance in enumerate(result):
    print("Vertex {}: {}".format(i, distance))
```

DISTANCE VECTOR

```
class DistanceVectorRouter:
    def __init__(self, router_id, neighbors):
        self.router_id = router_id
        self.neighbors = neighbors
        self.routing_table = {neighbor: float('inf') for neighbor in neighbors}
        self.routing_table[router_id] = 0

    def update_routing_table(self, neighbor, cost):
        if cost < self.routing_table[neighbor]:
            self.routing_table[neighbor] = cost
            return True
        return False
    def send_routing_table(self):
        return self.routing_table.copy()

    def receive_routing_table(self, neighbor_id, neighbor_routing_table):
        updated = False
        for destination, cost in neighbor_routing_table.items():
            total_cost = cost + 1
            updated |= self.update_routing_table(destination, total_cost)
        return updated

def simulate_distance_vector(routers, max_iterations=1):
    for _ in range(max_iterations):
        for router_id, router in routers.items():
            for neighbor_id in router.neighbors:
                neighbor_table = routers[neighbor_id].send_routing_table()
                if router.receive_routing_table(neighbor_id, neighbor_table):
                    print(f"Router {router.router_id} updated its routing table based on Router
{neighbor_id}'s table:")
                    print(router.routing_table)
                    print()

# Example usage
routers = {
    1: DistanceVectorRouter(1, neighbors=[2, 3]),
    2: DistanceVectorRouter(2, neighbors=[1, 3]),
    3: DistanceVectorRouter(3, neighbors=[1, 2])
}

simulate_distance_vector(routers)
```

HAMMING CODE

```
print("Enter the data bits:")
data=input()
print("Initial Data:D7 D6 D5 P4 D3 P2 P1")
print("Next string",data[0],data[1],data[2],"P4",data[3],"P2 P1")
d7=data[0]
d6=data[1]
d5=data[2]
d3=data[3]
#p1d3d5d7
#p2d3d6d7
#p4d5d6d7
t1=d3+d5+d7
t2=d3+d6+d7
```

```
t4=d5+d6+d7
a=0
b=0
c=0
for i in t1:
    if i=="1":
        a=a+1
if a%2==0:
    p1="0"
else:
    p1="1"
for i in t2:
    if i=="1":
        b=b+1
if b%2==0:
    p2="0"
else:
    p2="1"
for i in t4:
    if i=="1":
        c=c+1
if c%2==0:
    p4="0"
else:
    p4="1"
fs=d7+d6+d5+p4+d3+p2+p1
print("Final string:",fs)
```

CRC

```
a=input("Enter data bits:")
b=input("Enter divisor bits:")
c=len(b)
e=a
f=a
for i in range(c-1):
    f=f+'0'
print("Divident:",f)
def xori(x,y):
    d=[]
    for i in range(1,c):
        if x[i]==y[i]:
            d.append('0')
        else:
            d.append('1')
    return ''.join(d)
for i in range(c-1):
    d=''+xori(a,b)
    d=d+'0'
    a=d
    d=xori(a,b)
x=e+d
print("Data sent to receiver:",x)
for i in range(c-1):
    y=''+xori(x,b)
    y=y+d[i]
    x=y
    y=xori(x,b)
print("Remainder",y)
g=0
for i in range(c-1):
    if y[i]=='0':
        g=g+1
    else:
        continue
if g==c-1:
    print("Received data has no error")
else:
    print("Received data has error")
```