Subiect 1

Presupunem ca A, B si C sunt variabile aleatoare, care iau valorile date de zarurile respective cu o probabilitate de 1/6.

a.

```
P(A > B) = 1/6 * P(1 > B) + 1/6 * P(2 > B) + 1/6 * P(5 > B) + 1/6 * P(6 >
B) + 1/6 * P(7 > B) + 1/6 * P(9 > B)
= 1/6 * 0 + 1/6 * 1/6 + 1/6 * 1/2 + 1/6 * 4/6 + 1/6 * 4/6 + 1/6 * 5/6
= 1/6 (1/6 + 3/6 + 4/6 + 4/6 + 5/6) =
= 1/36 * (1 + 3 + 4 + 4 + 5) = 17/36
```

Cu aceeasi formula, putem calcula pe `P(B > C)` si `P(C > A)`.

Totusi, fiind simple calcule matematice, putem executa urmatoarea secventa de cod de Python:

```
A = [1, 2, 5, 6, 7, 9]
B = [1, 3, 4, 5, 8, 9]
C = [2, 3, 4, 6, 7, 8]

nr_a_mai_mare_b = []
nr_b_mai_mare_c = []
nr_c_mai_mare_a = []

for i in A:
    for j in B:
        if i > j:
            nr_a_mai_mare_b.append((i, j))


for i in B:
    for j in C:
        if i > j:
            nr_b_mai_mare_c.append((i, j))


for i in C:
    for j in A:
        if i > j:
            nr_c_mai_mare_a.append((i, j))

print(f"Perechile cu A > B: {nr_a_mai_mare_b}")
print(f"P(A > B) = {len(nr_a_mai_mare_b)}/36 = {len(nr_a_mai_mare_b) /
36}")

print(f"Perechile cu B > C: {nr_b_mai_mare_c}")
print(f"P(B > C) = {len(nr_b_mai_mare_c)}/36 = {len(nr_b_mai_mare_c) /
```

```
36}")

print(f"Perechile cu C > A: {nr_c_mai_mare_a}")
print(f"P(C > A) = {len(nr_c_mai_mare_a)}/36 = {len(nr_c_mai_mare_a) /
36}")
```

Aceasta ne da atat rezultatul, cat si o motivare a acestuia:

```
Perechile cu A > B: [(2, 1), (5, 1), (5, 3), (5, 4), (6, 1), (6, 3), (6,
4), (6, 5), (7, 1), (7, 3), (7, 4), (7, 5), (9, 1), (9, 3), (9, 4), (9, 5),
(9, 8)]
P(A > B) = 17/36 = 0.4722222222222222
Perechile cu B > C: [(3, 2), (4, 2), (4, 3), (5, 2), (5, 3), (5, 4), (8,
2), (8, 3), (8, 4), (8, 6), (8, 7), (9, 2), (9, 3), (9, 4), (9, 6), (9, 7),
(9, 8)]
P(B > C) = 17/36 = 0.4722222222222222
Perechile cu C > A: [(2, 1), (3, 1), (3, 2), (4, 1), (4, 2), (6, 1), (6,
2), (6, 5), (7, 1), (7, 2), (7, 5), (7, 6), (8, 1), (8, 2), (8, 5), (8, 6),
(8, 7)]
P(C > A) = 17/36 = 0.4722222222222222
```

Scriptul da inapoi submultimile din produsele carteziene care respecta cerintele. Cum toate elementele din produsurile carteziene au aceeasi probabilitatea, fractia acestora reprezinta fix probabilitatea evenimentului.

Asadar, `P(A > B) = P(B > C) = P(C > A) = 17/36`.

b.

Simliar ca la a, voi face de mana calculul `P(A = B)`, dupa care vom automatiza procesul.

Perechiile `(a, b)` din `A x B` a.i. `a=b` sunt:

`(1, 1)`, `(5, 5)` si `(9, 9)`.

Avem asadar `3` elemente favorabile din produsul cartezian de `36` de elemente.

Probabiliatea lui `P(A = B)` este asadar `3/36 = 1/12`.

Pentru a automatiza procesul avem urmatorul script de Python:

```
A = [1, 2, 5, 6, 7, 9]
B = [1, 3, 4, 5, 8, 9]
C = [2, 3, 4, 6, 7, 8]

a_egal_b = []
b_egal_c = []
c_egal_a = []

for i in A:
```

```
    for j in B:
        if i == j:
            a_egal_b.append((i, j))


for i in B:
    for j in C:
        if i == j:
            b_egal_c.append((i, j))


for i in C:
    for j in A:
        if i == j:
            c_egal_a.append((i, j))

print(f"Perechile cu A = B: {a_egal_b}")
print(f"P(A = B) = {len(a_egal_b)}/36 = {len(a_egal_b) / 36}")

print(f"Perechile cu B = C: {b_egal_c}")
print(f"P(B = C) = {len(b_egal_c)}/36 = {len(b_egal_c) / 36}")

print(f"Perechile cu C = A: {c_egal_a}")
print(f"P(C = A) = {len(c_egal_a)}/36 = {len(c_egal_a) / 36}")
```

Rezultatul scriptului este:

```
Perechile cu A = B: [(1, 1), (5, 5), (9, 9)]
P(A = B) = 3/36 = 0.08333333333333333
Perechile cu B = C: [(3, 3), (4, 4), (8, 8)]
P(B = C) = 3/36 = 0.08333333333333333
Perechile cu C = A: [(2, 2), (6, 6), (7, 7)]
P(C = A) = 3/36 = 0.08333333333333333
```

Avem asadar `P(A = B) = P(B = C) = P(C = A) = 1/12`.

Subiect 2

Observam ca `Enc_k(k)` este egal fix cu `2*k`.

a.

Elementele din inelele `Z2t+1` sunt resturi modulare ale lui `Z` modulo `(2t+1)`.

In mod evident, `2` este prim cu `2t+1`, `2t+1` fiind impar.

Asadar, din formula lui Euclid Extinsa (Extended GCD), stim ca exista `x` si `y` a.i.

`2 * x + (2t+1) * y = 1`.

Daca reducem ecuatia modulo `2t+1`, obtinem exact ceea ce ne dorim:

`2 * x === 1 (mod 2t+1)`.

Altfel spus, X este inversul nostru modular.

Putem chiar sa punem mana pe `x`: `2 * (t+1) === 1 (mod 2t+1)`, deci `x = t+1`.

Practic, `2` face parte din multimea elementelor inversabile ale grupului multiplicativ `(Z2t+1, *)`, fiind prim cu modulul. qed.

Pentru a arata ca `2` nu este inversabil in grupul `(Z2t, *)`, putem usor gasi un contra exemplu:

`2 * t === 0 (mod 2t)`.

Daca ar exista vreun invers modular al lui `2`, pe care il notam cu `x`, am putea inmulti la stanga cu `x` egalitatea de mai sus:

```
x * 2 * t = x * 0 <=>
(x * 2) * t = x * 0 <=>
t = 0
```

Dar din ipoteza, `t > 1`. Contradictie.

Asadar, `2` este multiplicativ inversabil in inelele `Z2t+1`, dar nu in inelele `Z2t`.

b.

Asa cum am explicat la punctul a, inversul modular al lui 2 poate fi obtinut in doua moduri:

- Prin euclid extins - functioneaza pentru inversul oricarui numar.
- Setand inversul modular ca fiind `(modul + 1) / 2` - functioneaza numai in cazul lui 2.

In cadrul subpunctului, stim ca m=2t+1. Asadar, inversul lui 2 in raport cu m este t+1.

Intradevar, `2 * (t+1) === 2*t+2 === 1 (modulo m)`.

Asadar, dandu-se `2*k`, il putem calcula pe `k` ca fiind `2^-1 * (2*k) = (t+1) * (2 * k)`.

Cum `Enc_k(k) = k + k = 2 * k`, avem algoritmul urmator (scris cu o sintaxa de Python pentru simplitate):

```python
def break_cypher(enc_k_k, m):
    """
    Input:
        * encodarea cu cheia K al lui k
        * modulul m, unde m este impar
    Output:
        Cheia k
    """
    # inversul lui 2.
```

```
    # il putem calcula si cu euclid extins daca ne dorim
    inv_modular_2 = (m + 1) // 2

    # inmultim toate componentele sirului enc_k_k cu inversul lui 2, modulo
m
    k = [inv_modular_2 * x % m for x in enc_k_k]

    return k
```

C.

Pentru a rezolva punctul C, vom fi putin mai tehnici: Cand un numar `X`, cu `0 <= X < M`, `M` par, este inmultit cu 2, acesta are probabilitatea de `1/2` sa ramana in continuare mai mic decat M: Intervalul `[0, M/2)` se va duce in elementele pare din `[0, M)`, si `[M/2, M)` se va duce in `[M, 2*M)`.

Daca avem norocul ca `X` sa se afle in `[0, M/2)`, atunci inmultirea cu 2 se efectueaza similar in `Z` si in `Zm`, si inversul inmultirii cu `2` este impartirea pe `Z` al lui `2*X`. Acest lucru se intampla in mod evident cu probabilitatea de `1/2`.

Asadar, afirmam ca urmatorul pseudocod calculeaza cheia `K` cu o probabilitate de `1/2^n`, unde `n` este lungimea sirului `k`:

```
def break_cypher_probabilistic(enc_k_k, m):
    """
    Input:
        * encodarea cu cheia K al lui k
        * modulul m, unde m este par
    Output:
        Cheia k - corecta cu o probabilitate de 1/2^n
    """

    k = [x // 2 for x in enc_k_k]
    return k
```

Demonstratie

Toate elementele sunt pare, fiind ca `enc_k_k = Enc_k(k)` este `2*k`, in `Z2t`. Le impartim asadar ca numere intregi, ceea ce este corect cu o probabilitate de `1/2` pentru fiecare pozitie. Cum avem N pozitii in total, probabilitatea ca toate pozitiile sa se imparta corect este de `1/2^N`.

Totusi, daca cheia este gandita inteligent, este posibil ca toate cheile generate sa aiba cel putin un element mai mare de `M/2`, pentru a impiedica acest tip de atac.

Daca `X >= M/2`, observam ca `2 * X (modulo M) = 2 * X - M`, deci putem de asemenea inversa inmultirea cu 2.

Propunem asadar un alt algoritm, care alege pentru fiecare element din sir in mod randomizat in ce interval se afla X:

```python
def break_cypher_probabilistic(enc_k_k, m):
    """
    Input:
        * encodarea cu cheia K al lui k
        * modulul m, unde m este par
    Output:
        Cheia k - corecta cu o probabilitate de 1/2^n
    """

    k = [x // 2 if random() % 2 == 0 else (x + m) // 2 for x in enc_k_k]
    return k
```

Observam ca linia `k = [x // 2 if random() % 2 == 0 else (x + m) // 2 for x in enc_k_k]` alege `X/2` cu probabilitatea de `1/2`, si `(X+m)/2` cu probabilitatea tot de `1/2`. Asadar, face alegerea buna cu o probabilitate de `1/2`.

Asadar, algoritmul are sansa de `1/2^n` sa sparga cypher-ul, dandu-se `Enc_k(K)`, indiferent de masurile luate de aparatori pentru a impiedica acest tip de atac.