Diskrete Mathematik - Klausurzettel

```python
# returns gcd, x, y s.t. a*x + b*y = gcd(a,b)
def extended_gcd(a, b):
    x, y, lastx, lasty = 0, 1, 1, 0
    while b != 0:
        quo, a, b = a // b, b, a % b
        x, lastx, y, lasty = lastx - quo * x, x, lasty - quo * y, y
    return a, lastx, lasty

# returns x s.t. x = a_i mod m_i for all i
def crt(a : list, m : list):
    M, s = prod(m), 0
    for a_i, m_i in zip(a, m):
        M_i = M // m_i
        e_i = M_i * extended_gcd(m_i, M_i)[2]
        s += a_i * e_i
    return s % M

# returns a^e mod m
def binpowmod(a, e, m):
    p, q = 1, a
    while e > 0:
        if e % 2 == 1:
            p = p * q % m
        q = q * q % m
        e //= 2
    return p

# returns S = T * R^-1 mod N
# requires N_prime s.t. N * N_prime = -1 mod R
def REDC(T, N_prime, R, N):
    t = (T + T % R * N_prime % R * N) / R
    return t - N if T >= N else t

# returns factorization of u
from galois import *
import numpy as np
def berlekamp(u : Poly, randomized : bool = True) -> list[tuple[Poly, int]]:
    Fp, p, n = u.field, u.field.order, u.degree
    if n < 2: return [(u, 1)]
    u_inp, u = u, u // gcd(u, u.derivative())
    Q = Fp([(lambda a,s : np.pad(a,(s-len(a),0)))( # unimportant python part
        (Poly([1,0], Fp) ** (i*p) % u).coeffs, u.degree)[::-1]
        for i in range(u.degree)])
    V = (Q - Fp.Identity(u.degree)).T.null_space().T[::-1].T
    u_fact = (u)
    if randomized:
        while len(u_fact) < len(V):
            for factor in u_fact:
                v = Poly(np.sum([(np.random.randint(p)*v) for v in V],axis=0)%p,Fp)
                g = gcd(factor, v)
                if g == 1: g = gcd(factor, v**((p-1)//2) + Fp(1))
                if g.degree < factor.degree and g != 1:
                    new_facts = (g, factor // g)
                    u_fact.remove(factor)
                    u_fact |= new_facts
                    break
    else:
```

```
        for v in V[1:]:
            if len(u_fact) >= len(V): break
            for factor in u_fact:
                gen = (gcd(Poly(v, Fp)-Fp(i), u) for i in range(u.degree))
                factor_fact = (g for g in gen if g != 1)
                if len(factor_fact) > 0:
                    u_fact |= factor_fact
                    break
        u_fact.remove(u)
    ret = []
    for factor in u_fact:
        e = 0
        while u_inp % factor == 0:
            e += 1
            u_inp //= factor
        ret.append((factor, e))
    return ret
```

```
def null_space(A : list[list[Fp]]) -> list:
    c = [-1] * len(A)
    ret = []
    for k in range(len(A)):
        for j in range(len(A)):
            if A[k][j] != 0 and c[j] == -1:
                for i in range(len(A)): A[i][j] *= Fp(-1) // A[k][j]
                for i in range(len(A)):
                    if i == j: continue
                    for h in range(len(A)): A[h][i] += A[k][i] * A[h][j]
                c[j] = k
                break
        else:
            v = [0] * len(A)
            for j in range(len(A)):
                if j == k: v[j] = 1
                else:
                    for s in range(j):
                        if c[s] == j and c[s] >= 0: v[j] = A[k][s]
            ret.append(v)
    return ret
```

```
# returns DFT(A), len(A) must be a power of 2
import numpy as np
def FFT(A : list) -> list :
    if len(A) == 1: return A
    roots = np.exp(2j*np.pi / len(A)*np.arange(len(A)))
    evens = FFT(A[::2])
    odds = FFT(A[1::2])
    return [even + root*odd for even,root,odd in zip(evens,roots,odds)] + [
        even - root*odd for even,root,odd in zip(evens,roots,odds)]
```

```
# factors n if it has only prime factors p <= S, returns list of exponents
import sympy
def smooth_fact(n, S) -> list:
    fact = []
    for factor in sympy.primerange(S+1):
        e = 0
        while n % factor == 0:
            n //= factor
```

```
            e += 1
        fact.append(e)
    return fact if n == 1 else None
```

```
# factors n using quadratic sieve with smooth number limit S
from math import *
import numpy as np
import galois
import sympy
def quadratic_sieve(n, S, spacetime):
    gen = ((x, smooth_fact(x**2 % n, S)) for x in range(ceil(sqrt(n)), spacetime))
    fact = np.array([f for _, f in gen if f is not None]).T
    roots = [x for x, f in gen if f is not None]
    A = galois.GF(2)(fact % 2)
    solution = np.array(np.linalg.solve(A, galois.GF(2)(np.zeros(len(A)))))
    factor1 = gcd(n, prod(roots) - sympy.primerange(S+1) ** (fact * solution))
    return factor1, n // factor1
```

```
# returns orthogonalized and normed form of basis b
def gram_schmidt(b : list) -> list :
    mu = lambda u,v : (b[v]*b_gs[u]) / (b_gs[u]*b_gs[u])
    b_gs = [b[0]]
    for k in range(1, len(b)):
        b_gs.append(b[k] - sum(mu(b_gs[j], b[k]) * b_gs[j] for j in range(1,k)))
    return b_gs
```

```
# LLL-reduces lattice basis b, vectors are shorter and mostly orthogonal
from math import *
from numpy.linalg import norm
def LLL(b : list) -> list:
    b_gs = gram_schmidt(b)
    mu = lambda u,v : (b[v]*b_gs[u]) / (b_gs[u]*b_gs[u])
    k = 1
    while k <= len(b):
        for j in reversed(range(k)):
            if mu(k, j) > 0.5:
                b[k] -= round(mu(k, j)) * b[j]
                b_gs = gram_schmidt(b)
        if norm(b_gs[k])**2 >= (0.75 - mu(k,k-1))**2 * norm(b_gs[k-1])**2: k += 1
        else:
            b[k], b[k-1] = b[k-1], b[k]
            b_gs = gram_schmidt(b)
            if k > 1: k -= 1
```

- kleiner Fermatsche Satz
  - $a^p \equiv a \bmod p$
  - Falls $p \nmid a : a^{p-1} \equiv 1 \bmod p$
  - Satz von Euler: $a^{\varphi(n)} \equiv 1 \bmod n$ falls $\gcd(a, n) = 1$
- eulersche Phi-Funktion: Sei $n = \prod_{i=1}^{k} p_i^{e_i}$. Dann ist $\varphi(n) = \prod_{i=1}^{k} p_i^{k_i - 1}(p_i - 1)$
- Punktaddition auf epileptischen Kurven
  - geg. epileptische Kurve $y^2 = x^3 + ax + b$ und 2 Punkte $P = (x_1, y_1)$ und $Q = (x_2, y_2)$
  - $m = \frac{y_1 - y_2}{x_1 - x_2}$ falls $x_1 \neq x_2$
  - $m = \frac{3x_1^2 + a}{2y_1}$ sonst
  - $x_3 = m^2 - x_1 - x_2$
  - $y_3 = -y_1 + m(x_1 - x_3)$
  - $P + Q = (x_3, y_3)$

- Hasse Ungleichung: $|N-(q+1)| \leq 2\sqrt{q}$ mit $N =$ Ordnung der epileptischen Kurve, $q =$ Ordnung des endlichen Körpers
- Pocklington-Test für N
  - Sei $a < N$ und p prim, $p \mid N-1$ und $p > \sqrt{N}-1$
  - Fermat: $a^{N-1} \equiv 1 \bmod N$
  - Pocklington: $\gcd\left(a^{\frac{N-1}{p}}-1, N\right) = 1$
- Lucas-Lehmer-Test für $M_p = 2^p - 1$
  - $s_i = s_{i-1}^2 - 2$, $s_0 = 4$
  - M_p ist prim falls $s_{p-2} \equiv 0 \bmod M_p$
- Legendre-Symbol: $\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \bmod p$ mit $p > 2$
  - $\left(\frac{a}{p}\right) = 1$ falls a quadratischer Rest mod p ist
  - $\left(\frac{a}{p}\right) = -1$ falls a quadratischer Nichtrest mod p ist
  - $\left(\frac{a}{p}\right) = 0$ falls $a \equiv 0 \bmod p$
- Jacobi-Symbol: $\left(\frac{a}{n}\right) = \prod_{i=1}^{k} \left(\frac{a}{p_i}\right)^{e_i}$ mit n ungerade und $\prod_{i=1}^{k} p_i^{e_i}$ die Primfaktorzerlegung von n
- Solovay-Strassen-Test: Falls $a^{\frac{n-1}{2}} \not\equiv \left(\frac{a}{n}\right) \bmod n$ für zufällige $a \in [2, n-1]$ ist n zusammengesetzt
- Miller-Rabin-Test für n
  - Wähle $a \in [2, n-2]$ zufällig und sei $d \cdot 2^j + 1 = n$
  - Falls $a^d \equiv 1 \bmod n$ oder $\exists r \in [0, j-1] : a^{d \cdot 2^r} \equiv -1 \bmod n$ ist n prim oder stark pseudoprim bzgl. a
- Karatsuba: Sei $x = x_1 B_m + x_0, y = y_1 B_m + y_0$, dann ist $xy = z_2 B^{2m} + z_1 B^m + z_0$ mit
  - $z_0 = x_0 y_0$
  - $z_2 = x_2 y_2$
  - $z_1 = x_1 y_0 + x_0 y_1 = (x_1 + x_0)(y_1 + y_0) - z_2 - z_0$
- DFT: $X_{k(x)} = \sum_{n=0}^{N-1} x_n \exp\left(\frac{-2i\pi}{N} kn\right)$
- glatte Zahlen: n ist S-glatt, falls in deren Primfaktorzerlegung nur Primfaktoren $p \leq S$ vorkommen
- Montgomeryform
  - $aR = a \cdot R \bmod R$
  - $aR + bR = (a+b)R$
  - $(aR \bmod N)(bR \bmod N)R^{-1} \equiv (ab)R \bmod N$
  - REDC$((aR \bmod N)(bR \bmod N)) = (ab)R \bmod N$
- **p-q-Formel:** $x_0 = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}$
- Rekursionsfolgen:

$F(z) = \sum_{n=0}^{\infty} a_n z^n$

$a_{n+1} - 3a_{n-1} - 2a_{n-2} = 0$

$F(z) - 3z^2 F(z) - 2z^3 F(z) = a_0 + a_1 z + a_2 z^2 - 3a_0 z^2 = 2 + z^2$

$\iff F(z) = \frac{2+z^2}{1-3z^2-2z^3}$

$F(z) = \frac{2+z^2}{1-3z^2-2z^3} = \frac{2+z^2}{(z+1)^2(-2z+1)} = \frac{A}{z+1} + \frac{B}{(z+1)^2} + \frac{C}{-2z+1}$

$\iff 2 + z^2 = A(z+1)(-2z+1) + B(-2z+1) + C(z+1)^2$

Sei $z = \frac{1}{2} \implies C = 1$

Sei $z = -1 \implies B = 1$

Sei $z = 0 \implies A = 0$

$\Longrightarrow F(z) = \frac{1}{(z+1)^2} + \frac{1}{-2z+1} = \frac{1}{(z-(-1))^2} + \frac{1}{1-2z}$

$= \sum_{n=0}^{\infty} \big((-1)^{n(n+1)} + 2^n\big) z^n$

$\Longrightarrow a_n = (-1)^{n(n+1)} + 2^n$

- Summenidentitäten
  - $\sum_{n=0}^{\infty} n z^n = \frac{z}{(1-z)^2}$
  - $\sum_{n=0}^{\infty} n^2 z^n = \frac{z(1+z)}{(1-z)^3}$
  - $\sum_{n=0}^{\infty} a^n z^n = \frac{1}{1-az}$
  - $\sum_{n=0}^{\infty} \binom{c}{n} z^n = (1+z)^c$
  - $\sum_{n=0}^{\infty} \binom{c+n-1}{n} z^n = \frac{1}{(1-z)^c}$
  - $\sum_{n=1} \frac{1}{n} z^n = \ln \frac{1}{1-z}$
- Graphen
  - Eulerkreis: jede Kante einmal, alle Knoten haben geraden Grad
  - Eulerweg: 2 Knoten haben ungeraden Grad
  - Hamilton: jede Ecke einmal
  - Grad: $d(u) = |\{v \in V : (u,v) \in E\}|$
  - Minimalgrad: $\delta(G) = \min\{d(v) : v \in V\}$
  - Maximalgrad: $\Delta(G) = \max\{d(v) : v \in V\}$
  - Kantengraph: $L(G) = (E, \{(e_1, e_2) : e_1, e_2 \in E, e_1 \cap e_2 \neq \emptyset\})$
  - $|\text{Vertices}| - |\text{Edges}| + |\text{Flächen}| = |\text{Zusammenhangskomponenten}| + 1$
  - Matching: $M \subset E$ sodass je 2 Kanten aus M keinen Knoten gemeinsam haben