



American International University-Bangladesh

## AIUB Eclipse

MD Siyam Talukder  
Kazi Shoaib Ahmed Saad  
Faysal Ahammed Chowdhury

```
int legendre(int n, int p) {
    int ex = 0;
    while(n) {
        ex += (n / p);
        n /= p;
    }
    return ex;
}
```

### 3.9 Binary Expo

```
int power(int x, long long n, int mod) {
    int ans = 1 % mod;
    while (n > 0) {
        if (n & 1) {
            ans = 1LL * ans * x % mod;
        }
        x = 1LL * x * x % mod;
        n >>= 1;
    }
    return ans;
}
```

### 3.10 Digit Sum of 1 to N

```
// for n=10, ans = 1+2+...+9+1+0
ll solve(ll n) {
    ll res = 0, p = 1;
    while (n / p > 0) {
        ll left = n / (p * 10);
        ll cur = (n / p) % 10;
        ll right = n % p;
        res += left * 45 * p;
        res += (cur * (cur - 1) / 2) * p;
        res += cur * (right + 1);
        p *= 10;
    }
    return res;
}
```

### 3.11 Pollard Rho

```
namespace PollardRho {
mt19937 rnd(chrono::steady_clock::now().
    time_since_epoch().count());
const int P = 1e6 + 9;
ll seq[P];
int primes[P], spf[P];
inline ll add_mod(ll x, ll y, ll m) {
    return (x += y) < m ? x : x - m;
}
inline ll mul_mod(ll x, ll y, ll m) {
    ll res = __int128(x) * y % m;
    return res;
// ll res = x * y - (ll)((long double)x
//    * y / m + 0.5) * m;
// return res < 0 ? res + m : res;
}
inline ll pow_mod(ll x, ll n, ll m) {
    ll res = 1 % m;
    for (; n; n >>= 1) {
        if (n & 1) res = mul_mod(res, x, m);
        x = mul_mod(x, x, m);
    }
    return res;
}
// O(it * (logn)^3), it = number of
//    rounds performed
inline bool miller_rabin(ll n) {
    if (n <= 2 || (n & 1 ^ 1)) return (n
        == 2);
    if (n < P) return spf[n] == n;
    ll c, d, s = 0, r = n - 1;
    for (; !(r & 1); r >>= 1, s++) {}
    // each iteration is a round
    for (int i = 0; primes[i] < n &&
        primes[i] < 32; i++) {
        c = pow_mod(primes[i], r, n);
        for (int j = 0; j < s; j++) {
            d = mul_mod(c, c, n);
            if (d == 1 && c != 1 && c != (n -
                1)) return false;
            c = d;
        }
        if (c != 1) return false;
        return true;
    }
}
void init() {
    int cnt = 0;
    for (int i = 2; i < P; i++) {
        if (!spf[i]) primes[cnt++] = spf[i]
            = i;
        for (int j = 0, k; (k = i *
            primes[j]) < P; j++) {
            spf[k] = primes[j];
            if (spf[i] == spf[k]) break;
        }
    }
}
// returns O(n^(1/4))
ll pollard_rho(ll n) {
    while (1) {
        ll x = rnd() % n, y = x, c = rnd() %
            n, u = 1, v, t = 0;
        ll *px = seq, *py = seq;
        while (1) {
            *py++ = y = add_mod(mul_mod(y, y,
                n), c, n);
            *py++ = y = add_mod(mul_mod(y, y,
                n), c, n);
            if ((x = *px++) == y) break;
            v = u;
            u = mul_mod(u, abs(y - x), n);
            if (!u) return __gcd(v, n);
            if (++t == 32) {
                t = 0;
                if ((u = __gcd(u, n)) > 1 && u <
                    n) return u;
            }
        }
        if (t && (u = __gcd(u, n)) > 1 && u
            < n) return u;
    }
}
vector<ll> factorize(ll n) {
    if (n == 1) return vector<ll>();
    if (miller_rabin(n)) return vector<ll>
        {n};
    vector<ll> v, w;
    while (n > 1 && n < P) {
        v.push_back(spf[n]);
        n /= spf[n];
    }
    if (n >= P) {
        ll x = pollard_rho(n);
        v = factorize(x);
        w = factorize(n / x);
        v.insert(v.end(), w.begin(), w.end());
    }
    return v;
}
```

}

### 3.12 [Problem] How Many Bases - UVA

```
// Given a number  $N^M$ , find out the
// number of integer bases in which it
// has exactly T trailing zeroes.
int solve_greater_or_equal(vector<int>
    e, int t) {
    int ans = 1;
    for (auto i : e) {
        ans = 1LL * ans * (i / t + 1) % mod;
    }
    return ans;
}
// e contains  $e_1, e_2 \rightarrow p_1^{e_1}, p_2^{e_2}$ 
int solve_equal(vector<int> e, int t) {
    return (solve_greater_or_equal(e, t) -
        solve_greater_or_equal(e, t + 1) +
        mod) % mod;
}
```

### 3.13 [Problem] Power Tower - CF

```
// A sequence  $w_1, w_2, \dots, w_n$  and Q
// queries, l and r will be given.
// Calculate  $w_l^{(w_{l+1}^{(w_{l+2}^{(w_r)})})}$ 
//  $n^x \bmod m = n^{\phi(m)+x \bmod \phi(m)} \bmod m$ 
inline int MOD(int x, int m) {
    if (x < m) return x;
    return x % m + m;
}
int power(int n, int k, int mod) {
    int ans = MOD(1, mod);
    while (k) {
        if (k & 1) ans = MOD(ans * n, mod);
        n = MOD(n * n, mod);
        k >>= 1;
    }
    return ans;
}
int f(int l, int r, int m) {
    if (l == r) return MOD(a[l], m);
    if (m == 1) return 1;
    return power(a[l], f(l + 1, r,
        phi(m)), m);
}
```

### 3.14 Formula and Properties

- $\phi(n) = n \cdot \frac{p_1-1}{p_1} \cdot \frac{p_2-1}{p_2} \dots$
- $\phi(p^e) = p^e - \frac{p^e}{p} = p^e \cdot \frac{p-1}{p}$
- For  $n > 2$ ,  $\phi(n)$  is always even.
- $\sum_{d|n} \phi(d) = n$
- NOD:  $(e_1 + 1) \cdot (e_2 + 1) \dots$
- SOD:  $\frac{p_1^{e_1+1}-1}{p_1-1} \cdot \frac{p_2^{e_2+1}-1}{p_2-1} \dots$
- $\log(a \cdot b) = \log(a) + \log(b)$
- $\log(a^x) = x \cdot \log(a)$
- $\log_a(x) = \frac{\log_b(x)}{\log_b(a)}$
- Digit Count of n:  $\lfloor \log_{10}(n) \rfloor + 1$

- Arithmetic Progression Sum:  $\frac{n}{2} \cdot (a + p), \quad \frac{n}{2} \cdot (2a + (n-1)d)$
- Geometric Sum:  $S_n = a \cdot \frac{r^n - 1}{r - 1}$
- $(1^2 + 2^2 + \dots + n^2) = \frac{n(n+1)(2n+1)}{6}$
- $(1^3 + 2^3 + \dots + n^3) = \frac{n^2(n+1)^2}{4}$
- $(2^2 + 4^2 + \dots + (2n)^2) = \frac{2n(n+1)(2n+1)}{3}$
- $(1^2 + 3^2 + \dots + (2n-1)^2) = \frac{n(2n-1)(2n+1)}{3}$
- $(2^3 + 4^3 + \dots + (2n)^3) = 2n^2(n+1)^2$
- $(1^3 + 3^3 + \dots + (2n-1)^3) = n^2(2n^2 - 1)$
- For any number  $n$  and bases  $> \sqrt{n}$ , there will be no representation where the number contains 0 at its second least significant digit. So it is enough to check for bases  $\leq \sqrt{n}$ .
- For some  $x$  and  $y$ , let's try to find all  $m$  such that  $x \bmod m \equiv y \bmod m$ . We can rearrange the equation into  $(x - y) \equiv 0 \pmod{m}$ . Thus, if  $m$  is a factor of  $|x - y|$ , then  $x$  and  $y$  will be equal modulo  $m$ .

## 4 Combinatorics and Probability

### 4.1 Combinations

```
// Prime Mod in O(n)
void prec() {
    fact[0] = 1;
    for (int i = 1; i < N; i++) {
        fact[i] = 1LL * fact[i - 1] * i % mod;
    }
    ifact[N - 1] = inverse(fact[N - 1]);
    for (int i = N - 2; i >= 0; i--) {
        ifact[i] = 1LL * ifact[i + 1] * (i +
            1) % mod;
    }
}
int nCr(int n, int r) {
    if (r > n) return 0;
    return 1LL * fact[n] * ifact[r] % mod
        * ifact[n - r] % mod;
}
int nPr(int n, int r) {
    if (r > n) return 0;
    return 1LL * fact[n] * ifact[n - r] %
        mod;
}
```

### 4.2 nCr for any mod

```
// Time:  $O(n^2)$ 
// nCr =  $(n-1)C(r-1) + (n-1)Cr$ 
for (int i = 0; i < N; i++) {
    C[i][i] = 1;
    for (int j = 0; j < i; j++) {
        C[i][j] = (C[i - 1][j] + C[i - 1][j
            - 1]) % mod;
    }
}
```

### 4.3 nCk without mod in O(r)

```
ll nCk(ll n, ll k) {
    double res = 1;
    for (ll i = 1; i <= k; ++i)
        res = res * (n - k + i) / i;
    return (ll)(res + 0.01);
}
```

### 4.4 Lucas Theorem

```
// Credit: YouKnOwWho
// returns nCr modulo mod where mod is a
// prime
// Complexity: log(n)
const int N = 1e6 + 3, mod = 1e6 + 3;
template <const int32_t MOD>
struct modint {
    int32_t value;
    modint() = default;
    modint(int32_t value_) : value(value_)
    {}
    inline modint<MOD> operator +
    (modint<MOD> other) const {
        int32_t c = this->value +
        other.value; return modint<MOD>(c
        >= MOD ? c - MOD : c); }
    inline modint<MOD> operator -
    (modint<MOD> other) const {
        int32_t c = this->value -
        other.value; return modint<MOD>(c
        < 0 ? c + MOD : c); }
    inline modint<MOD> operator *
    (modint<MOD> other) const {
        int32_t c = (int64_t)this->value *
        other.value % MOD; return
        modint<MOD>(c < 0 ? c + MOD : c); }
    inline modint<MOD> & operator +=
    (modint<MOD> other) { this->value
    += other.value; if (this->value >=
    MOD) this->value -= MOD; return
    *this; }
    inline modint<MOD> & operator -=
    (modint<MOD> other) { this->value
    -= other.value; if (this->value <
    0) this->value += MOD; return
    *this; }
    inline modint<MOD> & operator ==
    (modint<MOD> other) { this->value
    = (int64_t)this->value *
    other.value % MOD; if (this->value
    < 0) this->value += MOD; return
    *this; }
    inline modint<MOD> operator - () const
    { return modint<MOD>(this->value ?
    MOD - this->value : 0); }
    modint<MOD> pow(uint64_t k) const {
        modint<MOD> x = *this, y = 1; for
        (; k >>= 1) { if (k & 1) y *=
        x; x *= x; } return y; }
    modint<MOD> inv() const { return
    pow(MOD - 2); } // MOD must be a
    prime
    inline modint<MOD> operator /
    (modint<MOD> other) const { return
    *this * other.inv(); }
```

```
inline modint<MOD> operator /=
    (modint<MOD> other) { return
    *this * other.inv(); }
    inline bool operator == (modint<MOD>
    other) const { return value ==
    other.value; }
    inline bool operator != (modint<MOD>
    other) const { return value !=
    other.value; }
    inline bool operator < (modint<MOD>
    other) const { return value <
    other.value; }
    inline bool operator > (modint<MOD>
    other) const { return value >
    other.value; }
};
template <int32_t MOD> modint<MOD>
operator * (int32_t value,
modint<MOD> n) { return
modint<MOD>(value) * n; }
template <int32_t MOD> modint<MOD>
operator * (int64_t value,
modint<MOD> n) { return
modint<MOD>(value % MOD) * n; }
template <int32_t MOD> istream &
operator >> (istream & in,
modint<MOD> & n) { return in >>
n.value; }
template <int32_t MOD> ostream &
operator << (ostream & out,
modint<MOD> n) { return out <<
n.value; }
using mint = modint<mod>;
struct combi {
    int n; vector<mint> facts, finvs, invs;
    combi(int _n): n(_n), facts(_n),
    finvs(_n), invs(_n) {
        facts[0] = finvs[0] = 1;
        invs[1] = 1;
        for (int i = 2; i < n; ++i) invs[i]
        = invs[i-1] * (-mod / i);
        for (int i = 1; i < n; ++i) {
            facts[i] = facts[i-1] * i;
            finvs[i] = finvs[i-1] * invs[i];
        }
    }
    inline mint fact(int n) { return
    facts[n]; }
    inline mint finv(int n) { return
    finvs[n]; }
    inline mint inv(int n) { return
    invs[n]; }
    inline mint ncr(int n, int k) { return
    n < k or k < 0 ? 0 : facts[n] *
    finvs[k] * finvs[n-k]; }
};
combi C(N);
mint lucas(ll n, ll r) {
    if (r > n) return 0;
    if (n < mod) return C.ncr(n, r);
    return lucas(n / mod, r / mod) *
    lucas(n % mod, r % mod);
}
cout << lucas(100000000, 2322) << '\n';
```

### 4.5 Catalan Number

```
const int MOD = 1e9 + 7, int MAX = 1e7;
int catalan[MAX];
void init(ll n) {
    catalan[0] = catalan[1] = 1;
    for (ll i = 2; i <= n; i++) {
        catalan[i] = 0;
        for (ll j = 0; j < i; j++) {
            catalan[i] += (catalan[j] *
            catalan[i - j - 1]) % MOD;
            if (catalan[i] >= MOD) {
                catalan[i] -= MOD;
            }
        }
    }
}
```

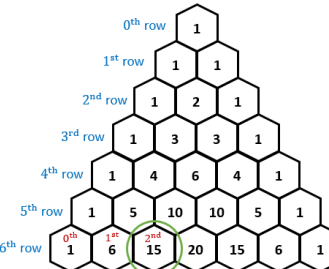
### 4.6 Derangement

```
// number of combinations such that
// a_i != i of a permutation a
const int N = 1e6 + 100, int p = 1e9 + 7;
ll der[N];
void countDer() {
    der[1] = 0;
    der[2] = 1;
    for (ll i = 3; i <= N; ++i) {
        der[i] = (i - 1) % p * (der[i - 1] %
        p + der[i - 2] % p);
        der[i] %= p;
    }
}
```

### 4.7 Stars and Bars Theorem

- Find the number of  $k$ -tuples of non-negative integers whose sum is  $n$ .  $\binom{n+k-1}{n}$
- Find the number of  $k$ -tuples of non-negative integers whose sum is  $\leq n$ .  $\binom{n+k}{k}$
- Combination with Repetition (choose  $k$  elements from  $n$  objects, same element can be chosen multiple times).  $\binom{n+k-1}{k}$
- How many ways to go from  $(0,0)$  to  $(n,m)$ .  $\binom{n+m}{m}$

Pascals Triangle is equivalent to nCr:



### 4.8 Properties of Pascal's Triangle

- $(a + b)^n = \sum_{k=0}^n \binom{n}{k} a^k b^{n-k}$
- $(k + 1)^n = \sum_{i=0}^n k^i \cdot \binom{n}{i}$

- $\sum_{i=0}^n \binom{n}{i} = 2^n$
- $\binom{k}{n} = \frac{k!}{n!(k-n)!}$
- $\sum_{k=0}^m \binom{n+k}{k} = \binom{n+m+1}{m}$
- $\binom{n}{0}^2 + \binom{n}{1}^2 + \dots + \binom{n}{n}^2 = \binom{2n}{n}$
- $1 \binom{n}{1} + 2 \binom{n}{2} + \dots + n \binom{n}{n} = n 2^{n-1}$

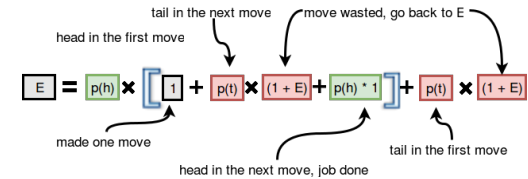
### 4.9 Contribution Technique

**Rule:** Calculate the contribution of each element separately.

- Sum of all pair sums:  $\sum_{i=1}^n \sum_{j=1}^n (a_i + a_j)$   
Every element will be added  $2n$  times.  
 $\sum_{i=1}^n (2 \times n \times a_i) = 2 \times n \times \sum_{i=1}^n a_i$ .
- Sum of all subarray sums —  $\sum_{i=1}^n (a_i \times i \times (n - i + 1))$ .
- Sum of all Subsets sums —  $\sum_{i=1}^n (2^{n-1} \times a_i)$ .
- Product of all pair product —  $\prod_{i=1}^n (a_i^{2 \times n})$ .
- XOR of subarray XORs — How many subarrays does an element have?  $(i \cdot (n - i + 1))$  times.  
If subarray count is odd then this element can contribute in total XORs.
- Sum of max minus min over all subset — Sort the array.  $Min = 2^{n-i}$ ,  $Max = 2^{i-1}$ .  
 $\sum_{i=1}^n (a_i \cdot 2^{i-1} - a_i \cdot 2^{n-i})$
- Sum using bits —  $\sum_{k=0}^{30} (cnt_k[1] \times 2^k)$ .
- Sum of Pair XORs — XOR will 1 if two bits are different  $\sum_{k=0}^{30} (cnt_k[0] \times cnt_k[1] \times 2^k)$ .
- Sum of Pair ANDs —  $\sum_{k=0}^{30} (cnt_k[1]^2 \times 2^k)$ .
- Sum of Pair ORs —  $\sum_{k=0}^{30} ((cnt_k[1]^2 + 2 \times cnt_k[1] \times cnt_k[0]) \times 2^k)$ .
- Sum of Subset XORs — where  $cnt_0! = 0$   
 $\sum_{k=0}^{30} (2^{cnt_k[1] + cnt_k[0] - 1} \times 2^k)$ .
- Sum of Subset ANDs —  $\sum_{k=0}^{30} ((2^{cnt_k[1]} - 1) \times 2^k)$ .
- Sum of Subset ORs —  $\sum_{k=0}^{30} ((2^n - 2^{cnt_k[0]}) \times 2^k)$ .
- Sum of subarray XORs — Convert to prefix xor, then solve for pairs.

4.10 Probability and Expected Value

- Expected value:  $E = \sum_{i=1}^n P_i \cdot i$
- Variance:  $V(x) = E(x^2) - \{E(x)\}^2$
- To get two consecutive heads, what is the expected number of tosses?



- To get  $n$  heads, what is the expected number of tosses? Let's define: to get  $n$  heads, we need to toss  $E(n)$  times. Now — I can get a head; I need to toss  $E(n - 1)$  more times, or if I get a tail; I need to toss  $E(n)$  times. So, the recurrence is:  $E(n) = 0.5 \cdot (1 + E(n - 1)) + 0.5 \cdot (1 + E(n))$

তোমার কাছে  $n$  টা বাস্ব আছে, শুরুতে সবগুলো বাস্ব অফ। প্রতিটা মুভ এ তুমি random একটা বাস্ব সিলেক্ট করতে পারো। এখন বাস্বটা যদি অফ থাকে তাহলে তুমি একটা কয়েন টস করবে, যদি হেড পাও তাহলে বাস্বটা অন করবে, যদি টেইল পাও তাহলে কিছুই করবে না। আর বাস্বটা যদি আগেই অন থাকে তাহলে সেই মুভে তোমার কিছুই করা দরকার নেই। এক্সপেক্টেড কয়টা মুভে তুমি সবগুলো বাস্ব অন করতে পারবে? কয়েনটা ফেয়ার কয়েন না, প্রতিবার টেইল পাবার প্রোবাবিলিটি  $p$ ।

এই প্রবলেমও রিকার্সিভলি সলভ করতে হবে। তোমার মূলত জানা দরকার বর্তমানে কয়টা বাস্ব অন আছে। ধরো বর্তমানে  $x$  টা বাস্ব অন আছে এবং এক্সপেক্টেড মুভ লাগবে  $e(x)$  টি। তাহলে অলরেডি অন আছে এমন বাস্ব সিলেক্ট করার প্রোবাবিলিটি  $\frac{x}{n}$ , সেক্ষেত্রে এক্সপেক্টেড মুভ লাগবে আরো  $\frac{x}{n} (1 + e(x))$  টি। অলরেডি অফ আছে এমন বাস্ব সিলেক্ট করার প্রোবাবিলিটি  $\frac{n-x}{n}$ । সেক্ষেত্রে আবার ২টি ঘটনা ঘটতে পারে,  $p$  প্রোবাবিলিটিতে তুমি টেইল পাবে এবং আরো  $e(x)$  টি মুভ লাগবে, অথবা  $1 - p$  প্রোবাবিলিটিতে হেড পাবে এবং আরো  $e(x + 1)$  টি মুভ লাগবে।

**Eq:**  $e(x) = \frac{x}{n} \cdot (1 + e(x)) + \frac{n-x}{n} \cdot (p \cdot (1 + e(x)) + (1 - p) \cdot (1 + e(x + 1)))$