American International University-Bangladesh

AIUB Eclipse

MD Siyam Talukder
Kazi Shoaib Ahmed Saad
Faysal Ahammed Chowdhury

Contents

1 Setup

1.1 Sublime Build

```
"shell_cmd": "g++ -std=c++17 -o
    \"$file_base_name\" \"$file\" &&
    timeout 2.5s ./\"$file_base_name\" <
    input.txt > output.txt",
"file_regex":
    "^(..[^:]*):([0-9]+):?([0-9]+)?:?
    (.*)$",
"working_dir": "${file_path}",
"selector": "source.c, source.c++"
```

2 Stress Testing

2.1 Input Gen

```
mt19937_64 rnd(chrono::steady_clock::now
    ().time_since_epoch().count());
ll get_rand(ll l, ll r) {
  assert(l <= r);
  return l + rnd() % (r - l + 1);
}
```

2.2 Bash Script

```
// run -> bash script.sh
set -e
g++ code.cpp -o code
g++ gen.cpp -o gen
g++ brute.cpp -o brute
for((i = 1; ; ++i)); do
    ./gen $i > input_file
    ./code < input_file > myAnswer
    ./brute < input_file > correctAnswer
    diff -Z myAnswer correctAnswer >
        /dev/null || break
    echo "Passed test: "  $i
done
echo "WA on the following test:"
cat input_file
echo "Your answer is:"
cat myAnswer
echo "Correct answer is:"
cat correctAnswer
```

3 Number Theory

3.1 Euler Totient Function

```
// Time: O(√N)
map<int, int> dp; // memo
int phi(int n) {
  if (dp.count(n)) return dp[n];
  int ans = n, m = n;
  for (int i = 2; i * i <= m; i++) {
    if (m % i == 0) {
      while (m % i == 0) m /= i;
      ans = ans / i * (i - 1);
    }
  }
  if (m > 1) ans = ans / m * (m - 1);
  return dp[n] = ans;
}
```

3.2 Phi 1 to N

```
void phi_1_to_n(int n) {
  vector<int> phi(n + 1);
  for (int i = 0; i <= n; i++)
    phi[i] = i;
  for (int i = 2; i <= n; i++) {
    if (phi[i] == i) {
      for (int j = i; j <= n; j += i)
        phi[j] -= phi[j] / i;
    }
  }
}
```

3.3 Segmented Sieve

```
vector<char> segmentedSieve(ll L, ll R) {
  // generate all primes up to √R
  ll lim = sqrt(R);
  vector<char> mark(lim + 1, false);
  vector<ll> primes;
  for (ll i = 2; i <= lim; ++i) {
    if (!mark[i]) {
      primes.emplace_back(i);
      for (ll j = i * i; j <= lim; j +=
          i) mark[j] = true;
    }
  }
  vector<char> isPrime(R - L + 1, true);
  for (ll i : primes)
    for (ll j = max(i * i, (L + i - 1) /
        i * i); j <= R; j += i)
      isPrime[j - L] = false;
  if (L == 1) isPrime[0] = false;
  return isPrime;
}
```

3.4 Extended GCD

```
// ax + by = gcd(a,b)
int egcd(int a, int b, int& x, int& y) {
  if (b == 0) {
    x = 1, y = 0;
    return a;
  }
  int x1, y1;
  int d = egcd(b, a % b, x1, y1);
  x = y1;
  y = x1 - y1 * (a / b);
  return d;
}
```

3.5 Linear Diophantine Equation

```
// ax + by = c, find any x and y
bool find_any_solution(int a, int b, int
    c, int &x0, int &y0, int &g) {
  g = egcd(abs(a), abs(b), x0, y0);
  if (c % g) return false;
  x0 *= c / g;
  y0 *= c / g;
  if (a < 0) x0 = -x0;
  if (b < 0) y0 = -y0;
  return true;
}
void shift_solution(int & x, int & y,
    int a, int b, int cnt) {
  x += cnt * b;
  y -= cnt * a;
}
int find_all_solutions(int a, int b, int
    c, int minx, int maxx, int miny, int
    maxy) {
  int x, y, g;
  if (!find_any_solution(a, b, c, x, y,
      g)) return 0;
  a /= g, b /= g;
  int sign_a = a > 0 ? +1 : -1;
  int sign_b = b > 0 ? +1 : -1;
  shift_solution(x, y, a, b, (minx - x)
      / b);
  if (x < minx) shift_solution(x, y, a,
      b, sign_b);
  if (x > maxx) return 0;
  int lx1 = x;
  shift_solution(x, y, a, b, (maxx - x)
      / b);
  if (x > maxx) shift_solution(x, y, a,
      b, -sign_b);
  int rx1 = x;
  shift_solution(x, y, a, b, -(miny - y)
      / a);
  if (y < miny) shift_solution(x, y, a,
      b, -sign_a);
  if (y > maxy) return 0;
  int lx2 = x;
  shift_solution(x, y, a, b, -(maxy - y)
      / a);
  if (y > maxy) shift_solution(x, y, a,
      b, sign_a);
  int rx2 = x;
  if (lx2 > rx2) swap(lx2, rx2);
  int lx = max(lx1, lx2);
  int rx = min(rx1, rx2);
  if (lx > rx) return 0;
  return (rx - lx) / abs(b) + 1;
}
```

3.6 Modular Inverse using EGCD

```
// finding inverse(a) modulo m
int x, y;
int g = extended_euclidean(a, m, x, y);
if (g != 1) cout << "No solution!";
else {
  x = (x % m + m) % m;
  cout << x << endl;
}
```

3.7 Exclusion DP

```
ll f[N], g[N];
for (int i = N - 1; i >= 1; i--) {
  f[i] = nC4(div_cnt[i]);
  g[i] = f[i];
  for (int j = i + i; j < N; j += i) {
    g[i] -= g[j];
  }
}
```

Here, $f[i]$ = how many pairs/k-tuple such that their gcd is i or it's multiple (count of pairs those are divisible by i).
$g[i]$ = how many pairs/k-tuple such that their gcd is i.
$g[i] = f[i] - \sum_{i|j} g[j]$.

Sum of all pair gcd:

We know, how many pairs are there such that their gcd is i for every i (1 to n). So now, $\sum_{i=1}^{n} g[i] * i$.

Sum of all pair lcm (i = 1, j = 1):
We know, $\text{lcm}(a, b) = \frac{a*b}{\gcd(a,b)}$.

Now, $f[i]$ = All pair product sum of those, whose gcd is i or it's multiple.
$g[i]$ = All pair product sum of those, whose gcd is i.

Ans $= \sum_{i=1}^{n} \frac{g[i]}{i}$.

All pair product sum $= (a_1 + a_2 + \cdots + a_n) * (a_1 + a_2 + \cdots + a_n)$

3.8 Legendres Formula

```
// n!/p^x - you will get the largest x
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 $e_1$, $e_2$ -> ${p_1}^{e_1}$, ${p_2}^{e_2}$
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 $w_1, w_2, ..., w_n$ and Q
    queries, l and r will be given.
    Calculate $w_l^{(w_{l+1}^{\cdots^{(w_r)}})}$
// $n^x \bmod m = n^{\phi(m)+x \bmod \phi(m)} \bmod m$
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

- $\phi(n) = n \cdot \frac{p_1-1}{p_1} \cdot \frac{p_2-1}{p_2} \cdots$
- $\phi(p^e) = p^e - \frac{p^e}{p} = p^e \cdot \frac{p-1}{p}$
- For $n > 2$, $\phi(n)$ is always even.
- $\sum_{d|n} \phi(d) = n$
- NOD: $(e_1+1) \cdot (e_2+1) \cdots$
- SOD: $\frac{p_1^{e_1+1}-1}{p_1-1} \cdot \frac{p_2^{e_2+1}-1}{p_2-1} \cdots$
- $\log(a \cdot b) = \log(a) + \log(b)$
- $\log(a^x) = x \cdot \log(a)$
- $\log_a(x) = \frac{\log_b(x)}{\log_b(a)}$
- **Digit Count of n:** $\lfloor \log_{10}(n) \rfloor + 1$
- **Arithmetic Progression Sum:** $\frac{n}{2} \cdot (a + p)$, $\frac{n}{2} \cdot (2a + (n-1)d)$
- **Geometric Sum:** $S_n = a \cdot \frac{r^n - 1}{r - 1}$
- $(1^2 + 2^2 + \cdots + n^2) = \frac{n(n+1)(2n+1)}{6}$
- $(1^3 + 2^3 + \cdots + n^3) = \frac{n^2(n+1)^2}{4}$
- $(2^2 + 4^2 + \cdots + (2n)^2) = \frac{2n(n+1)(2n+1)}{3}$
- $(1^2 + 3^2 + \cdots + (2n-1)^2) = \frac{n(2n-1)(2n+1)}{3}$
- $(2^3 + 4^3 + \cdots + (2n)^3) = 2n^2(n+1)^2$
- $(1^3 + 3^3 + \cdots + (2n-1)^3) = n^2(2n^2 - 1)$
- 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 $\le \sqrt{n}$.
- 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; 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[mod % i] * (-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;
      }
    }
  }
}
```

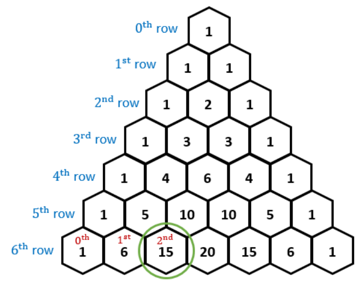
4.6 Derangement

```
    // number of combinations such that
    a_i! = i of a permuation 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

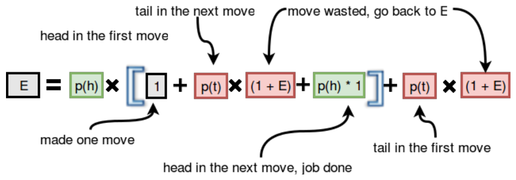
- $(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$
- $(k+1)^n = \sum_{i=0}^{n} k^i \cdot \binom{n}{i}$
- $\displaystyle\sum_{i=0}^{n} \binom{n}{i} = 2^n$
- $\binom{k}{n} = \frac{k}{n}\binom{k-1}{n-1}$
- $\displaystyle\sum_{k=0}^{m} \binom{n+k}{k} = \binom{n+m+1}{m}$
- $\binom{n}{0}^2 + \binom{n}{1}^2 + \cdots + \binom{n}{n}^2 = \binom{2n}{n}$
- $1\binom{n}{1} + 2\binom{n}{2} + \cdots + n\binom{n}{n} = n\,2^{n-1}$

4.9 Contribution Technique

- 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 — $\sum_{i=1}^{n}\left(a_i \times i \times (n-i+1)\right)$.
- Sum of all Subsets sums — $\sum_{i=1}^{n}\left(2^{n-1} \times a_i\right)$.
- Product of all pair product — $\prod_{i=1}^{n}\left(a_i^{2\times n}\right)$.
- 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. $Min = 2^{n-i}$, $Max = 2^{i-1}$. $\sum_{i=1}^{n}\left(a_i \cdot 2^{i-1} - a_i \cdot 2^{n-i}\right)$
- Sum using bits — $\sum_{k=0}^{30}\left(cnt_k[1] \times 2^k\right)$.
- Sum of Pair XORs — XOR will 1 if two bits are different $\sum_{k=0}^{30}\left(cnt_k[0] \times cnt_k[1] \times 2^k\right)$.
- Sum of Pair ANDs — $\sum_{k=0}^{30}\left(cnt_k[1]^2 \times 2^k\right)$.
- Sum of Pair ORs — $\sum_{k=0}^{30}\left(\left(cnt_k[1]^2 + 2 \times cnt_k[1] \times cnt_k[0]\right) \times 2^k\right)$.
- Sum of Subset XORs — where $cnt0! = 0$ $\sum_{k=0}^{30}\left(2^{cnt_k[1]+cnt_k[0]-1} \times 2^k\right)$.
- Sum of Subset ANDs — $\sum_{k=0}^{30}\left(\left(2^{cnt_k[1]} - 1\right) \times 2^k\right)$.
- Sum of Subset ORs — $\sum_{k=0}^{30}\left(\left(2^n - 2^{cnt_k[0]}\right) \times 2^k\right)$.
- 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?


tail in the next move
move wasted, go back to E
head in the first move
E = p(h) × [1 + p(t) × (1 + E) + p(h) * 1] + p(t) × (1 + E)
made one move
tail in the first move
head in the next move, job done

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

এই প্রবলেমও রিকার্সিভলি সলভ করতে হবে। তোমার মুলত জানা দরকার বর্তমানে কয়টা বাল্ব অন আছে। ধরো বর্তমানে x টা বাল্ব অন আছে এবং এক্সপেক্টেড মুভ লাগবে $e(x)$ টি। তাহলে অলরেডি অন আছে এমন বাল্ব সিলেক্ট করার প্রোবাবিলিটি $\frac{x}{n}$, সেক্ষেত্রে এক্সপেক্টেড মুভ লাগবে আরো $\frac{x}{n}(1 + e(x))$ টি। অলরেডি অফ আছে এমন বাল্ব 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: $e(x) = \frac{x}{n} \cdot (1 + e(x)) + \frac{n-x}{n} \cdot (p \cdot (1 + e(x)) + (1 - p) \cdot (1 + e(x+1))$