Math Handbook

Cody Barnson

January 15, 2019

1 Math

1.0.1 Log base conversion

$$\frac{log_x n}{log_x B} = log_B n$$

1.0.2 Ceiling integer division

$$\left\lfloor \frac{n}{d} \right\rfloor = \frac{n + d - 1}{d}$$

1.0.3 Bit shift equivalent of multiply by 10

```
// (x << 3) + (x << 1) ≡ x * 10
int x, y;
// ...
x = (x << 3) + (x << 1);
y = y * 10;
assert(x == y);
```

1.1 C++

1.1.1 Logarithm base 2

```
// log2(n)
log2(n) = 31 - __builtin_clz(n);
```

1.1.2 Add value, update average

```
// avg_{n+1} = (sum_n(n+1) + kn - sum) / (n(n+1))
int n, sum;
// ...
double avg = sum / n;
while ((int)(avg + 0.5) < k) {
    avg = sum * n + sum + k * n - sum;
    avg /= n * n + n;
    sum += k;
    n++;
}
```

1.1.3 Binomial coefficient

```
// (n choose k) = (n-1 choose k-1) + (n-1 choose k)
typedef long long ll;
ll binom(int n, int k) {
    if (k == 0 || k == n) return 1;
    k = min(k, n - k); // Since (n choose k) ≡ (n choose n-k)
    ll ans = 1LL;
    for (ll i = 1; i <= k; i++) {
        ans = ans * (n - k + i) / i;
    }
}
ll choose(int n, int k, ll p = 1e9+7) {
    if (n < k) return 0;
    k = min(k, n - k);
    ll num = 1, den = 1;
    for (int i = 0; i < k; i++) num = num * (n - i) % p;
    for (int i = 1; i <= k; i++) den = den * i % p;
    return num * powmod(den, p - 2, p) % p;
}
ll multichoose(int n, int k, ll p = 1e9+7) {
    return choose(n + k - 1, k, p);
}
```

1.1.4 Catalan numbers

```
typedef long long ll;
ll catalan(int n, ll p = 1e9+7) {
    return choose(2 * n, n, p) * powmod(n + 1, p - 2, p) % p;
}
ll powmod(ll x, ll n, ll m) {
    ll a = 1, b = x;
    for (; n > 0; n >>= 1) {
        if (n & 1) a = mulmod(a, b, m);
        b = mulmod(b, b, m);
    }
    return a % m;
}
ll mulmod(ll x, ll n, ll m) {
    ll n = 0, b = x % m;
    for (; n > 0; n >>= 1) {
        if (n & 1) a = (a + b) % m;
        b = (b << 1) % m;
    }
    return a % m;
}
```

1.1.5 Count number of digits in a number

```
// digits = ⌊log10(n)⌋ + 1
int countDigits(long long n) {
    return n > 0 ? (int)log10((double)n) + 1 : 1;
}
```

1.1.6 Enumerate combinations of N elements in K in lexical order

```
// N, K ∈ ℕ, consider set numbers 1...N, derive all its different subsets of
// cardinality K, in lexical order.
bool next_combination(vector<int> &a, int n) {
    int k = a.size();
    for (int i = k - 1; i >= 0; --i) {
        if (a[i] < n - k + i + 1) {
            ++a[i];
            for (int j = i + 1; j < k; ++j) {
                a[j] = a[j - 1] + 1;
            }
            return true;
        }
    }
    return false;
}
```

1.1.7 Prime factorization

```
typedef vector<int> vi;
vi factor(int n) {
    vi f;
    if (n < 2) return vi();
    while (~n & 1) n /= 2, f.push_back(2);
    for (long long p = 3; p * p <= n; p += 2)
        while (n % p == 0) n /= p, f.push_back((int)p);
    if (n > 1) f.push_back(n);
    return f;
}
```

1.1.8 Fibonacci

```
// Matrix Exponentiation method
// Complexity: O(log(n))
// fib(0) = 0, fib(1) = 1
// Note: fib(≥ 47) will overflow a 32-bit signed integer
int f[1000];
int fib(int n) {
  if (n < 2) return n;
  if (f[n]) return f[n];
  int k = (n + 1) / 2;
  f[n] = (n & 1) ? fib(k) * fib(k) + fib(k - 1) * fib(k - 1)
                 : (2 * fib(k - 1) + fib(k)) * fib(k);
  return f[n];
}
```

1.1.9 Modular Exponentiation

```
// Complexity: O(log(n))
// Compute x^n mod m
int modexp(int x, int n, int m) {
  if (n == 0) return 1;
  if (n & 1) return ((x % m) * modexp(x, n - 1, m)) % m;
  int y = modexp(x, n / 2, m);
  return (y * y) % m;
}
```

1.1.10 Sieve + Optimized primality testing

```
// Sieve + optimized prime testing
typedef long long ll;
typedef vector<int> vi;

ll sz;
bitset<10000010> p; // 10^7 + 10
vi primes;
void sieve(ll m) {
    sz = m + 1;
    p.set();
    p[0] = p[1] = 0;
    for (ll i = 2; i <= sz; i++) {
        if (p[i]) {
            for (ll j = i * i; j <= sz; j += i) {
                p[j] = 0;
            }
            primes.push_back((int)i);
        }
    }
}
bool isPrime(ll x) {
    if (x <= sz) return p[x];
    for (int i = 0; i < (int)primes.size(); i++) {
        if (x % primes[i] == 0) return false;
    }
    return true;
}
```

1.1.11 Base conversion

```
// Base conversion
// Complexity: O(N), N digits
// Given digits of int x in base a, return x's digits in base b.

typedef vector<int> vi;

// x : digit representation of number
// a : base of x
// b : desired base
// returns => vector<int> digits of number in base b.
// Note: vec[0] stores the most significant digit.
vi convert_base(const vi &x, int a, int b) {
    unsigned long long base10 = 0;
    FR(i, x.size()) base10 += x[i] * pow(a, x.size() - i - 1);
    int N = ceil(log(base10 + 1) / log(b));
    vi bb;
    for (int i = 1; i <= N; i++)
        bb.emplace_back((int)(base10 / pow(b, N - i)) % b);
    return bb;
}

// x : number
// b : desired base
// returns => vector<int> digits of number in base b
vi base_digits(int x, int b = 10) {
    vi bb;
    while (x != 0) bb.emplace_back(x % b), x /= b;
    reverse(begin(bb), end(bb));
    return bb;
}

int main() {
    // consider 123_5, (i.e. 123 in base 5)
    vi x{1, 2, 3}; int a = 5;
    vi z = convert_base(x, a, 10); // 123_5 = 38_10, z = {3, 8}
    vi y = convert_base(x, a, 3); // 123_5 = 1102_3, y = {1, 1, 0, 2}
}
```