Chapter 1

Introduction

Competitive programming menggabungkan dua topik:

(1) desain algoritma dan
(2) implementasi algoritma.

Desain algoritma terdiri dari pemecahan masalah dan pemikiran matematis. Keterampilan untuk menganalisis masalah dan menyelesaikannya dengan kreatif sangat diperlukan. Algoritma untuk menyelesaikan sebuah masalah harus benar dan efisien, dan inti dari masalah sering kali terletak pada menciptakan algoritma yang efisien.

Pengetahuan teoretis tentang algoritma sangat penting bagi para competitive programmer. Biasanya, solusi untuk sebuah masalah adalah kombinasi dari teknik-teknik yang sudah dikenal dan wawasan baru. Teknik-teknik yang muncul dalam **competitive programming** juga membentuk dasar untuk penelitian ilmiah tentang algoritma.

Implementasi algoritma membutuhkan keterampilan pemrograman yang baik. Dalam competitive programming, solusi dinilai dengan menguji algoritma yang diimplementasikan menggunakan serangkaian kasus uji. Jadi, tidak cukup hanya ide algoritma yang benar, implementasinya juga harus benar.

Gaya penulisan kode yang baik dalam kontes adalah yang langsung dan ringkas. Program harus ditulis dengan cepat, karena waktu yang tersedia terbatas. Berbeda dengan rekayasa perangkat lunak tradisional, program-program ini singkat (biasanya tidak lebih dari beberapa ratus baris kode), dan tidak perlu dipelihara setelah kontes.

Bahasa Pemrograman

Saat ini, bahasa pemrograman yang paling populer digunakan dalam kontes adalah C++, Python, dan Java. Misalnya, dalam **Google Code Jam 2017**, di antara 3.000 peserta terbaik, 79% menggunakan C++, 16% menggunakan Python, dan 8% menggunakan Java. Beberapa peserta juga menggunakan beberapa bahasa.

Banyak orang berpendapat bahwa C++ adalah pilihan terbaik untuk seorang **competitive programmer**, dan C++ hampir selalu tersedia di sistem kontes. Keuntungan menggunakan C++ adalah bahwa ini adalah bahasa yang sangat efisien dan pustaka standarnya mengandung banyak struktur data dan algoritma.

Di sisi lain, ada baiknya menguasai beberapa bahasa pemrograman dan memahami kekuatannya. Misalnya, jika angka besar dibutuhkan dalam masalah, Python bisa menjadi pilihan yang baik, karena memiliki operasi built-in untuk perhitungan dengan angka besar. Namun, sebagian besar masalah dalam kontes pemrograman disusun sedemikian rupa sehingga penggunaan bahasa pemrograman tertentu tidak memberikan keuntungan yang tidak adil.

Semua program contoh dalam buku ini ditulis dalam C++, dan struktur data serta algoritma dari pustaka standar sering digunakan. Program-program ini mengikuti standar **C++11**, yang dapat digunakan dalam sebagian besar kontes saat ini. Jika kamu belum bisa memrogram menggunakan C++, sekarang adalah waktu yang tepat untuk mulai belajar.

Template Kode C++

Template kode C++ yang khas untuk **competitive programming** terlihat seperti ini:

```
#include <bits/stdc++.h>

using namespace std;

int main() {
    // solusi disini
}
```

Baris `#include` di awal kode adalah fitur dari compiler g++ yang memungkinkan kita untuk menyertakan seluruh pustaka standar. Dengan demikian, kita tidak perlu menyertakan pustaka seperti `iostream`, `vector`, dan `algorithm` secara terpisah, karena pustaka-pustaka tersebut tersedia secara otomatis.

Baris `using` mendeklarasikan bahwa kelas dan fungsi dari pustaka standar dapat digunakan langsung dalam kode. Tanpa baris `using`, kita harus menulis, misalnya, `std::cout`, tetapi sekarang cukup menulis `cout`.

Kode ini dapat dikompilasi menggunakan perintah berikut:

```
g++ -std=c++11 -O2 -Wall test.cpp -o test
```

Perintah ini menghasilkan file biner `test` dari kode sumber `test.cpp`. Compiler mengikuti standar **C++11** (`-std=c++11`), mengoptimalkan kode (`-O2`), dan menampilkan peringatan tentang kemungkinan kesalahan (`-Wall`).

Input dan Output

Dalam sebagian besar kontes, aliran standar (standard streams) digunakan untuk membaca input dan menulis output. Dalam C++, aliran standar untuk input adalah `cin` dan untuk output adalah `cout`. Selain itu, fungsi C `scanf` dan `printf` juga dapat digunakan.

Input untuk program biasanya terdiri dari angka dan string yang dipisahkan dengan spasi dan baris baru. Input tersebut dapat dibaca dari aliran `cin` sebagai berikut:

```
int a, b;
string x;
cin >> a >> b >> x;
```

Kode seperti ini selalu bekerja, dengan asumsi ada setidaknya satu spasi atau baris baru di antara setiap elemen input. Sebagai contoh, kode di atas dapat membaca kedua input berikut:

```
123 456 monkey
```

```
123 456
monkey
```

Aliran `cout` digunakan untuk output sebagai berikut:

```
int a = 123, b = 456;
string x = "monkey";
cout << a << " " << b << " " << x << "\n";
```

Input dan output kadang-kadang bisa menjadi hambatan dalam program. Baris berikut di awal kode membuat input dan output menjadi lebih efisien:

```
ios::sync_with_stdio(0);
cin.tie(0);
```

Perhatikan bahwa newline "`\n`" lebih cepat daripada `endl`, karena `endl` selalu menyebabkan operasi flush.

Fungsi C `scanf` dan `printf` adalah alternatif dari aliran standar C++. Mereka biasanya sedikit lebih cepat, tetapi juga lebih sulit digunakan. Berikut adalah contoh kode untuk membaca dua angka bulat dari input:

```
int a, b;
scanf("%d %d", &a, &b);
```

Berikut adalah contoh kode untuk mencetak dua angka bulat:

```
int a = 123, b = 456;
printf("%d %d\n", a, b);
```

Terkadang program perlu membaca satu baris penuh dari input, yang mungkin mengandung spasi. Hal ini bisa dilakukan dengan menggunakan fungsi `getline`:

```
string s;
getline(cin, s);
```

Jika jumlah data tidak diketahui, loop berikut berguna:

```
while (cin >> x) {
    // kode
}
```

Loop ini membaca elemen-elemen dari input satu per satu, hingga tidak ada lagi data yang tersedia di input.

Dalam beberapa sistem kontes, file digunakan untuk input dan output. Solusi yang mudah untuk ini adalah dengan menulis kode seperti biasa menggunakan aliran standar, tetapi menambahkan baris berikut di awal kode:

```
freopen("input.txt", "r", stdin);
freopen("output.txt", "w", stdout);
```

Setelah ini, program akan membaca input dari file "input.txt" dan menulis output ke file "output.txt".

Bekerja dengan Angka

Integers / Bilangan Bulat

Tipe bilangan bulat yang paling sering digunakan dalam competitive programming adalah `int`, yang merupakan tipe 32-bit dengan rentang nilai -2^{31} ... $2^{31}-1$ atau sekitar $-2 \cdot 10^9$ ... $2 \cdot 10^9$. Jika tipe `int` tidak cukup, maka tipe 64-bit `long long` dapat digunakan. Tipe ini memiliki rentang nilai -2^{63} ... $2^{63}-1$ atau sekitar $-9 \cdot 10^8$ ... $9 \cdot 10^{18}$.

Kode berikut mendefinisikan sebuah variabel long long:

```
long long x = 123456789123456789LL;
```

Akhiran `LL` menunjukkan bahwa tipe angka tersebut adalah `long long`.

Kesalahan umum saat menggunakan tipe `long long` adalah masih menggunakan tipe `int` di beberapa bagian kode. Misalnya, kode berikut mengandung kesalahan halus:

```
int a = 123456789;
long long b = a * a;
cout << b << "\n"; // Output: -1757895751 (salah!)
```

Meskipun variabel `b` bertipe `long long`, kedua angka dalam ekspresi `a*a` bertipe `int`, sehingga hasilnya juga bertipe `int`. Akibatnya, variabel `b` akan berisi hasil yang salah.

Masalah ini dapat diselesaikan dengan mengubah tipe `a` menjadi `long long` atau dengan mengubah ekspresi menjadi `(long long)a*a`, seperti berikut:

```
long long a = 123456789;
long long b = a * a;
cout << b << "\n"; // Output yang benar
```

Atau:

```
int a = 123456789;
long long b = (long long)a * a;
cout << b << "\n"; // Output yang benar
```

Biasanya, soal dalam kompetisi telah disusun sedemikian rupa sehingga tipe `long long` sudah cukup untuk menangani angka besar. Namun, perlu diketahui bahwa kompiler g++ juga menyediakan tipe 128-bit `__int128_t` dengan rentang nilai -2^{127} ... 2^{127} atau sekitar -10^{38} ... 10^{38}.

Meskipun demikian, tipe `__int128_t` tidak tersedia di semua sistem kontes.

Aritmetika Modular

Kita menyatakan x mod m sebagai sisa pembagian x oleh m. Sebagai contoh, 17 mod 5 = 2, karena $17 = 3 \cdot 5 + 2$.

Dalam beberapa soal, jawaban yang dihasilkan bisa berupa angka yang sangat besar. Namun, cukup mencetak hasilnya dalam bentuk "modulo m", yaitu sisa hasil bagi ketika jawaban dibagi dengan m. (misalnya, "modulo $10^9 + 7$").

Dengan cara ini, meskipun jawaban sebenarnya sangat besar, kita hanya perlu menggunakan tipe data `int` atau `long long`.

Salah satu sifat penting dari operasi modulus adalah bahwa dalam operasi **penjumlahan**, **pengurangan**, dan **perkalian**, kita bisa mengambil sisa hasil bagi sebelum melakukan operasi tersebut:

$$(a + b) mod\ m = (a\ mod\ m + b\ mod\ m) mod\ m$$

$$(a - b) mod\ m = (a\ mod\ m - b\ mod\ m) mod\ m$$

$$(a.b) mod\ m = (a\ mod\ m.\ b\ mod\ m) mod\ m$$

Dengan cara ini, kita dapat mengambil modulus setelah setiap operasi agar angka tidak menjadi terlalu besar.

Sebagai contoh, kode berikut menghitung faktorial dari n, yaitu $n!$, dalam modulo m:

```
long long x = 1;
for (int i = 2; i <= n; i++) {
    x = (x * i) % m;
}
cout << x % m << "\n";
```

Biasanya, kita ingin sisa hasil bagi selalu berada dalam rentang $0 \ldots m - 1$. Namun, dalam C++ dan bahasa pemrograman lain, sisa dari bilangan negatif bisa bernilai nol atau negatif. Cara mudah untuk memastikan hasilnya selalu positif adalah dengan terlebih dahulu menghitung modulus seperti biasa, lalu menambahkan m jika hasilnya negatif:

```
x = x % m;
if (x < 0) x += m;
```

Namun, ini hanya diperlukan ketika ada operasi **pengurangan** dalam kode yang dapat menyebabkan hasil negatif.

Bilangan Floating Point

Tipe data floating point yang umum digunakan dalam competitive programming adalah `double` (64-bit) dan, sebagai ekstensi pada kompiler g++, `long double` (80-bit). Dalam kebanyakan kasus, `double` sudah cukup, tetapi `long double` memiliki presisi yang lebih tinggi.

Biasanya, soal akan menentukan tingkat presisi yang diperlukan dalam jawaban. Cara mudah untuk mencetak angka dengan presisi tertentu adalah menggunakan fungsi `printf` dan menentukan jumlah angka desimal dalam format string. Sebagai contoh, kode berikut mencetak nilai x dengan 9 angka desimal:

```
printf("%.9f\n", x);
```

Salah satu kesulitan dalam menggunakan floating point adalah bahwa **beberapa angka tidak dapat direpresentasikan dengan presisi sempurna**, sehingga akan terjadi **kesalahan pembulatan**. Misalnya, perhatikan hasil berikut:

```
double x = 0.3 * 3 + 0.1;
printf("%.20f\n", x); // 0.99999999999999988898
```

Hasilnya mengejutkan karena nilai x ternyata sedikit lebih kecil dari **1**, padahal secara matematis hasil yang benar adalah **1**.

Perbandingan Bilangan Floating Point

Karena adanya **kesalahan pembulatan**, **membandingkan floating point menggunakan operator == sangat berisiko**. Dua angka yang seharusnya sama bisa dianggap berbeda karena perbedaan kecil dalam representasi biner. Solusi yang lebih baik adalah menganggap dua angka sama jika perbedaan absolutnya lebih kecil dari ε **(epsilon)**, yaitu angka kecil yang mewakili batas toleransi kesalahan.

Sebagai contoh, untuk $(\varepsilon = 10^{-9})$, kita dapat membandingkan dua angka sebagai berikut:

```
if (abs(a - b) < 1e-9) {
    // a dan b dianggap sama
}
```

Meskipun bilangan floating point memiliki keterbatasan presisi, integer hingga batas tertentu masih bisa direpresentasikan dengan akurat. Sebagai contoh, dengan double, kita bisa merepresentasikan semua bilangan bulat dengan nilai absolut hingga 2^{53} secara akurat.

Memperpendek Kode

Kode yang ringkas sangat ideal dalam competitive programming, karena program harus ditulis secepat mungkin. Oleh karena itu, banyak programmer kompetitif menggunakan nama yang lebih pendek untuk tipe data dan bagian kode lainnya.

Menyingkat Nama Tipe Data

Menggunakan perintah `typedef`, kita dapat memberi nama yang lebih pendek untuk sebuah tipe data. Sebagai contoh, tipe `long long` cukup panjang, sehingga kita dapat mendefinisikan nama **lebih pendek** seperti `ll`:

```
typedef long long ll;
```

Setelah ini, kode berikut:

```
long long a = 123456789;
long long b = 987654321;
cout << a * b << "\n";
```

dapat diperpendek menjadi:

```
ll a = 123456789;
ll b = 987654321;
cout << a * b << "\n";
```

Selain itu, `typedef` juga bisa digunakan untuk **tipe data yang lebih kompleks**. Contohnya:

```
typedef vector<int> vi;
typedef pair<int, int> pi;
```

Sekarang, kita bisa menulis `vi` untuk `vector<int>` dan `pi` untuk `pair<int, int>`.

Menggunakan Macro

Cara lain untuk memperpendek kode adalah dengan **mendefinisikan macro**. **Macro** adalah bagian kode yang akan **diganti sebelum kompilasi**, menggunakan perintah `#define`.

Sebagai contoh, kita dapat mendefinisikan beberapa macro berikut:

```
#define F first
#define S second
#define PB push_back
#define MP make_pair
```

Setelah ini, kode berikut:

```
v.push_back(make_pair(y1, x1));
v.push_back(make_pair(y2, x2));
int d = v[i].first + v[i].second;
```

dapat diperpendek menjadi:

```
v.PB(MP(y1, x1));
v.PB(MP(y2, x2));
int d = v[i].F + v[i].S;
```

Macro dengan Parameter

Macro juga bisa memiliki **parameter**, sehingga dapat digunakan untuk memperpendek loop atau struktur lainnya. Sebagai contoh, kita bisa mendefinisikan macro untuk **loop** sebagai berikut:

```
#define REP(i, a, b) for (int i = a; i <= b; i++)
```

Setelah ini, kode:

```
for (int i = 1; i <= n; i++) {
    search(i);
}
```

dapat diperpendek menjadi:

```
REP(i, 1, n) {
    search(i);
}
```

Potensi Bug pada Macro

Kadang-kadang, macro dapat menyebabkan bug yang sulit dideteksi. Misalnya, perhatikan macro berikut yang menghitung kuadrat dari sebuah angka:

```
#define SQ(a) a * a
```

Jika kita menulis:

```
cout << SQ(3 + 3) << "\n";
```

Maka kode ini akan diterjemahkan menjadi:

```
cout << 3 + 3 * 3 + 3 << "\n"; // 15 (SALAH!)
```

Karena perkalian memiliki **prioritas lebih tinggi** daripada penjumlahan, hasilnya menjadi salah. Solusi yang lebih baik adalah dengan menambahkan **kurung** pada ekspresi:

```
#define SQ(a) (a) * (a)
```

Sekarang, jika kita menjalankan:

```
cout << SQ(3 + 3) << "\n";
```

maka kode akan diterjemahkan dengan benar menjadi:

```
cout << (3 + 3) * (3 + 3) << "\n"; // 36 (BENAR)
```

Matematika dalam Competitive Programming

Matematika memainkan **peran penting** dalam **competitive programming**, dan **tidak mungkin** menjadi programmer kompetitif yang sukses tanpa memiliki keterampilan matematika yang baik. Bagian ini membahas beberapa **konsep dan rumus matematika penting** yang akan digunakan dalam pembahasan selanjutnya.

Rumus Penjumlahan

Setiap jumlah dalam bentuk berikut:

$$\sum_{x=1}^{n} x^k = 1^k + 2^k + 3^k + \cdots + n^k,$$

Dimana k adalah bilangan bulat positif, memiliki rumus bentuk tertutup yang merupakan polinomial berderajat $k + 1$. Sebagai contoh:

$$\sum_{x=1}^{n} x = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

dan

$$\sum_{x=1}^{n} x^2 = 1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

Barisan aritmetika adalah sebuah urutan bilangan di mana selisih antara dua bilangan yang berurutan selalu konstan. Sebagai contoh,

$$3, 7, 11, 15$$

adalah sebuah barisan aritmetika dengan selisih konstan 4. Jumlah dari sebuah barisan aritmetika dapat dihitung menggunakan rumus

$$\underbrace{a + \cdots + b}_{n \text{ numbers}} = \frac{n(a+b)}{2}$$

di mana a adalah angka pertama, b adalah angka terakhir, dan n adalah jumlah angka. Sebagai contoh,

$$3 + 7 + 11 + 15 = \frac{4.(3 + 15)}{2} = 36$$

Formula ini didasarkan pada fakta bahwa jumlah terdiri dari *n* angka dan nilai setiap angka rata-rata adalah $(a + b)/2$.

Barisan geometri adalah urutan angka di mana rasio antara dua angka berturut-turut adalah konstan. Sebagai contoh,

$$3, 6, 12, 24$$

adalah barisan geometri dengan konstanta 2. Jumlah dari suatu barisan geometri dapat dihitung menggunakan rumus berikut:

$$a + ak + ak^2 + \cdots + b = \frac{bk - a}{k - 1}$$

di mana a adalah bilangan pertama, b adalah bilangan terakhir, dan rasio antara dua bilangan yang berurutan adalah k. Sebagai contoh,

$$3 + 6 + 12 + 24 = \frac{24 \cdot 2 - 3}{2 - 1} = 45.$$

Rumus ini dapat diturunkan sebagai berikut. Misalkan

$$S = a + ak + ak^2 + \cdots + b.$$

Dengan mengalikan kedua sisi dengan k, kita mendapatkan

$$kS = ak + ak^2 + ak^3 + \cdots + bk,$$

dan menyelesaikan persamaannya.

$$kS - S = bk - a$$

menghasilkan rumus.

Kasus khusus dari jumlah deret geometri adalah rumus

$$1 + 2 + 4 + 8 + \cdots + 2^{n-1} = 2^n - 1.$$

Jumlah harmonik atau **harmonic sum** adalah jumlah dalam bentuk

$$\sum_{x=1}^{n} \frac{1}{x} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

Batas atas untuk jumlah harmonik adalah $log_2(n) + 1$. Secara khusus, kita dapat memodifikasi setiap suku $1/k$ sehingga k menjadi pangkat dua terdekat yang tidak melebihi k. Sebagai contoh, ketika $n = 6$, kita dapat memperkirakan jumlahnya sebagai berikut:

$$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} \leq 1 + \frac{1}{2} + \frac{1}{2} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4}.$$

Batas atas ini terdiri dari $log_2(n) + 1$ bagian ($1, 2 \cdot {}^1/_2, 4 \cdot {}^1/_4$, dst.), dan nilai dari setiap bagian paling banyak 1.

Set Theory / Teori Himpunan

Sebuah **himpunan** adalah kumpulan elemen. Sebagai contoh, himpunan

$$X = \{2,4,7\}$$

mengandung elemen 2, 4, dan 7. Simbol $\emptyset$ melambangkan himpunan kosong, dan $|S|$ melambangkan ukuran himpunan S, yaitu jumlah elemen dalam himpunan tersebut. Sebagai contoh, dalam himpunan di atas, $|X| = 3$.

Jika sebuah himpunan S mengandung elemen x, kita menuliskannya sebagai $x \in S$, dan jika tidak, kita menuliskannya sebagai $x \notin S$. Sebagai contoh, dalam himpunan di atas:

$$4 \in X \quad \text{dan} \quad 5 \notin X$$

Himpunan baru dapat dibentuk menggunakan operasi himpunan berikut:

- **Irisan $A \cap B$** terdiri dari elemen-elemen yang terdapat di **kedua** himpunan A dan B. Sebagai contoh, jika $A = \{1,2,5\}$ dan $B = \{2,4\}$, maka $A \cap B = \{2\}$.
- **Gabungan $A \cup B$** terdiri dari elemen-elemen yang terdapat di **A atau B atau keduanya**. Sebagai contoh, jika $A = \{3,7\}$ dan $B = \{2,3,8\}$, maka $A \cup B = \{2,3,7,8\}$.
- **Komplemen $\bar{A}$** terdiri dari elemen-elemen yang **tidak** berada dalam A. Interpretasi dari komplemen tergantung pada **himpunan semesta**, yaitu himpunan yang berisi semua elemen yang mungkin ada. Sebagai contoh, jika $A = \{1,2,5,7\}$ dan himpunan semesta adalah $\{1,2, \dots ,10\}$, maka $\bar{A} = \{3,4,6,8,9,10\}$.
- **Selisih $A \backslash B = A \cap \bar{B}$** terdiri dari elemen-elemen yang berada di **A tetapi tidak di B**. Perhatikan bahwa B bisa saja memiliki elemen yang tidak ada di A. Sebagai contoh, jika $A = \{2,3,7,8\}$ dan $B = \{3,5,8\}$, maka $A \backslash B = \{2,7\}$

Jika setiap elemen dalam A juga termasuk dalam S, kita mengatakan bahwa A adalah **himpunan bagian** dari S, yang dilambangkan dengan $A \subset S$. Sebuah himpunan S selalu memiliki $2^{|S|}$ himpunan bagian, termasuk himpunan kosong. Sebagai contoh, himpunan bagian dari $\{2,4,7\}$ adalah:

$$\emptyset, \{2\}, \{4\}, \{7\}, \{2,4\}, \{2,7\}, \{4,7\} \quad \text{dan} \quad \{2,4,7\}$$

Beberapa himpunan yang sering digunakan adalah:

- $\mathbb{N}$ (bilangan asli),
- $\mathbb{Z}$ (bilangan bulat),
- $\mathbb{Q}$ (bilangan rasional), dan
- $\mathbb{R}$ (bilangan riil).

Himpunan $\mathbb{N}$ dapat didefinisikan dalam dua cara, tergantung pada konteksnya: $\mathbb{N} = \{0,1,2, \dots\}$ atau $\mathbb{N} = \{1,2,3, \dots\}$. Kita juga dapat membentuk sebuah himpunan menggunakan suatu aturan dalam bentuk:

$$\{f(n) : n \in S\}$$

di mana $f(n)$ adalah suatu fungsi. Himpunan ini berisi semua elemen yang berbentuk $f(n)$, di mana n adalah elemen dalam S. Sebagai contoh, himpunan

$$X = \{2n : n \in \mathbb{Z}\}$$

berisi semua bilangan bulat genap.

Logika

Nilai dari sebuah **ekspresi logika** adalah **benar (1)** atau **salah (0)**. Operator logika yang paling penting adalah:

- ¬ (negasi),
- ∧ (konjungsi),
- ∨ (disjungsi),
- ⇒ (implikasi), dan
- ⇔ (ekuivalensi).

Tabel berikut menunjukkan makna dari operator-operator tersebut:

A	B	$\neg A$	$\neg B$	$A \wedge B$	$A \vee B$	$A \Rightarrow B$	$A \Leftrightarrow B$
0	0	1	1	0	0	1	1
0	1	1	0	0	1	1	0
1	0	0	1	0	1	0	0
1	1	0	0	1	1	1	1

Ekspresi $\neg A$ memiliki nilai yang **berlawanan** dengan A. Ekspresi $A \wedge B$ bernilai **benar** jika **kedua** A dan B bernilai benar. Ekspresi $A \vee B$ bernilai **benar** jika **salah satu atau kedua** A dan B bernilai benar. Ekspresi $A \Rightarrow B$ bernilai **benar** jika **setiap kali** A bernilai benar, maka B juga bernilai benar. Ekspresi $A \Leftrightarrow B$ bernilai **benar** jika A dan B **keduanya benar atau keduanya salah**.

Sebuah **predikat** adalah ekspresi yang bernilai **benar atau salah** tergantung pada parameternya. Predikat biasanya dilambangkan dengan huruf kapital. Sebagai contoh, kita dapat mendefinisikan predikat $P(x)$ yang bernilai **benar** jika dan hanya jika x adalah **bilangan prima**.

Dengan definisi ini:

- $P(7)$ bernilai **benar**, karena 7 adalah bilangan prima.
- $P(8)$ bernilai **salah**, karena 8 bukan bilangan prima.

Kuantor menghubungkan ekspresi logika dengan elemen dalam suatu himpunan. Kuantor yang paling penting adalah:

- ∀ (**untuk semua**), dan
- ∃ (**ada**).

Sebagai contoh:

$$\forall x(\exists y(y < x))$$

berarti bahwa untuk setiap elemen x dalam himpunan, **ada** elemen y dalam himpunan tersebut sehingga y lebih kecil dari x. Pernyataan ini **benar** dalam himpunan bilangan bulat ($\mathbb{Z}$), karena setiap bilangan bulat memiliki bilangan yang lebih kecil darinya.

Namun, pernyataan ini **salah** dalam himpunan bilangan asli ($\mathbb{N}$), karena dalam definisi $\mathbb{N} = \{0,1,2,\dots\}$ atau $\mathbb{N} = \{1,2,3,\dots\}$, tidak ada bilangan yang lebih kecil dari bilangan terkecil dalam himpunan tersebut.

Dengan menggunakan notasi yang telah dijelaskan sebelumnya, kita dapat mengekspresikan berbagai jenis **proposisi logika**. Sebagai contoh:

$$\forall x((x > 1 \land \neg P(x)) \Rightarrow (\exists a(\exists b(a > 1 \land b > 1 \land x = ab))))$$

Proposisi ini menyatakan bahwa jika suatu bilangan x lebih besar dari 1 dan bukan bilangan prima, maka ada bilangan a dan b yang lebih besar dari 1 sehingga hasil perkalian mereka adalah x.

Pernyataan ini benar dalam himpunan bilangan bulat (integer), karena setiap bilangan komposit (bilangan bukan prima yang lebih besar dari 1) dapat dinyatakan sebagai hasil kali dua bilangan yang lebih besar dari 1.

Fungsi

Fungsi $\lfloor x \rfloor$ membulatkan bilangan x **ke bawah** menjadi bilangan bulat, sedangkan fungsi $\lceil x \rceil$ membulatkan bilangan x **ke atas** menjadi bilangan bulat. Sebagai contoh:

$$\lfloor 3/2 \rfloor = 1 \quad \text{dan} \quad \lceil 3/2 \rceil = 2$$

Fungsi $\min(x_1, x_2, \dots, x_n)$ dan $\max(x_1, x_2, \dots, x_n)$ memberikan nilai terkecil dan terbesar dari himpunan bilangan $x_1, x_2, \dots, x_n$.

Sebagai contoh:

$$\min(1,2,3) = 1 \quad \text{dan} \quad \max(1,2,3) = 3$$

Faktorial

Faktorial $n!$ didefinisikan sebagai hasil kali semua bilangan bulat positif dari 1 hingga n:

$$\prod_{x=1}^{n} x = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$$

Atau secara rekursif:

$$0! = 1$$

$$n! = n \cdot (n-1)!$$

Bilangan Fibonacci

Bilangan Fibonacci muncul dalam berbagai situasi dan dapat didefinisikan secara rekursif sebagai berikut:

$$f(0) = 0$$

$$f(1) = 1$$

$$f(n) = f(n-1) + f(n-2)$$

Bilangan Fibonacci pertama adalah:

$$0,1,1,2,3,5,8,13,21,34,55, \ldots$$

Ada juga **rumus tertutup** untuk menghitung bilangan Fibonacci, yang sering disebut **rumus Binet**:

$$f(n) = \frac{(1+\sqrt{5})^n - (1-\sqrt{5})^n}{2^n\sqrt{5}}$$

Logaritma

Logaritma dari suatu bilangan x dilambangkan sebagai $log_k(x)$, di mana k adalah **basis** logaritma. Menurut definisi:

$$log_k(x) = a \text{ jika dan hanya jika } k^a = x$$

Salah satu sifat berguna dari logaritma adalah bahwa $log_k(x)$ menyatakan **jumlah pembagian x** dengan k hingga mencapai 1. Sebagai contoh, $log_2(32) = 5$, karena diperlukan **5 kali pembagian** dengan 2 untuk mencapai 1:

$$32 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$$

Penggunaan Logaritma dalam Analisis Algoritma

Logaritma sering digunakan dalam **analisis algoritma**, karena banyak algoritma efisien yang membagi sesuatu menjadi setengahnya di setiap langkah. Oleh karena itu, efisiensi algoritma tersebut dapat diestimasi menggunakan logaritma.

Properti Dasar Logaritma:

1. Logaritma dari Perkalian

$$log_k(ab) = log_k(a) + log_k(b)$$

2. Logaritma dari Pangkat

$$log_k(x^n) = n \cdot log_k(x)$$

3. Logaritma dari Pembagian

$$log_k\left(\frac{a}{b}\right) = log_k(a) - log_k(b)$$

4. Perubahan Basis Logaritma

$$log_u(x) = \frac{log_k(x)}{log_k(u)}$$

Dengan rumus ini, kita dapat menghitung logaritma dengan **basis apa pun**, asalkan kita memiliki metode untuk menghitung logaritma dengan **basis tetap**.

Logaritma Natural

Logaritma natural $\ln(x)$ adalah logaritma dengan **basis** $e \approx 2.71828$e.

Jumlah Digit dalam Basis b

Jumlah digit dari bilangan bulat x dalam basis b diberikan oleh rumus:

$$\lfloor log_b(x) + 1 \rfloor$$

Sebagai contoh, representasi bilangan **123 dalam basis 2** adalah **1111011**. Jumlah digitnya dapat dihitung sebagai $\lfloor log_2(123) + 1 \rfloor = 7$

Kontes dan Sumber Daya

IOI

International Olympiad in Informatics (IOI) adalah kompetisi pemrograman tahunan untuk siswa sekolah menengah. Setiap negara diperbolehkan mengirimkan tim beranggotakan empat siswa ke kompetisi ini. Biasanya, ada sekitar 300 peserta dari 80 negara.

IOI terdiri dari dua babak kontes, masing-masing berdurasi lima jam. Dalam setiap babak, peserta diminta untuk menyelesaikan tiga soal algoritma dengan tingkat kesulitan yang bervariasi. Setiap soal dibagi menjadi beberapa subsoal, masing-masing dengan skor tertentu. Meskipun peserta dibagi ke dalam tim berdasarkan negara, mereka tetap berkompetisi secara individu.

Silabus IOI mengatur topik-topik yang dapat muncul dalam soal IOI. Hampir semua topik dalam **silabus IOI** dicakup oleh buku ini.

Peserta IOI dipilih melalui **kompetisi nasional** di masing-masing negara. Sebelum IOI, terdapat berbagai **kompetisi regional**, seperti Baltic Olympiad in Informatics (BOI), Central European Olympiad in Informatics (CEOI), dan Asia-Pacific Informatics Olympiad (APIO)

Beberapa negara juga menyelenggarakan kontes latihan daring untuk calon peserta IOI, seperti Croatian Open Competition in Informatics, dan USA Computing Olympiad (USACO).

Selain itu, terdapat koleksi besar soal dari kompetisi Polandia yang tersedia secara daring.

ICPC

International Collegiate Programming Contest (ICPC) adalah kompetisi pemrograman tahunan untuk mahasiswa. Setiap tim dalam kompetisi ini terdiri dari tiga mahasiswa, dan berbeda dengan IOI, mereka bekerja sama; hanya satu komputer yang tersedia untuk setiap tim.

ICPC terdiri dari beberapa tahap seleksi, dan tim-tim terbaik akhirnya diundang ke World Finals. Meskipun ada puluhan ribu peserta dalam kompetisi ini, hanya sejumlah kecil tim yang berhasil mencapai babak final. Oleh karena itu, lolos ke final sudah menjadi pencapaian besar di beberapa wilayah.

Dalam setiap kontes ICPC, tim memiliki waktu lima jam untuk menyelesaikan sekitar sepuluh soal algoritma. Solusi hanya diterima jika berhasil menyelesaikan semua kasus uji secara efisien.

Selama kontes berlangsung, peserta dapat melihat peringkat dan hasil tim lain, tetapi pada satu jam terakhir, scoreboard dibekukan sehingga hasil dari pengiriman kode terbaru tidak dapat dilihat.

Topik yang dapat muncul di ICPC tidak ditentukan secara spesifik seperti di IOI. Namun, jelas bahwa dibutuhkan lebih banyak pengetahuan di ICPC, terutama kemampuan matematika yang lebih tinggi.

Kontes Daring

Terdapat banyak **kontes daring** yang terbuka untuk semua orang. Saat ini, situs kontes yang paling aktif adalah **Codeforces**, yang menyelenggarakan kontes hampir **setiap minggu**.

Di Codeforces, peserta dibagi menjadi dua divisi:

- **Div2** untuk pemula
- **Div1** untuk programmer yang lebih berpengalaman

Situs lain yang juga menyelenggarakan kontes pemrograman adalah:

- **AtCoder**
- **CS Academy**
- **HackerRank**
- **Topcoder**

Beberapa perusahaan juga menyelenggarakan **kontes daring dengan final langsung (onsite)**. Contohnya:

- **Facebook Hacker Cup**
- **Google Code Jam**
- **Yandex.Algorithm**

Tentu saja, perusahaan juga menggunakan kontes ini untuk **rekrutmen**. Berhasil tampil baik dalam kompetisi adalah **cara yang bagus untuk membuktikan kemampuan seseorang**.

Buku

Selain buku ini, sudah ada beberapa buku lain yang berfokus pada **competitive programming** dan **penyelesaian masalah algoritmik**:

- **S. S. Skiena dan M. A. Revilla**: *Programming Challenges: The Programming Contest Training Manual*
- **S. Halim dan F. Halim**: *Competitive Programming 3: The New Lower Bound of Programming Contests*
- **K. Diks et al.**: *Looking for a Challenge? The Ultimate Problem Set from the University of Warsaw Programming Competitions*

Dua buku pertama ditujukan untuk **pemula**, sedangkan buku terakhir berisi **materi lanjutan**.

Selain itu, buku algoritma **umum** juga cocok untuk **competitive programmers**. Beberapa buku populer dalam kategori ini adalah:

- **T. H. Cormen, C. E. Leiserson, R. L. Rivest, dan C. Stein**: *Introduction to Algorithms*
- **J. Kleinberg dan É. Tardos**: *Algorithm Design*
- **S. S. Skiena**: *The Algorithm Design Manual*