Matematika Diskret Dasar

Tim Olimpiade Komputer Indonesia

Pendahuluan

Melalui dokumen ini, kalian akan:

- Mempelajari **aritmetika modular**.
- Mempelajari bilangan prima dan **uji keprimaan**.
- Mempelajari algoritma **prime generation**.
- Memahami FPB dan KPK.
- Mempelajari dan memanfaatkan **pigeonhole principle**.

Matematika Diskret

- Merupakan cabang matematika yang mempelajari tentang sifat bilangan bulat, logika, kombinatorika, dan graf.

Bagian 1

Arimetika Modular

Konsep Modulo

- Operasi a mod m biasa disebut "a modulo m".
- Operasi modulo ini akan memberikan sisa hasil bagi a oleh m.
- Contoh:
  - $5 \bmod 3 = 2$
  - $10 \bmod 2 = 0$
  - $21 \bmod 6 = 3$

Sifat-Sifat Modulo

- $(A + B) \bmod M = ((A \bmod M) + (B \bmod M)) \bmod M$
- $(A - B) \bmod M = ((A \bmod M) - (B \bmod M)) \bmod M$
- $(A \times B) \bmod M = ((A \bmod M) \times (B \bmod M)) \bmod M$
- $(A^B) \bmod M = ((A \bmod M)^B) \bmod M$

Aplikasi Modulo

- Pada pemrograman kompetitif, tidak jarang kita harus menghitung $n!$ mod k (terutama dalam kombinatorika).
- Seandainya kita menghitung $n!$, kemudian menggunakan operasi mod, kemungkinan besar kita akan mendapatkan *integer overflow*.
- Untungnya, kita dapat memanfaatkan sifat modulo.

Aplikasi Modulo (lanj.)

Solusi:

```
MODULARFACTORIAL(n, k)
1   result = 1
2   for i = 1 to n
3       result = (result × i) mod k
4   return result
```

Hati-Hati!

- Aritmetika modular tidak serta-merta bekerja pada pembagian.
- $\frac{a}{b} \mod n \neq \left(\frac{a \bmod n}{b \bmod n}\right) \mod n$.
- Contoh: $\frac{12}{4} \mod 6 \neq \left(\frac{12 \bmod 6}{4 \bmod 6}\right) \mod 6$.

- Pada aritmetika modular, $\frac{a}{b} \mod n$ biasa ditulis sebagai:

$$a \times b^{-1} \mod n$$

  dengan b^{-1} adalah **modular multiplicative inverse** dari b.
- Jika tertarik, Anda bisa mempelajari *modular multiplicative inverse* melalui tautan Wikipedia ini.

Bagian 2

Bilangan Prima

Konsep Bilangan Prima

Bilangan Prima

Bilangan bulat positif yang tepat memiliki dua faktor (pembagi), yaitu 1 dan dirinya sendiri.
Contoh: $2, 3, 5, 13, 97$.

Bilangan Komposit

Bilangan yang memiliki lebih dari dua faktor.
Contoh: $6, 14, 20, 25$.

Uji Keprimaan

- **Uji keprimaan** (*primality testing*) adalah algoritma untuk mengecek apakah suatu bilangan bulat N adalah bilangan prima.
- Kita dapat memanfaatkan sifat bilangan prima yang disebutkan pada slide sebelumnya untuk mengecek apakah suatu bilangan merupakan suatu bilangan prima.

Uji Keprimaan (lanj.)

- Solusi yang mudah untuk mengecek apakah N prima atau tidak tentu dengan mengecek apakah ada bilangan selain 1 dan N yang habis membagi N.
- Maka, kita dapat melakukan iterasi dari 2 hingga $N - 1$ untuk mengetahui apakah ada bilangan selain 1 dan N yang habis membagi N.
- Kompleksitas: $O(N)$.

Uji Keprimaan (lanj.)

```
isPrimeNaive(n)
1   if n ≤ 1
2         return false

3   for i = 2 to n − 1
4         if n mod i == 0
5               return false

6   return true
```

Uji Keprimaan (lanj.)

- Ada solusi yang lebih cepat dari $O(N)$.
- Manfaatkan observasi bahwa jika $N = a \times b$, dan $a \leq b$, maka $a \leq \sqrt{N}$ dan $b \geq \sqrt{N}$.
- Kita tidak perlu memeriksa b; seandainya N habis dibagi b, tentu N habis dibagi a.
- Jadi kita hanya perlu memeriksa hingga $\sqrt{N}$.
- Kompleksitas: $O(\sqrt{N})$

Uji Keprimaan (lanj.)

```
isPrimeSqrt(n)
1   if n ≤ 1
2         return false

3   i = 2
4   while i × i ≤ n
5         if n mod i == 0
6               return false
7         i = i + 1

8   return true
```

Bagian 3

Prime Generation (Pembangkitan Bilangan Prima)

Solusi Naif

- Misalkan kita ingin mengetahui himpunan bilangan prima yang tidak lebih dari N.
- Kita dapat membangkitkan bilangan prima dengan iterasi dan uji keprimaan.
- Solusi:
  1. Lakukan iterasi dari 2 sampai N
  2. Untuk tiap bilangan, cek apakah dia bilangan prima. Jika ya, kita bisa memasukkannya ke daftar bilangan prima.

Solusi Naif (lanj.)

```
SIMPLEPRIMEGENERATION(N)
1   primeList = {}
2   for i = 2 to N
3       if ISPRIMESQRT(i)
4           primeList = primeList ∪ {i}
5   return primeList
```

Sieve of Eratosthenes

- Terdapat solusi yang lebih cepat untuk membangkitkan bilangan prima, yaitu **Sieve of Eratosthenes**.
- Ide utama utama dari algoritma ini adalah mengeliminasi bilangan-bilangan dari calon bilangan prima.
- Yang akan kita eliminasi adalah bilangan komposit.

Sieve of Eratosthenes (lanj.)

- Kita tahu bahwa suatu bilangan komposit c dapat dinyatakan sebagai $c = p \times q$, dengan p suatu bilangan prima.
- Seandainya kita mengetahui suatu bilangan prima, kita dapat mengeliminasi kelipatan-kelipatan bilangan tersebut dari calon bilangan prima.
- Contoh: jika diketahui 7 adalah bilangan prima, maka 14, 21, 28, ... dieliminasi dari calon bilangan prima.

Prosedur Sieve of Eratosthenes

1. Awalnya seluruh bilangan bulat dari 2 hingga N belum dieliminasi.
2. Lakukan iterasi dari 2 hingga N:
   - 2.1 Jika bilangan ini belum dieliminasi, artinya bilangan ini merupakan bilangan prima.
   - 2.2 Lakukan iterasi untuk mengeliminasi kelipatan bilangan tersebut.

Implementasi Sieve of Eratosthenes

- Kita dapat menggunakan *array boolean* untuk menyimpan informasi apakah suatu bilangan telah tereliminasi.
- Jika kita ingin mencari bilangan prima yang $\leq N$, maka diperlukan memori sebesar $O(N)$.
- Melalui perhitungan matematis, kompleksitas waktu solusi ini adalah $O(N \; log \; log \; N)$.

Implementasi Sieve of Eratosthenes (lanj.)

sieveOfEratosthenes(N)

```
 1  // Siapkan array boolean eleminated berukuran N
 2  // Inisialisasi array eleminated dengan false
 3  primeList = {}
 4  for i = 2 to n
 5        if not eleminated[i]
 6              primeList = primeList ∪ {i}
 7              j = i × i
 8              while j ≤ n
 9                    eleminated[j] = true
10                    j = j + i
11  return primeList
```

Implementasi Sieve of Eratosthenes (lanj.)

- Perhatikan baris ke-7 yang berisi $j = i \times i$.
- Di sini, j menyatakan kelipatan i yang akan dieliminasi.
- Perhatikan bahwa j dimulai dari $i \times i$, bukan $2i$.
- Alasannya adalah $2i, 3i, 4i, ..., (i-1)i$ pasti sudah tereliminasi pada iterasi-iterasi sebelumnya.

Bagian 4

FPB dan KPK

Faktorisasi Prima

- Ketika masih SD, kita pernah belajar memfaktorkan bilangan dengan pohon faktor.
- Melalui faktorisasi prima, kita dapat menyatakan suatu bilangan sebagai hasil perkalian faktor primanya.
- Contoh: $7875 = 3^2 \times 5^3 \times 7$.

FPB dan KPK

- FPB dan KPK dapat dicari melalui faktorisasi prima.
- Untuk setiap bilangan prima, kita menggunakan pangkat terkecil untuk FPB dan pangkat terbesar untuk KPK.
- Contoh:
  - $4725 = 3^3 \times 5^2 \times 7$
  - $7875 = 3^2 \times 5^3 \times 7$
- Maka:
  - $FPB(4725, 7875) = 3^2 \times 5^2 \times 7 = 1525$
  - $KPK(4725, 7875) = 3^3 \times 5^3 \times 7 = 23625$

- Terdapat pula sifat $KPK(a, b) = \frac{a \times b}{FPB(a,b)}$.

Algoritma Euclid

- Untuk mencari FPB suatu bilangan, menggunakan pohon faktor cukup merepotkan.
- Kita perlu mencari faktor prima bilangan tersebut, dan jika faktor primanya besar, tentu akan menghabiskan banyak waktu.
- Terdapat algoritma yang dapat mencari $FPB(a, b)$ dalam $O(\log(\min(a, b)))$.
- Algoritma ini bernama **Algoritma Euclid**.

Algoritma Euclid (lanj.)

```
EUCLID(a, b)
1  if b == 0
2        return a
3  else
4        return EUCLID(b, a mod b)
```

Algoritma Euclid (lanj.)

- Sangat pendek!
- Anda dapat menghemat waktu pengetikan kode dalam melakukan pencarian FPB dengan algoritma ini.
- Jika Anda tertarik dengan pembuktiannya, baca lebih lanjut di tautan Wikipedia ini.

Bagian 5

Pigeonhole Principle (PHP)

Pigeonhole Principle

- Konsep PHP adalah "Jika ada N ekor burung dan M sangkar, yang memenuhi $N > M$, maka pasti ada sangkar yang berisi setidaknya 2 ekor burung".
- Secara matematis, jika ada N ekor burung dan M sangkar, maka pasti ada sangkar yang berisi setidaknya $\left\lceil \frac{N}{M} \right\rceil$ ekor burung.

Pigeonhole Principle (lanj.)

- Terkesan sederhana?
- Simak contoh aplikasi prinsip ini.

Contoh Soal PHP

- Pak Dengklek memiliki sebuah *array* A berisi N bilangan bulat non-negatif.
- Anda ditantang untuk memilih angka-angka dari *array*-nya yang jika dijumlahkan habis dibagi N.
- Angka di suatu indeks *array* tidak boleh dipilih lebih dari sekali.
- Apabila mungkin, cetak indeks angka-angka yang Anda ambil.
- Apabila tidak mungkin, cetak "`tidak mungkin`".
- Batasan
  - $1 \leq N \leq 10^5$
  - *Array* A hanya berisi bilangan bulat non-negatif.

Analisis Contoh Soal PHP

- Inti soal ini adalah mencari apakah pada *array* berukuran N, terdapat subhimpunan tidak kosong yang jumlahan elemen-elemennya habis dibagi N.

Analisis Contoh Soal PHP (lanj.)

- Mari kita coba mengerjakan versi lebih mudah dari soal ini: Bagaimana jika yang diminta subbarisan, bukan subhimpunan?
- Anggap *array* A dimulai dari indeks 1 (*one-based*).
- Misalkan kita memiliki fungsi $sum(k) = \sum_{i=1}^{k} A[i]$.
- Untuk $sum(0)$, sesuai definisi nilainya adalah 0.

Analisis Contoh Soal PHP (lanj.)

- Perhatikan bahwa $\sum_{i=l}^{r} A[i] = sum(r) - sum(l-1)$.
- Jika subbarisan $A[l..r]$ habis dibagi N, maka $(sum(r) - sum(l-1)) \bmod N = 0$.
- Ini dapat kita tuliskan sebagai $sum(r) \bmod N = sum(l-1) \bmod N$.

Analisis Contoh Soal PHP (lanj.)

- Observasi 1:
  Ada N kemungkinan nilai $(sum(x) \bmod N)$, yaitu $[0..N-1]$.
- Observasi 2:
  Ada $N+1$ nilai x untuk $(sum(x) \bmod N)$, yaitu untuk $x \in [0..N]$.

  Ingat bahwa $sum(0)$ ada agar jumlahan subbarisan $A[1..k]$ untuk tiap k dapat kita nyatakan dalam bentuk: $(sum(k) - sum(0))$.

Analisis Contoh Soal PHP (lanj.)

Observasi 3:

- Ada $N + 1$ kemungkinan nilai x
- Ada N kemungkinan nilai $sum(x)$ mod N
- Pasti ada a dan b, sehingga $sum(b)$ mod $N = sum(a)$ mod N
- Subbarisan yang menjadi solusi adalah $A[a + 1..b]$.

Analisis Contoh Soal PHP (lanj.)

- Dengan menyelesaikan versi mudah dari soal awal kita, ternyata kita justru dapat menyelesaikan soal tersebut.
- Suatu subbarisan dari A pasti juga merupakan subhimpunan dari A.
- Ternyata, selalu ada cara untuk menjawab pertanyaan Pak Dengklek.
- Yakni, keluaran "`tidak mungkin`" tidak akan pernah terjadi.

Implementasi

findDivisibleSubsequence(A, N)

```
1   // Inisialisasi array sum[0..N] dengan 0
2   // Isikan nilai sum[i] dengan (A[1] + A[2] + ... + A[i])
3   // Inisialisasi array seenInIndex[0..N − 1] dengan −1

4   for i = 0 to N
5       if seenInIndex[sum[i] mod N] == −1
6           seenInIndex[sum[i] mod N] = i
7       else
8           a = seenInIndex[sum[i] mod N]
9           b = i
10          return [a + 1, a + 2, ..., b]
```

Penutup

- Matematika diskret merupakan topik yang sangat luas.
- Topik ini berisikan banyak konsep dasar yang umum digunakan pada pemrograman kompetitif.