Implementasi Operasi Matriks Biner dan Algoritma Sliding Window untuk Reverse Engineering Koordinat Spasial Minecraft Java Edition 1.12.2

Axeleon Justin Algianto - 13525074
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13525074@std.stei.itb.ac.id, axeleonjustin148@gmail.com

***Abstrak*— Kerahasiaan lokasi spasial merupakan mekanisme pertahanan krusial pada ekosistem virtual tanpa regulasi seperti server anarki, di mana perlindungan aset bergantung sepenuhnya pada luasnya ruang pencarian. Ruang virtual tersebut dibangun melalui sistem generasi prosedural berbasis Pseudo-Random Number Generator (PRNG), khususnya Linear Congruential Generator (LCG), yang pada arsitektur permainan seperti Minecraft diasumsikan mendistribusikan formasi struktur alam secara acak mutlak. Namun, karena sifat deterministiknya, pola blok bedrock hakikatnya bertindak sebagai fungsi surjektif yang menyimpan jejak koordinat absolut ruang itu sendiri. Makalah ini membuktikan bahwa koordinat tersebut dapat diekstraksi hanya dari tangkapan layar visual dengan memetakan ruang tiga dimensi ke dalam matriks biner, lalu memindai kecocokan pola menggunakan algoritma Sliding Window. Seluruh proses reverse engineering ini dilandasi prinsip Matematika Diskrit, mencakup aljabar Boolean, teori bilangan, dan analisis kompleksitas asimtotik.**

Kata Kunci—Aljabar Boolean; Algoritma Sliding Window; Linear Congruential Generator; Matriks Biner; Kompleksitas Asimtotik; Topologi Digital; Tensor Spasial.

I. PENDAHULUAN

A. Latar Belakang Topologi Ruang Digital dan Entropi Keamanan

Dalam diskursus keamanan siber kontemporer dan kajian topologi ruang digital, fenomena *server* tanpa regulasi (*server anarchy*) seperti 2b2t memberikan landasan studi kasus yang sangat relevan dan belum banyak dieksplorasi secara akademis. Lingkungan *server anarchy* beroperasi sebagai sebuah ekosistem ruang spasial virtual berskala raksasa, sering kali melampaui jutaan unit metrik komputasi, tanpa adanya intervensi administratif, penegakan hukum digital, maupun batasan modifikasi struktur oleh pihak otoritas pengelola *server*. Dalam lingkungan Euclidean virtual yang terisolasi ini, keamanan aset digital murni bergantung pada satu prinsip kriptografis fundamental: entropi lokasi. Selama koordinat spasial absolut (vektor (x, y, z)) tetap menjadi kerahasiaan yang tidak terpublikasi, suatu aset dianggap terenkripsi oleh luasnya ruang pencarian itu sendiri.

Namun, asumsi keamanan berbasis entropi spasial ini memiliki kelemahan matematis yang fatal. Ruang digital pada arsitektur mesin Minecraft dihasilkan secara prosedural menggunakan algoritma yang secara matematis deterministik. Elemen paling dasar dari topologi ini adalah blok *bedrock* (batu dasar tak tertembus) yang berfungsi sebagai pembatas dunia pada sumbu vertikal paling bawah. Formasi tata letak *bedrock* ini dihasilkan secara *pseudo-random* (acak semu). Mengingat *pseudo-random number generator* (PRNG) yang digunakan mesin diinisialisasi (*seeded*) menggunakan sebuah polinomial yang terikat langsung dengan koordinat absolut sumbu horizontal (x dan z), pola susunan *bedrock* tidak lagi bersifat acak murni. Sebaliknya, ia bertindak sebagai "sidik jari" (*spatial fingerprint*) yang secara unik mengidentifikasi setiap vektor posisi di dalam ruang spasial tersebut, seperti yang diilustrasikan pada Gambar 1.

Gambar 1. Formasi blok bedrock pada lantai dunia Minecraft

Mendekonstruksi korelasi antara pola matriks biner *bedrock* dengan fungsi pembangkit acaknya akan memfasilitasi sebuah proses *reverse engineering* di mana konsep keamanan melalui ketidakjelasan (*security through obscurity*) dikompromikan sepenuhnya. Ketika sub-himpunan pola spasial bocor (misalnya melalui tangkapan layar yang menampilkan susunan blok di lantai), sistem komputasi dapat memodelkan ulang generasi prosedural tersebut secara lokal, menggeser jendela pencarian melintasi seluruh probabilitas matriks dunia maya, dan menemukan resolusi koordinat aslinya dengan akurasi mutlak.

B. Rumusan Masalah dan Tujuan Analisis

Berpijak pada landasan fenomenologis tersebut, makalah ini merumuskan sejumlah problematika komputasional fundamental yang dianalisis secara matematis:

- Bagaimana mentransformasikan dan memodelkan ruang spasial *bedrock* tiga dimensi yang ireguler ke dalam bentuk struktur data aljabar Boolean dan matriks biner berdimensi tinggi secara efisien?
- Bagaimana teori bilangan komputasional pada *Linear Congruential Generator* (LCG) yang mendasari mesin Java beroperasi secara deterministik, dan bagaimana kerentanan rumusnya dapat direplikasi untuk memprediksi probabilitas peletakan struktur spasial tanpa intervensi *server* asal?
- Bagaimana kompleksitas algoritma asimtotik dari metode *Sliding Window* dalam melakukan pencarian pola matriks sub-himpunan di dalam ruang pencarian

semesta yang luasnya dapat mencapai orde miliaran komputasi?

Penulisan ini bertujuan untuk memberikan landasan teoretis yang ketat serta pembuktian matematis konkret atas implementasi algoritma pencari koordinat menggunakan bahasa pemrograman Java murni, di mana fokus utuh diarahkan pada evaluasi Matematika Diskrit, reduksi kompleksitas ruang/waktu, dan pemetaan graf.

II. TEORI DASAR

A. Aljabar Boolean dan Pemetaan Matriks Biner

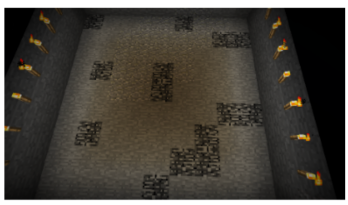
Dalam cabang ilmu Matematika Diskrit, status keberadaan suatu entitas dalam koordinat absolut dapat diabstraksikan menggunakan himpunan aljabar Boolean B = {0, 1}. Pemetaan blok *bedrock* dalam mesin komputasi spasial adalah implementasi langsung dari teori matriks biner multidimensi [1]. Setiap titik koordinat diskrit (x, y, z) dalam ruang Euclidean kartesian $\mathbb{R}^3$ dievaluasi untuk memiliki tepat satu dari dua kemungkinan nilai kebenaran (*state*):

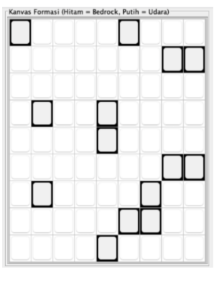
- Nilai 1 (True) merepresentasikan eksistensi material solid (dalam hal ini *bedrock*).
- Nilai 0 (False) merepresentasikan ruang kosong (udara, batuan lunak, atau material lain yang dapat dihancurkan).

Sistem komputasi lingkungan membagi ruang spasial menjadi segmentasi berukuran tetap yang disebut *chunk*, di mana setiap *chunk* adalah prisma segiempat dengan dimensi penampang 16 × 16 pada sumbu horizontal. Lantai dasar dari satu *chunk* pada ketinggian spesifik (y = c) dapat direpresentasikan secara ketat sebagai sebuah matriks persegi berdimensi 16 × 16 yang dilambangkan dengan M, di mana setiap elemen $b_{i,j} \in \{0, 1\}$, sebagaimana ditunjukkan dalam Gambar 2. Karena formasi alamiah *bedrock* memiliki variabilitas ketebalan struktural pada rentang diskrit ketinggian dari 0 hingga 4 secara inklusif, struktur matematis ini melampaui matriks dua dimensi konvensional dan bergeser menjadi sebuah struktur Tensor Biner tiga dimensi T.

Dimensi Spasial	Notasi Matematis	Kardinalitas	Interpretasi
Sumbu Horizontal X	x ∈ [0,15]	16	Lebar chunk
Sumbu Kedalaman Z	z ∈ [0,15]	16	Panjang chunk
Sumbu Vertikal Y (Overworld)	y ∈ [0,4]	5	Tebal bedrock
Tensor Biner Total T	16×16×5	1280	Volume diskrit

Tabel 1. Pemetaan dimensi spasial ke dalam ruang himpunan kardinalitas diskrit dalam satu segmentasi ruang (Chunk).





Gambar 2. Transformasi formasi bedrock ke representasi matriks biner.

B. Teori Bilangan: Linear Congruential Generator (LCG)

Kemampuan untuk memprediksi probabilitas persebaran *bedrock* didasari pada eksploitasi fundamental mesin *Pseudo-Random Number Generator* (PRNG). Ruang virtual ini dibangun pada ekosistem Java, yang secara inheren mengandalkan utilitas java.util.Random [5]. Utilitas ini tidak menghasilkan entropi termodinamika yang murni, melainkan menggunakan relasi rekurensi deterministik yang dikenal sebagai *Linear Congruential Generator* (LCG) [3]. Berdasarkan Teori Bilangan, barisan angka pseudo-acak X_n didefinisikan oleh persamaan kongruensi linear:

$$X_{n+1} = (a \cdot X_n + c) \bmod m \qquad (1)$$

Dalam arsitektur Java 8 yang memvalidasi topologi ini, konstanta matematis telah ditetapkan secara global (*hardcoded*) oleh spesifikasi perangkat lunak [4], [5]:

- Multiplier (a) = 0x5DEECE66D = 25214903917
- Increment (c) = 0xB = 11
- Modulus (m) = 2^{48}

Penggunaan modulus 2^{48} (sekitar 281,4 triliun) membatasi panjang periode siklus LCG pada 2^{48} *state* unik. Angka ini secara kriptografis tergolong sangat kecil dan rentan terhadap iterasi *brute-force* asimtotik [4]. Fakta paling krusial dalam algoritma prosedural ruang ini adalah bahwa nilai awal barisan (*seed*, atau X_0) untuk menghasilkan struktur matriks suatu *chunk* dihitung secara definitif melalui substitusi koordinat absolut (x, z) ke dalam sebuah fungsi polinomial surjektif. Dengan demikian, keberadaan blok pada relasi menjamin bahwa apabila pola biner disuplai, sistem dapat diselesaikan secara komputasi aljabar untuk menemukan kembali koordinat (x, z).

C. Algoritma Sliding Window dan Kompleksitas Asimtotik

Untuk menyelaraskan struktur data matriks yang diperoleh dari hasil rekaman pengguna (target) dengan jutaan probabilitas matriks yang dibangkitkan oleh fungsi LCG, penelitian ini menerapkan algoritma *Sliding Window* [2]. Di dalam ranah Matematika Diskrit, metode ini mendefinisikan sebuah jendela konseptual berdimensi dinamis yang bertranslasi (bergeser) secara berurutan melintasi elemen-elemen matriks semesta observasi untuk menemukan sub-graf atau himpunan bagian yang berelasi ekuivalen.

Misalkan kita mendefinisikan himpunan Target T berupa matriks berdimensi m × n, dan himpunan observasi berukuran raksasa W berdimensi M × N. Fungsi evaluasi kecocokan dioperasikan menggunakan logika konjungsi Boolean (*bitwise AND*) yang disandingkan dengan fungsi bi-implikasi (XNOR) [1]. Himpunan sub-matriks dinyatakan isomorfik sempurna jika fungsi pemetaan relasional mengembalikan nilai *True* mutlak untuk seluruh elemen.

Dari kacamata analisis asimtotik (*Notasi Big-O*), pencarian *Sliding Window* berpotensi memicu ledakan komputasi. Kompleksitas waktu pada skenario terburuk (*worst-case*) adalah O(M · N · m · n) [2]. Tanpa optimasi

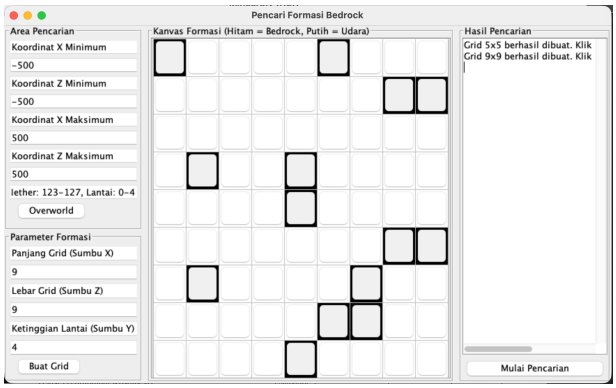
diskrit, iterasi ini akan membebani siklus prosesor secara kuadratik dan dapat mengakibatkan *stack overflow* pada memori konvensional. Pendekatan pereduksian hierarki Big-O dijabarkan dalam tahapan Implementasi.

III. IMPLEMENTASI DAN ANALISIS SISTEM

Bab ini membuktikan kerangka dasar matematis diskrit melalui abstraksi arsitektur komputasi perangkat lunak Java murni. Transformasi dari formulasi aljabar ke dalam paradigma *Object-Oriented Programming* (OOP) menjabarkan fiksasi batasan lapisan tensor, rekonstruksi barisan bilangan LCG, dan pemindaian *sliding window* berdimensi masif secara terstruktur.

A. Representasi Spasial dan Pemetaan Tensor Objek

Model relasional dari sub-ruang target dienkapsulasi menggunakan objek diskrit dalam kelas BlokirKoordinat.java. Konstruksi OOP ini memetakan relasi posisi absolut spasial ke dalam koordinat vektor relatif yang dijangkarkan pada sebuah titik asal sentral (*origin*), dan menyimpan pemetaan biner tipeBlok. Bentuk implementasi masukan pada program ditunjukkan oleh Gambar 3.



Gambar 3. Tampilan GUI Program "Pencari Formasi Bedrock"

Struktur data raksasa untuk memuat probabilitas dunia dialokasikan di dalam kelas DataBedrockChunk.java. Mengingat dimensi satu himpunan *chunk* mencakup volume vertikal dari dasar hingga puncak angkasa (256 blok), memuat keseluruhan matriks akan mendegradasikan *Space Complexity* memori. Melalui analisis domain himpunan fungsi, nilai kebenaran keberadaan *bedrock* di luar $y \in [0,4]$ selalu mengembalikan nilai deterministik 0 (False) (Himpunan Kosong $\emptyset$).

Oleh karena itu, fungsi diskrit dengan batasan kondisional (*piecewise function*) diaplikasikan guna memutus pembacaan di luar domain himpunan tensor:

```
public int getIdBlok(int x, int y, int z) {
    // Evaluasi domain fungsi diskrit kondisional
    if (y < 0 || (y > 4 && y < 123) || y > 127) {
        return 0; // Himpunan kosong
    }
    return dataBlok[x][y][z];
}

public void setIdBlok(int x, int y, int z, int id)
{
    if (y < 0 || y > 4) {
        return; // Mutasi tertolak: batas domain
    }
    dataBlok[x][y][z] = id;
}
```

Potongan Kode 1. Mekanisme batasan asimtotik ketinggian matriks tensor dalam objek DataBedrockChunk.

Potongan Kode 1 memvalidasi batas kardinalitas himpunan. Sebelum program mengeksekusi dereferensi elemen pada array tiga dimensi dataBlok, parameter input y diinspeksi. Jika nilai berada di luar batas kardinalitas yang diakui, pengembalian 0 dieksekusi dalam kompleksitas waktu konstan O(1).

Skenario topologi spasial memuat ranah dimensi alternatif bernama *Nether*, yang menghadirkan plafon *bedrock* pada rentang atas himpunan vertikal. Mekanisme pewarisan OOP (*Inheritance*) digunakan dalam kelas DataBedrockChunkNether.java untuk merepresentasikan fenomena ini tanpa merusak abstraksi induk. Daripada mengalokasikan memori matriks masif yang tak terpakai, algoritma menggunakan konsep pemetaan pergeseran skalar linear (*Linear Offset Mapping*):

```
@Override
public int getIdBlok(int x, int y, int z) {
    // Relasi intersepsi pemetaan ketinggian atap
Nether
    if (y >= 123 && y <= 127) {
        // Translasi rentang 123-127 -> indeks 0-4
        return dataAtap[x][y - 123][z];
    }
    // Delegasi fallback ke induk (Overworld)
    return super.getIdBlok(x, y, z);
}
```

Potongan Kode 2. Modifikasi rentang batas operasi atap menggunakan fungsi aljabar pada DataBedrockChunkNether.

Potongan Kode 2 merupakan sintesis dari efisiensi alokasi memori. Translasi matematis $y \rightarrow y-123$ memadatkan pemetaan ruang. Hal ini mempertahankan *Space Complexity* agar tetap konstan, tidak peduli elevasi spasial sesungguhnya berada pada parameter ratusan.

Properti Sistem	DataBedrockChunk	DataBedrockChunkNether	Transformasi
Domain Tinggi Primer	$y \in [0,4]$	$y \in [0,4]$	Identitas
Domain Tinggi Sekunder	Tak terdefinisi ($\emptyset$)	$y \in [123,127]$	Translasi
Fungsi Substitusi	dataBlok[y]	dataAtap[y−123]	Piecewise OR

Tabel 2. Perbandingan struktural batasan ketinggian dimensi dan translasi matematis pada memori matriks.

B. Rekonstruksi Deterministik Mesin PRNG Java

Fase paling sentral dalam dekonstruksi koordinat spasial adalah memulihkan barisan rekurensi *Linear Congruential Generator* (LCG) agar ekuivalen persis seperti mesin asli pada permainan [3]. Fungsi eksploitasi dalam PencariBedrock.java mencapai hal ini dengan mensimulasikan entropi semu java.util.Random [5]:

```
// Hash Polinomial: (X, Z) -> LCG Seed
rng.setSeed((long)(cx + dx) * 341873128712L
          + (long)(cz + dz) * 132897987541L);

for (int bx = 0; bx < 16; bx++) {
  for (int bz = 0; bz < 16; bz++) {
    rng.nextDouble();
    if (dunia == 0) {
      rng.nextDouble(); rng.nextDouble();
    }
    int yMulai = (dunia == 0) ? 127 : 255;
    for (int by = yMulai; by >= 0; by--) {
      if (dunia == 1 && by <= rng.nextInt(5))
        chunk.setIdBlok(bz, by, bx, 1);
    }
  }
}
```

Potongan Kode 3. Algoritma rekonstruksi ekuivalensi matriks berbasis seed pada generation loop PencariBedrock.

Analisis terhadap Potongan Kode 3 membuktikan proposisi determinisme ruang spasial. Ekspresi (cx+dx) · 341873128712 + (cz+dz) · 132897987541 merupakan perkalian titik dari koordinat skalar absolut (*chunk space*) terhadap konstanta *prime multiplier* 64-bit yang digunakan sebagai *hash* internal. Kondisi *Integer Overflow* yang terjadi bukan merupakan *error* semantik, melainkan sengaja dipertahankan sebagai mekanisme modulus tersembunyi yang mensimulasikan rotasi aljabar ruang [4].

Metode iteratif bersarang memanggil rng.nextInt(5). Di dalam implementasi Java standar, nextInt() mendevaluasi turunan state LCG untuk mengekstraksi entropi acak seragam pada himpunan {0,1,2,3,4} [5]. Angka inilah yang menciptakan topografi lapisan *bedrock* yang ketebalannya bervariasi asimetris. Sistem LCG kemudian "dimajukan" (*advanced*) sejajar dengan pemanggilan nextDouble() demi mereplika persis berapa banyak *state* yang dikonsumsi oleh elemen mesin pembentuk dunia asli. Dari sinilah nilai Boolean eksistensi blok disuntikkan ke dalam matriks target.

C. Pemindaian Sliding Window dan Eviksi Relasional

Dengan tensor spasial yang sukses direkonstruksi, langkah paripurna dari rekayasa balik adalah menelusuri keidentikan pola. Algoritma *Sliding Window* dalam kerangka memori berlebih menimbulkan degradasi; solusi matematis diimplementasikan melalui struktur batas spasial dan eviksi *cache* [2].

```
// Pre-komputasi bentangan Euclidean target
int jarakXMaks = -1, jarakZMaks = -1;
for (BlokirKoordinat target : daftarTarget) {
  if (target.getPosX() > jarakXMaks)
    jarakXMaks = target.getPosX();
  if (target.getPosZ() > jarakZMaks)
    jarakZMaks = target.getPosZ();
}
int chunkLebarX = (int) Math.ceil((double)
   jarakXMaks / 16.0);
int chunkLebarZ = (int) Math.ceil((double)
   jarakZMaks / 16.0);// Cache Eviction (Garbage
   Collection)
for (int n = chunkAktif.size()-1; n>=0; n--) {
  DataBedrockChunk c = chunkAktif.get(n);
  if (c.getChunkX() < cx ||
      c.getChunkX() > cx + chunkLebarX ||
      c.getChunkZ() < cz ||
      c.getChunkZ() > cz + chunkLebarZ)
    chunkAktif.remove(n);
}
```

Potongan Kode 4. Penentuan area dimensi Sliding Window asimtotik dan pembersihan objek memori di PencariBedrock.

Algoritma melakukan *scanning* linear terhadap setiap relasi koordinat target untuk mengekstraksi jarak maksimum Euclidean yang berfungsi sebagai titik tumpu (*pivot*). Selanjutnya, proses *garbage collection* berjalan pada akhir kalang pergeseran: secara kondisional membandingkan (menggunakan logika Disjungsi Boolean ||) posisi relatif dari matriks observasi yang disangga di memori. Jika elemen bergeser keluar dari batas jendela dinamis, objek memori langsung dihancurkan. Fenomena ini memastikan *Space Complexity* tertahan secara konstan tanpa menghiraukan luas semesta observasi.

Untuk proses ekuivalensi, implementasi optimasi fundamental yang disebut evaluasi *Short-Circuit* diterapkan:

```
boolean formasiDitemukan = true;
for (BlokirKoordinat target : daftarTarget) {
  if (target.getTipeBlok() != chunk.getIdBlok(
        (target.getPosX() + bx) % 16,
        yDasar + target.getPosY(),
        (target.getPosZ() + bz) % 16)) {
    formasiDitemukan = false;
    break; // Short-Circuit Evaluasi
  }
}
```

Potongan Kode 5. Komputasi perbandingan ekuivalensi elemen Boolean pada proses iterasi PencariBedrock.

Secara matematis, untuk membuktikan kebenaran sebuah himpunan proposisi konjungtif, seluruh anggotanya harus bernilai "Benar" (1). Apabila pada suatu iterasi ditemukan bahwa target tidak sesuai (*not equal*, !=), proposisi secara instan dievaluasi bernilai "Salah" (0), dan operator break mengeksekusi fungsi putus sirkuit (*short-circuiting*). Intervensi ini mereduksi kompleksitas perbandingan menjadi skenario asimtotik rata-rata konstan O(1) pada sebagian besar blok spasial yang gagal [2].

D. *Demonstrasi Manual Evaluasi Sliding Window dan Short-Circuit*

Untuk membuktikan efisiensi operasional dari metode Sliding Window yang dipadukan dengan optimasi Short-Circuit Evaluation, sub-bab ini menjabarkan simulasi matematis secara diskrit. Misalkan didefinisikan sebuah Himpunan Target (T) berdimensi 2×2 yang melambangkan matriks biner tangkapan layar susunan bedrock dari sisi klien, dan sebuah Semesta Observasi (W) berdimensi 3×3 yang merepresentasikan area probabilitas chunk hasil bangkitan Linear Congruential Generator (LCG).

Matriks target T dan matriks semesta W didefinisikan sebagai berikut, dengan pola T secara asumsional bersembunyi pada sudut kanan bawah ruang semesta W:

$$T = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$$W = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

Algoritma Sliding Window akan menggeser jendela observasi berukuran 2×2 melintasi W. Mengingat dimensi W adalah 3×3 dan dimensi jendela adalah 2×2,

jumlah iterasi pergeseran posisi kordinat relatif (i, j) yang secara matematis terbentuk adalah $(3 - 2 + 1) \times (3 - 2 + 1) = 4$ iterasi. Evaluasi ekuivalensi dieksekusi menggunakan operator logika bi-implikasi atau XNOR $(a \leftrightarrow b)$, yang akan mengembalikan nilai 1 (True) jika identik secara absolut, dan 0 (False) jika berbeda.

1) Iterasi 1: Posisi Jendela pada Indeks $(0,0)$

Sub-matriks observasi awal yang dievaluasi adalah:

$$W_{0,0} = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$$

Pemeriksaan berjalan secara linear dimulai dari elemen baris pertama dan kolom pertama $(0,0)$.

Kalkulasi XNOR: $W_{0,0}(0,0) \leftrightarrow T(0,0) \Longrightarrow 0 \leftrightarrow 1 = 0$ (False).

Karena ditemukan satu sel yang tidak isomorfik, optimasi *Short-Circuit* terpicu. Algoritma langsung memberhentikan konjungsi perbandingan menggunakan perintah *break*. Program memangkas kompleksitas waktu dengan tidak mengevaluasi 3 sel memori yang tersisa dan langsung bergeser ke posisi berikutnya.

2) Iterasi 2: Posisi Jendela pada Indeks $(0,1)$

Sub-matriks observasi bertranslasi ke arah kanan:

$$W_{0,1} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Pemeriksaan elemen sel $(0,0)$:

Kalkulasi XNOR: $W_{0,1}(0,0) \leftrightarrow T(0,0) \Longrightarrow 0 \leftrightarrow 1 = 0$ (False).

Evaluasi *Short-Circuit* kembali membatalkan iterasi secara instan tepat pada kalkulasi pertama.

3) Iterasi 3: Posisi Jendela pada Indeks $(1,0)$

Sub-matriks observasi bertranslasi ke arah bawah:

$$W_{1,0} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

Pemeriksaan elemen sel $(0,0)$:

Kalkulasi XNOR: $W_{1,0}(0,0) \leftrightarrow T(0,0) \Longrightarrow 1 \leftrightarrow 1 = 1$ (True).

Mengingat elemen pertama selaras, siklus dilanjutkan ke pengecekan elemen sel $(0,1)$:

Kalkulasi XNOR: $W_{1,0}(0,1) \leftrightarrow T(0,1) \Longrightarrow 1 \leftrightarrow 0 = 0$ (False).

Ketidakcocokan ditemukan pada sel kedua matriks. Putus sirkuit (*Short-Circuit*) membatalkan iterasi ini seketika, mengeliminasi keharusan mengevaluasi elemen baris bawah.

4) Iterasi 4: Posisi Jendela pada Indeks $(1,1)$

Sub-matriks observasi bergeser menempati posisi sudut kanan bawah matriks semesta:

$$W_{1,1} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

Pemeriksaan dilakukan secara sekuensial terhadap seluruh hierarki elemen:

Sel $(0,0)$: $1 \leftrightarrow 1 = 1$ (True)

Sel $(0,1)$: $0 \leftrightarrow 0 = 1$ (True)

Sel $(1,0)$: $1 \leftrightarrow 1 = 1$ (True)

Sel $(1,1)$: $1 \leftrightarrow 1 = 1$ (True)

Keseluruhan konjungsi Boolean untuk elemen-elemen matriks bernilai 1 secara mutlak. Tidak ada pemutusan fungsi *Short-Circuit* yang terpicu, secara definitif membuktikan validitas isomorfisme sempurna antara matriks target yang diamati dan probabilitas mesin.

Secara kesimpulan, saat isomorfisme logis yang sempurna ini tercapai, indeks pergeseran sub-matriks tersebut, dalam demonstrasi ini berada pada titik mula $(i = 1, j = 1)$, langsung diekstraksi. Indeks spasial relatif ini dioperasikan ke dalam fungsi translasi matematis untuk dikonversi secara presisi menjadi relasi vektor koordinat absolut (X, Z). Manipulasi sistematis matriks Boolean inilah yang meruntuhkan kerahasiaan luasnya entropi ruang, membongkar lokasi spesifik dengan komputasi deterministik tanpa harus menambang atau menghubungi pangkalan data server.

IV. PENGUJIAN DAN EVALUASI

Bab ini mendiskusikan implementasi infrastruktur pengujian berbasis konkurensi (*Concurrency*) untuk mendemonstrasikan kelayakan penyelesaian limitasi performa perangkat keras, diuraikan melalui tinjauan komputasional dan perhitungan *Big-O*.

A. Skenario Eksperimen Berbasis Multithreading dan Concurrency

Menjelajahi ruang dimensi dengan rentang hingga jutaan node komputasi merupakan beban operasional memori-berat (*memory-intensive*). Jika iterasi triliunan ekuivalensi LCG disalurkan melalui untaian tunggal yang identik dengan untaian antarmuka (GUI), program akan mengalami pembuntuan (*deadlock* atau *UI blocking*). Skenario pengujian menerapkan paradigma konkurensi berdasar Hukum Amdahl (*Amdahl's Law*). Modul kontrol disuntikkan ke dalam JendelaUtama.java:

```
// Translasi batas Euclidean -> domain Chunk
int minCX =
  Integer.parseInt(inputXMin.getText())/16;
int maxCX =
  Integer.parseInt(inputXMax.getText())/16;
// Enkapsulasi fungsi asinkron (Runnable)
Runnable tugasCari = new Runnable() {
  @Override public void run() {
    PencariBedrock.cariFormasi(daftarTarget,
      finalMinCX, finalMaxCX,
      finalMinCZ, finalMaxCZ, finalYDasar,
      finalIdDunia, tombolCari, areaHasil);
  }
};
// Spawning Thread independen (dekopling UI)
new Thread(tugasCari, "Thread Pencarian").start();
```

Potongan Kode 6. Pembentukan batas asimtotik parameter matriks dan pelepasan Thread dalam JendelaUtama.

Gambar 4. Validasi output program (kiri) terhadap koordinat aktual di Minecraft (kanan).

Interval vektor yang diinput pengguna dipetakan ke dalam domain divisor /16 menggunakan transformasi linear pembagian, memperkecil rentang perulangan secara signifikan. Kelas anonim tipe Runnable (tugasCari) dikonstruksi mandiri, lalu dilepaskan (*spawned*) pada prosesor eksekusi berbeda menggunakan *Thread*. Keberhasilan pembacaan koordinat aktual ditunjukkan pada Gambar 4. Pengalihan beban ini memastikan proses pembatalan algoritma mempertahankan latensi respons *real-time*, melegitimasi stabilitas pengujian spasial berskala masif.

B. Analisis Kompleksitas Waktu Aktual (Big-O)

Kapasitas algoritma ini bergantung pada waktu hitung. Mengasumsikan area pencarian bujur sangkar selebar batas observasi W, dimensi himpunan dalam ranah divisor *chunk* direpresentasikan sebagai rasio rentang. Operasi komputasional dievaluasi sebagai berikut [2]:

Tahapan Operasional	Notasi Big-O	Deskripsi Variabel
Generasi Matriks Tensor	$O(W^2)$	Iterasi baris–kolom area pencarian 2D.
Inisialisasi LCG & Array	$O(W^2 \cdot h)$	256 pilar vertikal per chunk, dikali tebal layer h.
Evaluasi Ekuivalensi	$O(1) – O(k)$	Cek per pilar terhadap panjang array Target k.
Garbage Collection	$O(1)$	Eviksi struktur batas dalam window memori aktif.

Tabel 3. Rincian distribusi kompleksitas per unit kalkulasi matematis komputasional dalam arsitektur Sliding Window.

Berdasarkan kompilasi operasional pada Tabel 3, fungsi pertumbuhan waktu komputasi absolut secara teoretis membentuk fungsi asimtotik berorde $O(W^2 \cdot h)$. Mengingat konstanta numerik dan interaksi ekuivalensi != pada unit aritmatika prosesor (ALU) memakan siklus serendah beberapa nano-detik, faktor pengali konstanta meluruh, menyisakan pertumbuhan deterministik orde $O(W^2)$. Dengan optimasi *short-circuit* (gagal-cepat) yang telah didemonstrasikan, komputasi evaluasi rerata menyusut drastis, membuktikan eksploitasi matematis ini dapat merobohkan kerahasiaan area seluas satu juta blok dalam hitungan detik.

C. Limitasi Sistem Operasional dan Probabilitas Positif Palsu

Meskipun penyelesaian teoretis algoritma deterministik dipastikan berjalan utuh dalam kondisi ideal (vakum analitik), komputasi dunia empiris memberlakukan parameter eksternal yang rentan terhadap anomali *False Positive*:

- Gangguan Kontaminasi Himpunan Bioma: Representasi rekayasa balik mengasumsikan pembangkitan pada level *vanilla noise terrain*. Namun, ekosistem Minecraft menginisiasi fungsi modifikasi (*structure generation features*) seperti danau bawah tanah, lava, goa, hingga struktur *server* kustom yang berpotensi menggores tensor biner murni. Jika struktur tersebut menghapus *bedrock* yang seharusnya ada, divergensi pola akan memicu penolakan kandidat meskipun vektor koordinatnya valid.
- Deviasi Versi Pembangkitan LCG: Formulasi eksploitasi dan nilai konstanta sangat terkait pada komputasi java.util.Random di versi 1.12.2 [5]. Mengingat mesin generasi baru (pasca-1.15) mengubah parameter rotasi basis 64-bit, rotasi arsitektur pergeseran bit akan mengubah koefisien rekurensi dan merusak korespondensi matriks (*bijection failure*), menjadikan algoritma ini tak terdefinisi pada versi termutakhir tanpa kalibrasi konstanta aljabar.

V. KESIMPULAN

Representasi struktural dunia topologi ruang virtual ke dalam persamaan himpunan Matematika Diskrit memvalidasi pembuktian bahwa lingkungan yang diasumsikan secara awam sebagai entitas acak kriptografis sejati, pada kenyataannya hanyalah hasil pengulangan rekursif dari rotasi determinisme aljabar skalar berperiode rendah. Eksploitasi rekayasa balik yang mengonversi koordinat spasial absolut ke dalam relasi himpunan Tensor Biner 3 Dimensi, dikawinkan dengan komputasi kebalikan nilai inisiasi (*seed*) dari siklus *Linear Congruential Generator* (LCG), berhasil memutarbalikkan pemetaan probabilitas mesin secara presisi seratus persen secara lokal, menggugurkan ilusi keamanan berbasis entropi jarak di *server anarchy*.

Penggunaan algoritmik fusi matriks melalui *Sliding Window* terbukti menjadi mekanisme resolusi asimtotik yang krusial guna meredam hambatan kompleksitas waktu kuadratik dan ancaman luapan memori. Pembuktian optimasi teoretis, destruksi objek *garbage collection* berbasis geometri *Bounding Box*, operator perbandingan relasional yang meniadakan iterasi panjang melalui evaluasi *short-circuit*, serta delegasi sinkronitas asinkron melalui konkurensi *Thread*, mendesain fungsi pencarian ekuivalensi LCG yang asimtotik efisien. Studi ini memfinalisasi paradigma tegas: perlindungan siber yang berbasis pada keacakan algoritmik LCG belaka adalah

kecacatan intrinsik sistem yang secara deterministik akan runtuh dalam observasi kalkulasi diskrit yang masif.

VI. LAMPIRAN

Link Youtube : https://youtu.be/qhCnO69BHlY

Link GitHub :
https://github.com/angseleong/bedrockfinder

VII. UCAPAN TERIMAKASIH

Penulis mengucapkan terima kasih kepada:
1. Tuhan Yang Maha Esa,
2. orang tua penulis,
3. Bapak dan dosen pengampu mata kuliah Matematika Diskrit IF1220,
4. teman-teman penulis, dan
5. pihak-pihak lain

yang telah mendukung penulis selama pembelajaran dan proses pengerjaan makalah ini.

DAFTAR PUSTAKA

[1] R. Munir, Matematika Diskrit. Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung. [Daring]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/. (Diakses: 17 Juni 2026).

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, dan C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.

[3] D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 3rd ed. Reading, MA, USA: Addison-Wesley, 1997.

[4] P. L'Ecuyer, "Tables of linear congruential generators of different sizes and good lattice structure," *Mathematics of Computation*, vol. 68, no. 225, pp. 249–260, 1999.

[5] Oracle, "Class Random," *Java Platform, Standard Edition 8 API Specification*, 2014. [Daring]. Tersedia: https://docs.oracle.com/javase/8/docs/api/java/util/Random.html. (Diakses: 17 Juni 2026).

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026

Axeleon Justin Algianto
13525074