

NAMA. : Gerald Bimo  
KELAS : 2025B  
NIM : 066

## PENJELASAN KODE SIMULASI HASH TABLE DENGAN LINEAR PROBING

### Analogi: Parkiran Motor di Kampus

Bayangkan sebuah kampus memiliki 15 slot parkir motor bernomor 0–14.

Setiap mahasiswa yang datang membawa motor akan diarahkan ke slot tertentu berdasarkan rumus khusus dari NIM mereka. Rumus ini disebut hash function.

Contohnya:

- a) NIM 2501 → menghasilkan angka 4 → parkir di Slot 4
- b) NIM 2502 → menghasilkan angka 9 → parkir di Slot 9

Petugas tidak perlu mengecek slot satu per satu. Ia langsung tahu harus menuju slot mana.

Itulah keunggulan hash table: cepat dan langsung ke tujuan.

### Collision dan Linear Probing

Masalah muncul ketika dua mahasiswa mendapat slot yang sama.

Misalnya:

- a) NIM 2510 juga menghasilkan angka 4  
Padahal Slot 4 sudah ditempati mahasiswa sebelumnya.

Ini disebut collision

(tabrakan). Solusinya: Linear

Probing Jika Slot 4 penuh:

- a) Coba Slot 5
- b) Jika penuh → coba Slot 6
- c) Jika penuh → coba Slot 7

Dan seterusnya sampai menemukan yang kosong.

Jika sudah sampai Slot 14 dan masih penuh, maka sistem kembali ke Slot 0 (berputar).

### Contoh Proses Penyimpanan

NIM	Hash Awal	Slot Akhir	Probe	Keterangan
2501	Slot 4	Slot 4	0	Langsung parkir
2502	Slot 9	Slot 9	0	Langsung parkir
2510	Slot 4	Slot 5	1	Collision dengan 2501
2520	Slot 5	Slot 6	1	Collision dengan 2510
2530	Slot 4	Slot 7	3	Lewati 4,5,6 baru kosong

Perhatikan NIM 2530.

Ia harus melewati tiga slot penuh sebelum mendapatkan tempat.

Fenomena ini disebut clustering

Artinya slot yang berdekatan menjadi penuh sehingga proses semakin lambat.

### Load Factor

Load factor adalah ukuran seberapa penuh tabel tersebut.

Rumusnya:

Jumlah slot terisi ÷ Total slot Misalnya:

- a) 10 slot terisi dari 15
- b) Load factor =  $10 \div 15 = 0.66$  (66%)

### Dampak Load Factor

Load Factor	Kondisi	Dampak
0% – 50%	Sangat longgar	Hampir tidak ada collision
50% – 70%	Normal	Collision mulai muncul
70% – 85%	Padat	Probing makin sering
>85%	Sangat padat	Clustering parah, lambat

Karena itu, biasanya hash table akan **di-resize otomatis** ketika load factor melewati 0.7 agar tetap efisien.

### Penjelasan Struktur Program

#### 1. plan\_inserts()

Fungsi ini mensimulasikan seluruh proses parkir:

- a) Menghitung hash
- b) Mengecek apakah slot kosong
- c) Jika collision → melakukan probing
- d) Menyimpan setiap langkah dalam bentuk “frame”

Semua proses direkam agar bisa ditampilkan kembali sebagai animasi.

#### 2. main()

Fungsi utama yang menampilkan visualisasi:

- a) Kotak-kotak slot parkir
- b) Warna berubah saat sedang diproses
- c) Grafik load factor terus diperbarui

Semakin banyak slot terisi, grafik akan naik.

### Kontrol Interaktif

Program biasanya memiliki kontrol seperti:

- a) SPASI → pause/play
- b) ← → → maju mundur satu langkah
- c) R → restart
- d) Q / Esc → keluar

Fitur ini membantu memahami setiap fase collision dan probing secara detail.

### **Alur Satu Proses Penyimpanan**

Setiap data baru masuk melewati 5 fase:

1. Start → Hitung hash dan sorot slot tujuan
2. Check → Apakah slot kosong?
3. Collision → Jika penuh, tambah probe
4. Probe → Geser ke slot berikutnya
5. Place → Data berhasil ditempatkan
6. Update → Load factor diperbarui

Fase collision dan probe bisa terjadi berkali-kali tergantung kepadatan tabel.

### **Kesimpulan**

Hash table bekerja seperti sistem parkir otomatis yang cerdas:

- a) Hash function → menentukan slot secara cepat
- b) Linear probing → menyelesaikan tabrakan dengan geser maju
- c) Load factor → indikator kesehatan sistem

Jika terlalu penuh, clustering terjadi dan performa menurun.

Dengan simulasi visual, konsep yang biasanya abstrak menjadi mudah dipahami karena bisa dilihat langsung setiap prosesnya.