Tugas Struktur Data

Nama : Farhan Bagas Firmansyah
Nim : 25091397039
Mata Kuliah : Struktur Data

Jelaskan Program ini

```
import argparse
import random
import numpy as np

import matplotlib
matplotlib.use("TkAgg")  # CMD Windows GUI backend

import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

def plan_inserts(keys, size=20):
    table = [None] * size
    frame_data = []
    load_factors = []

    for step, key in enumerate(keys, 1):
        start = hash(key) % size
        idx = start
        probes = 0

        def snap(phase, placed=False):
            frame_data.append({
                "phase": phase,
                "step": step,
                "key": key,
                "start": start,
                "idx": idx,
                "probes": probes,
                "table": list(table),
                "placed": placed
            })

        snap("start", placed=False)

        while table[idx] is not None:
            snap("collision", placed=False)
            probes += 1
            idx = (idx + 1) % size
```

```
            snap("probe", placed=False)

        table[idx] = key
        snap("place", placed=True)

        lf = sum(v is not None for v in table) / size
        load_factors.append(lf)

        snap("pause", placed=True)

    return frame_data, load_factors

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("--size", type=int, default=20)
    parser.add_argument("--nkeys", type=int, default=18)
    parser.add_argument("--seed", type=int, default=42)
    parser.add_argument("--save", choices=["none", "gif"], default="none")
    args = parser.parse_args()

    random.seed(args.seed)
    keys = [f"k{i}" for i in range(args.nkeys)]
    random.shuffle(keys)

    frames, load_factors = plan_inserts(keys, size=args.size)

    # ==== FIGURE LAYOUT ====
    fig = plt.figure(figsize=(12, 6), constrained_layout=True)
    gs = fig.add_gridspec(2, 1, height_ratios=[3, 1])

    ax_table = fig.add_subplot(gs[0])
    ax_lf = fig.add_subplot(gs[1])

    fig.suptitle("Hash Table Premium+ (Linear Probing) — Keyboard Control", fontsize=14)

    # ---- TABLE GRID ----
    n = args.size
    ax_table.set_xlim(-0.5, n - 0.5)
    ax_table.set_ylim(-0.5, 0.5)
    ax_table.set_yticks([])
    ax_table.set_xticks(range(n))
    ax_table.set_xlabel("Index Bucket")

    rects, texts = [], []
    for i in range(n):
        r = plt.Rectangle((i - 0.5, -0.35), 1.0, 0.7, fill=True, alpha=0.25)
        ax_table.add_patch(r)
        rects.append(r)
        t = ax_table.text(i, 0, "", ha="center", va="center", fontsize=9)
        texts.append(t)
```

```
info = ax_table.text(0.01, 1.08, "", transform=ax_table.transAxes, fontsize=11)
legend = ax_table.text(
    0.01, 1.01,
    "SPACE: pause/resume | ←/→: step (pause) | R: restart | Q/Esc: quit",
    transform=ax_table.transAxes, fontsize=9
)

# ---- LOAD FACTOR CHART ----
ax_lf.set_xlim(1, max(2, args.nkeys))
ax_lf.set_ylim(0, 1.05)
ax_lf.set_xlabel("Insert ke-")
ax_lf.set_ylabel("Load Factor")
ax_lf.grid(True, alpha=0.25)

lf_line, = ax_lf.plot([], [])
lf_dot, = ax_lf.plot([], [], marker="o", linestyle="")

bar_bg = plt.Rectangle((0.02, 0.15), 0.96, 0.2, transform=ax_lf.transAxes, alpha=0.15)
bar_fg = plt.Rectangle((0.02, 0.15), 0.00, 0.2, transform=ax_lf.transAxes, alpha=0.35)
ax_lf.add_patch(bar_bg)
ax_lf.add_patch(bar_fg)
lf_text = ax_lf.text(0.02, 0.45, "", transform=ax_lf.transAxes, fontsize=10)

# ==== PLAYER STATE ====
state = {
    "i": 0,          # current frame index
    "paused": False  # paused flag
}

def render(frame):
    """Render a single frame dict."""
    table = frame["table"]
    start = frame["start"]
    idx = frame["idx"]
    phase = frame["phase"]

    # reset buckets
    for j in range(n):
        rects[j].set_alpha(0.25)
        rects[j].set_linewidth(1.0)
        texts[j].set_text("" if table[j] is None else str(table[j]))

    # highlight target hash bucket
    rects[start].set_alpha(0.55)
    rects[start].set_linewidth(2.5)

    # highlight current idx
    rects[idx].set_alpha(0.75)
    rects[idx].set_linewidth(3.0)
```

```
        # collision emphasis
        if phase == "collision":
            rects[idx].set_alpha(0.9)

        filled = sum(v is not None for v in table)
        lf = filled / n

        info.set_text(
            f"Frame {state['i']+1}/{len(frames)} | Step {frame['step']}/{args.nkeys} |
key='{frame['key']}' | "
            f"hash%size={start} | idx={idx} | probes={frame['probes']} | load_factor={lf:.2f} |
phase={phase} | "
            f"{'PAUSED' if state['paused'] else 'PLAY'}"
        )

        # completed inserts so far
        completed = frame["step"] - 1 + (1 if frame["placed"] else 0)
        if completed <= 0:
            x, y = [], []
        else:
            x = list(range(1, completed + 1))
            y = load_factors[:completed]

        lf_line.set_data(x, y)
        if x:
            lf_dot.set_data([x[-1]], [y[-1]])
            bar_fg.set_width(0.96 * y[-1])
            lf_text.set_text(f"Load Factor: {y[-1]:.2f} ({int(y[-1]*100)}%)")
        else:
            lf_dot.set_data([], [])
            bar_fg.set_width(0.00)
            lf_text.set_text("Load Factor: 0.00 (0%)")

        return rects + texts + [info, legend, lf_line, lf_dot, bar_fg, lf_text]

    def update(_):
        """Animation tick: advance if not paused; otherwise keep frame."""
        if not state["paused"]:
            state["i"] = min(state["i"] + 1, len(frames) - 1)
        return render(frames[state["i"]])

    # Init render
    def init():
        return render(frames[state["i"]])

    ani = FuncAnimation(
        fig, update, init_func=init,
        frames=np.arange(len(frames)),  # timer ticks, actual frame controlled by state["i"]
        interval=450, blit=False, repeat=False
```

```
    )

    def on_key(event):
        k = event.key
        if k == " ":
            state["paused"] = not state["paused"]
            fig.canvas.draw_idle()

        elif k == "right":
            if state["paused"]:
                state["i"] = min(state["i"] + 1, len(frames) - 1)
                render(frames[state["i"]])
                fig.canvas.draw_idle()

        elif k == "left":
            if state["paused"]:
                state["i"] = max(state["i"] - 1, 0)
                render(frames[state["i"]])
                fig.canvas.draw_idle()

        elif k in ("r", "R"):
            state["i"] = 0
            state["paused"] = True  # restart dalam kondisi pause biar enak step
            render(frames[state["i"]])
            fig.canvas.draw_idle()

        elif k in ("escape", "q", "Q"):
            plt.close(fig)

    fig.canvas.mpl_connect("key_press_event", on_key)

    if args.save == "gif":
        try:
            from matplotlib.animation import PillowWriter
            ani.save("hash_table_premium_keyboard.gif", writer=PillowWriter(fps=2))
            print("Tersimpan: hash_table_premium_keyboard.gif")
        except Exception as e:
            print("Gagal simpan GIF. Pastikan 'pillow' terpasang. Error:", e)

    plt.show()

if __name__ == "__main__":
    main()
```

Penjelasannya

A. Program ini

adalah sebuah simulasi serta penggambaran dari langkah-langkah penyisipan data dalam struktur data Hash Table dengan memanfaatkan metode Linear Probing untuk mengatasi terjadinya collision.

Penggambaran ini dibuat dengan menggunakan pustaka matplotlib dan dilengkapi dengan kontrol keyboard agar pengguna dapat menjalankan, menghentikan, serta menelusuri langkah-langkah proses secara manual.

Tujuan dari program ini adalah untuk memfasilitasi pemahaman tentang bagaimana collision ini terjadi dan bagaimana prosedur linear probing berjalan secara bertahap.

B. Konsep Dasar yang Digunakan

2.1 Hash Table

Hash Table adalah struktur data yang menyimpan data berdasarkan hasil fungsi hash.
Rumus dasar untuk menentukan index:
index = hash(key) % size
Keterangan:

- key = data yang dimasukkan
- size = ukuran tabel
- index = posisi penyimpanan dalam tabel

2.2 Collision

Collision terjadi ketika dua key menghasilkan index yang sama.
Contoh:

- hash("A") % 5 = 2
- hash("B") % 5 = 2

Keduanya ingin masuk ke index 2 → terjadi collision.

2.3 Linear Probing

Linear Probing adalah metode penanganan collision dengan cara:

- Jika slot penuh → geser ke kanan satu per satu
- Gunakan rumus:

(index + 1) % size

Proses ini dilakukan sampai ditemukan slot kosong.

C. Struktur Program

Program ini terdiri dari beberapa komponen utama:

3.1 Fungsi plan_inserts()

Fungsi ini memiliki tugas sebagai berikut:

Membuat tabel tanpa isi
Menambahkan kunci satu demi satu
Mencatat setiap tahapan dari proses
Menghitung faktor beban

Seluruh langkah dicatat dalam bentuk bingkai untuk ditampilkan sebagai animasi.

3.2 Bagian Visualisasi

Antarmuka program terbagi menjadi dua bagian:

A. Bagian Atas – Tabel Hash

Menampilkan:

Kotak-kotak bucket
Isi dari setiap bucket
Penyorotan indeks hasil hash
Penyorotan indeks yang sedang diperiksa
Informasi rinci mengenai proses (langkah, kunci, pencarian, faktor beban)

B. Bagian Bawah – Grafik Faktor Beban

Menampilkan:
Grafik peningkatan faktor beban
Titik penyisipan terakhir
Balk persentase kepenuhan table

D. Load Factor

Load factor adalah ukuran kepadatan tabel.
Rumus:

load_factor = jumlah_elemen / ukuran_tabel

Contoh:
Jika tabel berukuran 10 dan terisi 7 elemen:
7 / 10 = 0.7 (70%)
Semakin tinggi load factor:

- Collision semakin sering
- Proses probing semakin Panjang

E. Sistem Animasi

Program menggunakan FuncAnimation untuk memutar frame satu per satu.
Terdapat variabel state:

- i → posisi frame saat ini
- paused → status animasi (berjalan atau berhenti)

Jika tidak pause:

- Frame bertambah otomatis

Jika pause:

- Frame tidak berubah

F. Manfaat Program

Program ini mendukung:
• Memahami tabrakan dengan cara visual
• Mengamati gerakan probing secara langsung
• Melihat peningkatan faktor beban
• Mempelajari struktur data melalui interaksi
• Mengerti relasi antara kepadatan tabel dan kinerja

G. Rangkuman

Program Hash Table Premium+ ini adalah simulasi yang bersifat pendidikan yang menunjukkan:
1. Cara kerja Hash Table
2. Bagaimana tabrakan terjadi
3. Cara Linear Probing mengatasi tabrakan
4. Bagaimana faktor beban mempengaruhi kinerja
5. Penggunaan animasi dan event keyboard dalam pemrograman visual
Dengan adanya visualisasi ini, konsep yang seringkali sukar dipahami menjadi lebih mudah untuk dimengerti secara langsung.