# 10. CVIČENÍ Z DATOVÝCH STRUKTUR 1, ZS24/25

Řešení kolizí v hešovacích tabulkách: lineární přidávání a kukačka.

1. *Lineární přidávání s většími skoky.* Uvažujme hešování s otevřenou adresací řízené obecnou lineární posloupností $h(x, i) = (f(x) + c \cdot i) \mod m$, kde $c$ je konstanta nesoudělná s velikostí tabulky $m$. Srovnejte jeho chování s obyčejným lineárním přidáváním.

2. *Opravdové mazání pro lineární přidávání.* Na přednášce bylo řešení kolizí pomocí lineárního přidávání s tím, že pokud prvek mažeme, pouze ho označíme za smazaný („postavíme tam pomníček“). Zkuste domyslet detaily a alternativní řešení:

a) Pokud provedeme hodně operací mazání, bude hešovací tabulka obsahovat více označených (tj. smazaných) prvků než nesmazaných. Co provést v takovém případě? Jak zajistit konstantní amortizovanou časovou složitost (ve střední hodnotě) pro libovolnou sekvenci operací Insert a Delete? (Můžete předpokládat větu z přednášky o konstantní střední hodnotě počtu přihrádek do nejbližší volné.)

b) Alternativní způsob: mazaný prvek skutečně smažeme a poté vhodně přesuneme nějaké prvky. Vymyslete, jak to přesně udělat, abychom pak mohli vyhledat všechny nesmazané prvky (tedy abychom tabulku „nerozbili“).

3. *Špátná kukačka.* Proč je následující implementace insertu pro kukačkové hashování problematická? (Implementaci přehashování a podmínky pro jeho spuštění teď zameteme pod koberec.)

```
for i=1 to n
    if T[h1(x)] je prázdná
        T[h1(x)] = x
        return
    swap(T[h1(x)], x)
    if T[h2(x)] je prázdná
        T[h2(x)] = x
        return
    swap(T[h2(x)], x)
```

4. *Kukačka s přehešováním na místě.* Jednoduchá implementace přehešování je nasypat prvky do pomocného pole a potom je po jednom přidávat do tabulky použitím Insertu. Vymyslete implementaci přehešování, která nepotřebuje pomocné pole. (Pozor na to, že během přehešování se opět může spustit přehešování...)