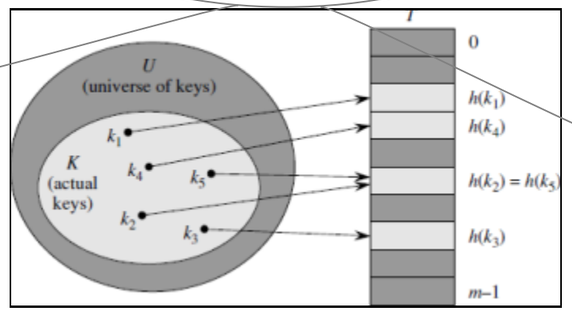
**Hašování -** Operace: SEARCH, INSERT, DELETE
**Univerzum klíčů U** a **K**⊆U je množina použitých klíčů
**Hašovací funkce h: U→{0,1,..,m−1}** mapuje univerzum U do (menší) tabulky T [0,..,m−1], |U| > m
**Kolize** je situace: $h(k_i) = h(k_j)$, pro $k_i \neq k_j$; $k_i, k_j \in K$

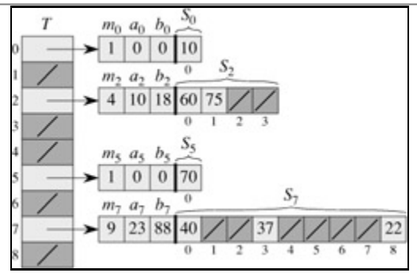



**Univerzální hashování**
Abychom si vylepšili naše šance že h bude fungovat dobře, zaveďme si obecnější třídu (různorodých majících pro |K|≤m mají malý počet kolizí) funkcí H; než začneme hashovat, vybereme z nich náhodnou funkci h∈H (ta se stane atributem konkrétní instance hashovací tabulky).

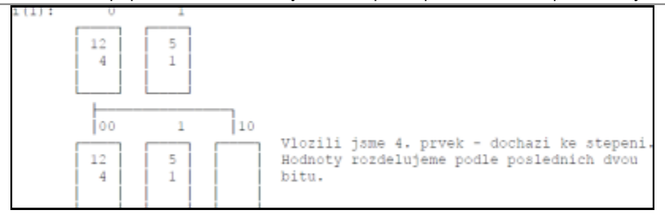
snaha o efektivní prostorovou reprezentaci a malou časovou náročnost operací

**Příklad:**
$h_{a,b}(k) = ((ak + b) \bmod p) \bmod m$
kde p je prvočíslo > max(k∈U)

**Perfektní hashování -** nepřipouští kolize. Nevýhoda této metody je, že nelze dost dobře implementovat operaci INSERT, proto se dá prakticky použít pouze tam, kde předpokládáme hodně operací MEMBER a jen velmi málo operací INSERT. Kolize se potom dají řešit třeba malou pomocnou tabulkou, kam se ukládají kolidující data.
Příklad perfektního hašování pomocí dvouúrovňového univerzálního hašování:



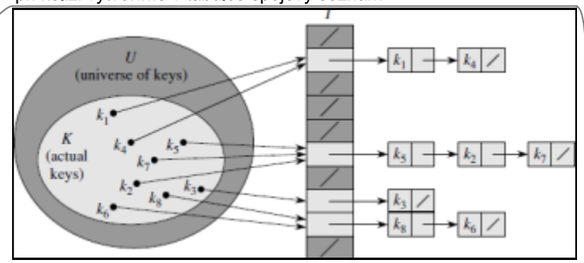

**Lineární hashování (Litwin)-** umožňuje zvětšovat hashovací tabulku každých L vložení
- hashovací funkce bere poslední bity a podle toho zařazuje do stránek/přihrádek
- každých L vložení rozštěpí přihrádku na dvě zvýší u nich počet posledních bitů podle kterých rozděluje




**Hašování s uspořádanými řetězci**
seznamy jsou uspořádány podle velikosti
-rychlejší op.SEARCH

nevýhodou obou je nerovnoměrné využití paměti
někde můžou být dlouhé seznamy jinde nic

**Hašování separovanými řetězci**
při kolizi vytvoříme v tabulce spojový seznam




nevýhoda: přemísťování při INSERTu zpomaluje

**Hašování s přemisťováním**
každé políčko má dva ukazatele (**prev, next**), jejichž pomocí tvoří kolizní položky řetězec
pokud kolizní položka zabírá místo položce, co na místo dle haše patří, je přemístěna

| řádek | key | next | previous |
|---|---|---|---|
| P(0) | | | |
| P(1) | 1 | 9 | |
| P(2) | | | |
| P(3) | 73 | 6 | |
| P(4) | 11 | 5 | 9 |
| P(5) | 161 | | 4 |
| P(6) | 53 | | 3 |
| P(7) | 7 | | |
| P(8) | 28 | | |
| P(9) | 141 | 4 | |

zrychlí INSERT a DELETE ale zpomalí SEARCH

**Hashování se dvěma ukazateli**
-podobné, políčko má ukazatele následník a začátek řetězce ( **begin, next**)
-prvky se nepřemisťují, místo toho může být začátek přesměrován pomocí druhého ukazatele

**Hašování s lineárním přidáváním**
-tabulka má jen klíč, kolizní prvky se dávají na první volné místo
-použitelné do zaplnění 75% pak moc velké shluky