IAL – 6. přednáška

Vyhledávací tabulky III.

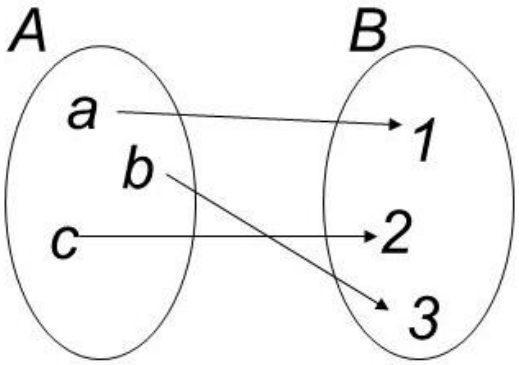
29. a 30. října 2024

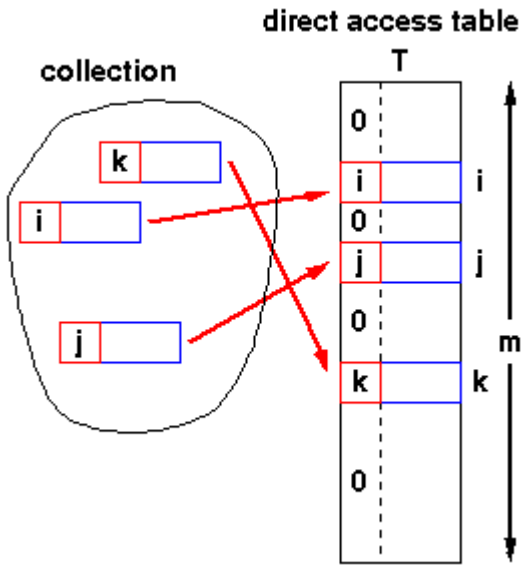
Obsah přednášky

- Tabulky s přímým přístupem (TPP)
- Mapovací funkce
- Tabulky s rozptýlenými položkami (TRP)
  - Princip index-sekvenčního vyhledávání
  - Explicitní a implicitní zřetězení synonym
- Zhodnocení metod vyhledávání
- Hashovací funkce

Tabulka s přímým přístupem (TPP)

- Implementace vyhledávací tabulky polem, ve které jsou klíče mapovány na indexy pole:
  - Ideální struktura z pohledu vyhledávání
  - Bohužel obvykle nerealizovatelná
- Vyžaduje **vzájemně jednoznačné zobrazení (bijekce)** mapující každý prvek množiny klíčů **K** do množiny indexů pole **H** (sousedních adres v paměti).





TPP: příklad

- Uvažujme množinu klíčů danou intervalem **<0..999>** a tabulku reprezentovanou polem **T[0..999]**.
- Pak můžeme použít zobrazení, které mapuje klíč **K** na pozici **T[K]**.

- Klíče typu `int` mohou takto být snadno mapovány na indexy, proto se používají v řadě případů.

- ***Pozn.:*** Všimněte si, že rozsah pole zde odpovídá rozsahu hodnot klíče (počtu všech možných hodnot klíče). Tento rozsah však v praxi bývá často neúnosně velký.

TPP: implementace

- Každému prvku pole pořídíme booleovskou složku `busy`, která určuje, zda v tabulce prvek s daným klíčem existuje nebo neexistuje (je nebo není).
- Při inicializaci se pro všechny prvky tabulky nastaví `busy ← false`.
- **Vyhledávání:** spočívá v přímém zjištění, zda na pozici klíče (indexu) dané tabulky je nebo není obsazeno.

- Časová složitost přístupu v TPP: $\Theta(1)$
- **Obtíž:** nalezení **vhodné mapovací funkce.**

TPP: příklad 2

- Předpokládejte, že chceme ukládat informace o studentech. Každému studentovi je přiřazeno jedinečné sedmimístné identifikační číslo, které může sloužit jako index do pole realizující vyhledávací tabulku:

0		
:	:	:
12345	andy	81.5
:	:	:
33333	betty	90
:	:	:
56789	david	56.8
:	:	:
:	:	:
9908080	bill	49
:	:	:
9999999		

- Pro uložení informací např. o 1 000 studentech bychom potřebovali pole o 10 000 000 prvcích (využití paměti 0,1 ‰)! Identifikační číslo tedy není vhodné použít přímo, ale přes **mapovací funkci**.

Mapovací funkce

- Nalezení vzájemně jednoznačného zobrazení (mapovací funkce) je velmi obtížné ⇒ je potřeba počítat s tím, že běžná mapovací funkce může různým klíčům přiřadit stejnou hodnotu (stejné místo v paměti).
  - **Kolize** – dva různé klíče jsou namapovány do stejného místa.
  - **Synonyma** – dva nebo více klíčů, které jsou namapovány do téhož místa.

⇒ **tabulka s rozptýlenými položkami**
(TRP, hashovací tabulka)

Mapovací funkce

- **Obtíž:** Nalezení vhodné mapovací funkce – proč?
  - Například pro 31 prvků, které se mají zobrazit do 41 prvkové množiny, existuje 41^{31} tj. cca 10^{50} mapovacích funkcí.
  - Přitom jen (41!/10!) tj. cca 10^{43} z nich jsou vzájemně jednoznačné.
  - Poměr *ideálních* funkcí, které žádné dva klíče nemapují na stejné místo, ku všem možným je zde asi 1:10 000 000.

- *Příklad:* Paradox společných narozenin
  - Je dobrá naděje, že mezi 23 osobami, které se sejdou ve společnosti, se najdou dvě osoby, které mají narozeniny ve stejný den.
  - Jinými slovy: Vybereme-li náhodně funkci, která mapuje 23 klíčů do tabulky o 365 prvcích, je pravděpodobnost, že se žádné dva klíče nenamapují do stejného místa, 49,3 %.

Mapovací funkce

- Nechť je dáno mapovací pole s rozsahem [0…N] nebo [1…N].
- Mapovací funkce transformuje klíč na index v daném rozsahu.
- Typicky lze rozdělit do dvou etap:
  - převod klíče na přirozené číslo (N>0),
  - převod přirozeného čísla na hodnotu spadající do intervalu (nejčastěji s použitím operace modulo).

Mapovací funkce – požadavky

- Determinismus
  - Pro daný klíč vrátí vždy stejnou hodnotu.
- Rovnoměrné (uniformní) rozložení
  - Na každé místo se mapuje přibližně stejně velké množství klíčů.
- Využití celých vstupních dat
- Vyhnutí se kolizím podobných klíčů
  - V praxi bývá řada klíčů velice podobných.
- Rychlý výpočet

- Neexistuje obecné pravidlo, jak nalézt nejvhodnější mapovací (rozptylovací) funkci.

Mapovací funkce: příklady

- PJW (Peter J. Weinberger)
- ELF Hash (UNIX)
- BKDR (Brian Kernighan, Dennis Ritchie)
- DEK (Donald E. Knuth)
- DJB (Daniel J. Bernstein)
- …

Ukázka mapovací funkce – BKDR

```
unsigned int BKDRHash(char* str, unsigned int length)
{
  unsigned int seed = 131;
  unsigned int hash = 0;
  unsigned int i = 0;
    for (i = 0; i < length; str++, i++)
    {
      hash = (hash * seed) + (*str);
    }
  return hash;
}
```

Ukázka mapovací funkce – DJB

```
unsigned long DJBHash(unsigned char *str)
{
  unsigned long hash = 5381;
  int c;
  while (c = *str++)
                                  // hash * 33 + c
    hash = ((hash << 5) + hash) + c;
  return hash;
}
```

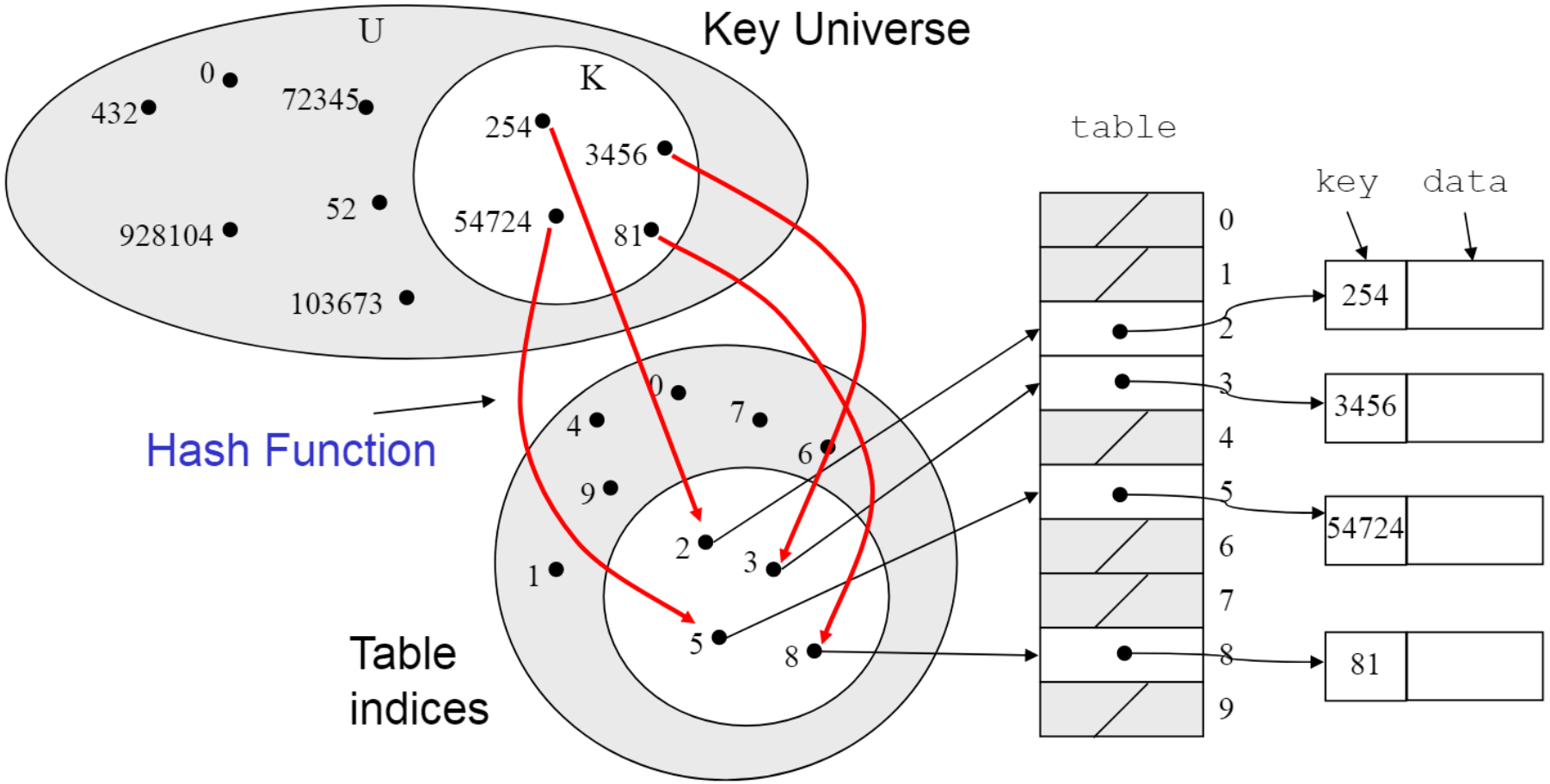
Převod hodnot do hranic pole

- Nechť **K** je celé číslo větší než nula.
- Funkce **$h_0(K) = K \bmod (Max+1)$**
  získá hodnoty z intervalu <0..Max>.
- Funkce **$h_1(K) = K \bmod Max + 1$**
  získá hodnoty z intervalu <1..Max>.

Tabulka s rozptýlenými položkami

- Tabulka s rozptýlenými položkami (TRP) sestává:
  - z mapovacího prostoru (pole) a
  - ze seznamů synonym.
- Seznam synonym (i prázdný) začíná na každém prvku mapovacího pole.
  - Explicitní zřetězení – adresa následníka je obsažena v jeho předchůdci (zřetězení záznamů).
  - Implicitní zřetězení – adresa následníka se získá pomocí funkce z adresy předchůdce (otevřená adresace).

Ukázka tabulky s rozptýlenými položkami



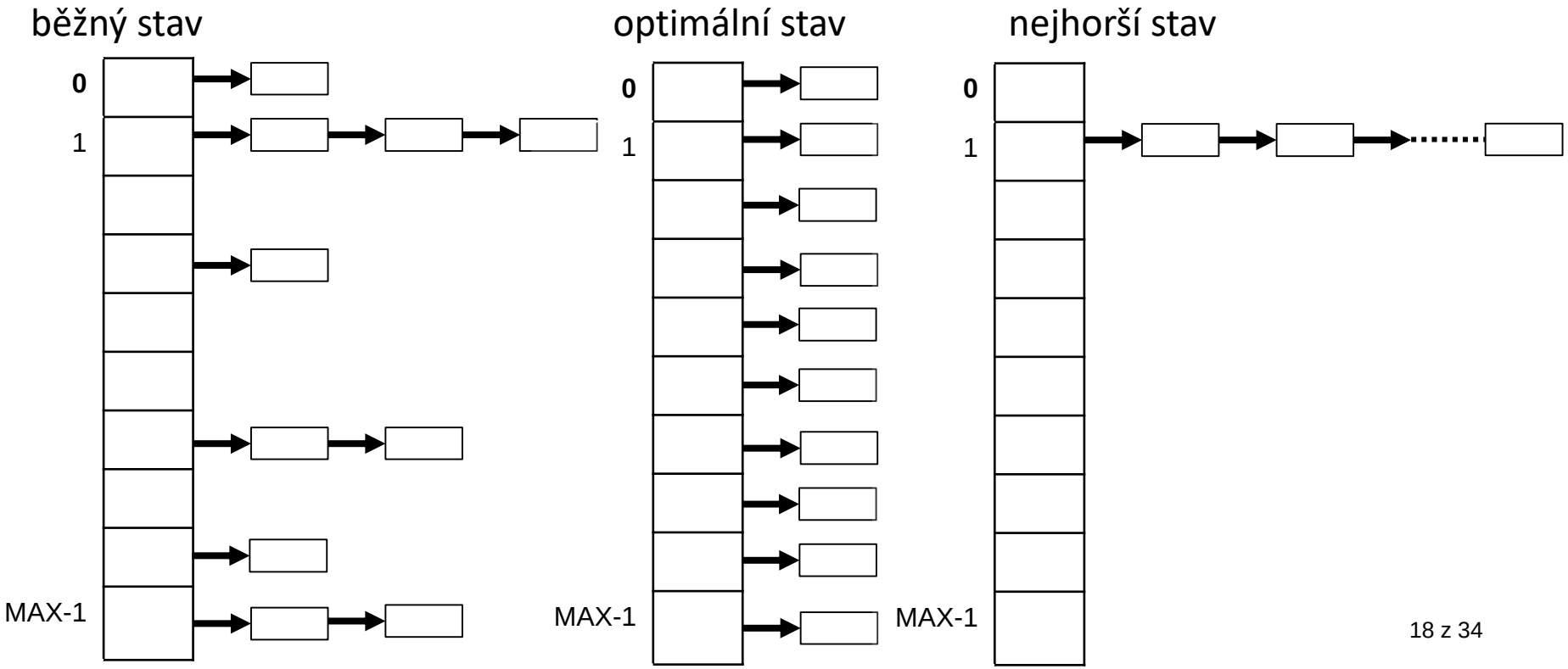
Vyhledávání v TRP

- Princip vyhledávání v TRP spočívá ve dvou krocích:
  1. Nalezení indexu prvku v poli k danému klíči pomocí mapovací funkce (na tomto indexu začíná seznam synonym, které se namapovaly do tohoto místa).
  2. Sekvenční průchod tímto seznamem synonym (vyhledáváme položku s daným klíčem).

$\Rightarrow$ Vyhledávání v TRP má index-sekvenční charakter.

TRP s explicitním zřetězením synonym

- Seznam synonym je obvykle realizován jako lineární seznam.
- Maximální doba vyhledávání je pak dána délkou nejdelšího seznamu synonym – ***O(n)***.

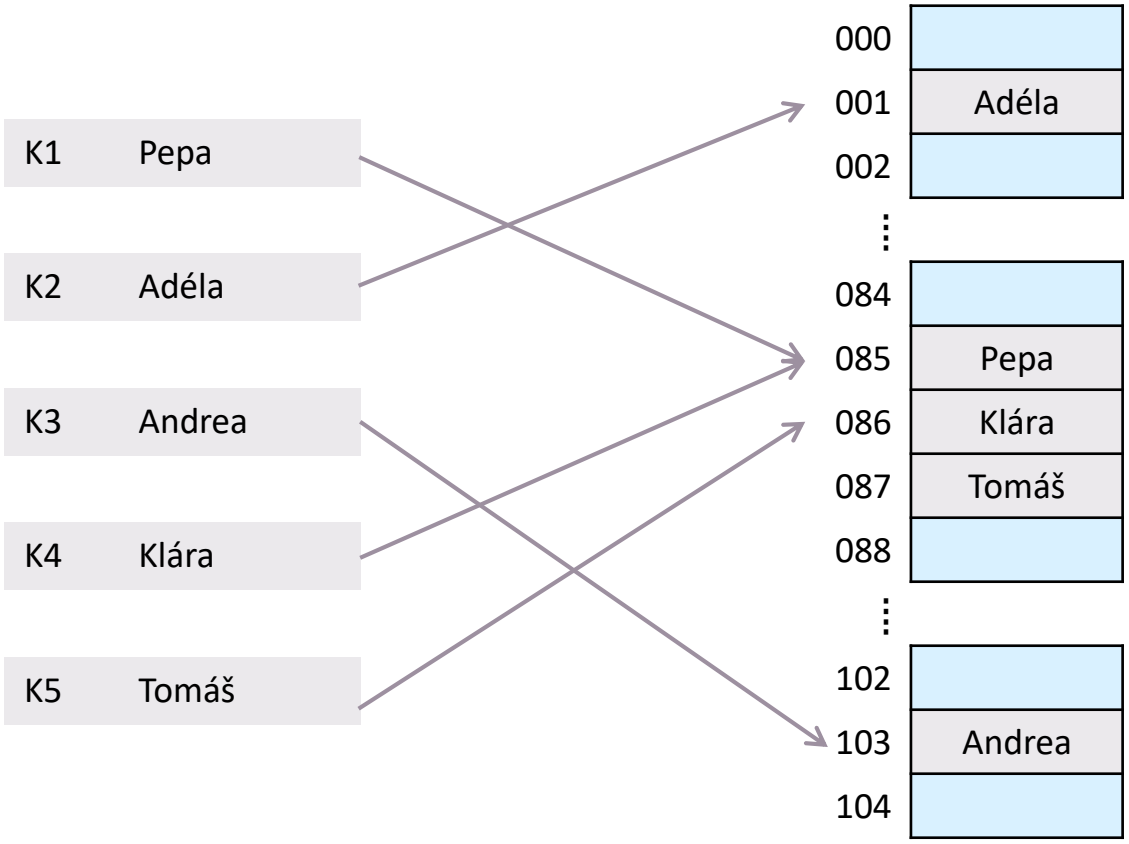


Alternativní zřetězení synonym

- Místo lineárních seznamů pro uložení synonym lze použít **vyvažované binární vyhledávací stromy**.
- Pak je časová složitost v nejhorším případě $O(\log_2 n)$.

TRP s implicitním zřetězením synonym

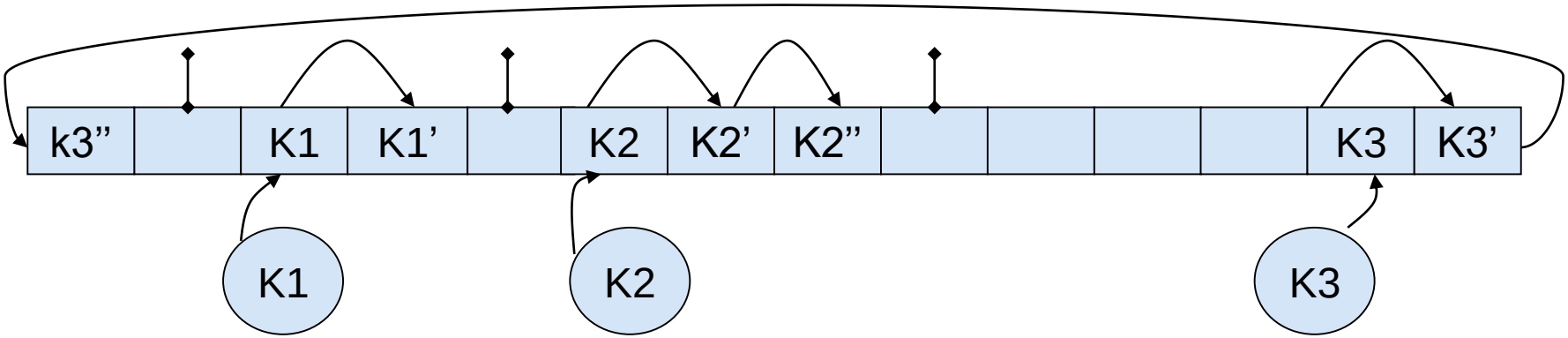
- TRP implementovaná polem, ve kterém jsou uloženy jak první prvky seznamů synonym, tak jejich další položky



TRP s implicitním zřetězením synonym

- TRP s otevřenou adresací
- Pro přístup k synonymům existují různé metody pro určení kroku:
  - Lineární: $h(k, i) = (h(k) + C*i) \% (Max+1)$
  - Kvadratická: $h(k, i) = (h(k) + C_1*i + C_2*i^2) \% (Max+1)$
  - S dvojí rozptylovací funkcí:
    $h(k, i) = (h_1(k) + h_2(k)*i) \% (Max+1)$
  - *Legenda:*
    $i = 0, 1, 2, \ldots$ – pokusy o vložení
    C, C_1, C_2 – konstanty
    $Max+1$ – velikost pole

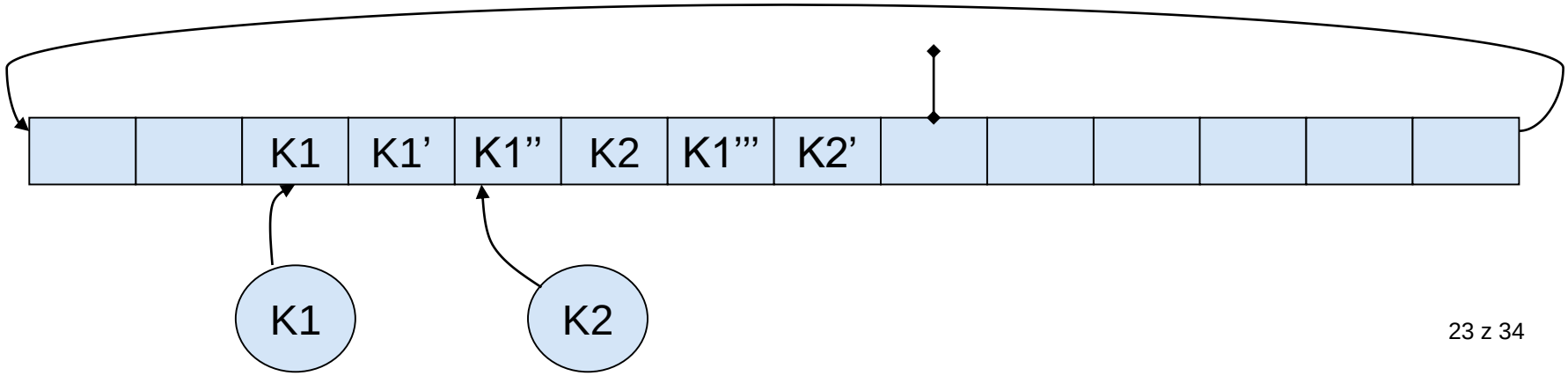
Implicitní zřetězení s pevným krokem



- Krok = 1: **a(i+1) = a(i) + 1**
- Konec seznamu synonym je dán prvním volným prvkem, který se najde se zadaným krokem.
- Nové synonymum se vloží na první volné místo (na konec seznamu).
- Tabulka (pole) musí obsahovat alespoň jeden volný prvek. Efektivní kapacita je o 1 menší než počet položek.
- Tabulka je implementovaná kruhovým polem.

Překrývání seznamů synonym

- Nechť jsou již do tabulky vloženy klíče K1, K1’ a K1’’.
- Následně se klíč K2 namapoval do položky, která je obsazena (je tam klíč K1’’). Klíč byl uložen na první volné místo. Další klíč K1’’’ se namapoval do položky K1. První volné místo pro klíč K1’’’ bylo nalezeno za klíčem K2. Klíč K2’ se namapoval do položky K1’’. První volné místo se nalezlo za klíčem K1’’’.
- V této tabulce se dva seznamy synonym překrývají. Prvek K1’’ je vstupním bodem seznamu synonym K2.



Zrušení položky v TRP

- **Zrušení** položky v TRP s explicitním zřetězením synonym je poměrně snadné (odpovídá zrušení prvku v lineárním seznamu případně v binárním vyhledávacím stromu).

- *K zamyšlení:*
  Jak lze zrušit položku v TRP s implicitním zřetězením synonym?

Velikost rozptylovacího pole (1/2)

- Krok s hodnotou 1 má tendenci vytvářet *shluky* (angl. *cluster*).
- Výhodnější je krok větší než 1. Takový krok by ale měl mít možnost *navštívit* všechny položky pole. Kdyby pole mělo sudý počet prvků, pak sudý krok by z výchozího lichého indexu mapovaného prvku mohl navštívit pouze liché indexy.
  - Např. pole [1..10], krok 2, začátek na indexu 5 projde indexy: 5, 7, 9, 1, 3.
  - Krok 4 a začátek na indexu 7 projde indexy: 7, 1, 5, 9, 3.

Velikost rozptylovacího pole (2/2)

- Kdyby měl krok hodnotu prvočísla, které je nesoudělné s jakoukoli velikostí pole, pak by mohl postupně projít všemi prvky pole.
- Výhodnější ale je, aby hodnotu prvočísla měla velikost mapovacího pole. Pak jakýkoli krok dovolí projít všemi prvky mapovacího pole.
- Je vhodné dimenzovat velikost mapovacího pole TRP tak, aby bylo rovno prvočíslu.

TRP s dvojí rozptylovací funkcí

- Metoda s **dvojí rozptylovací** (hashovací) **funkcí:**
  krok v rozptylovacím poli je určen za běhu
  druhou rozptylovací funkcí.
- Nechť má rozptylovací pole rozsah <0..Max>, (kde hodnota Max+1 je prvočíslo) a nechť KInt je klíč transformovaný na celou nezápornou hodnotu.
- ***První rozptylovací funkce*** vrací index z intervalu <0..Max>:

$$\mathbf{ind_0 = h_1(KInt) = KInt \bmod (Max+1)}$$

- ***Druhá rozptylovací funkce*** vytváří krok z intervalu <1..Max>:

$$\mathbf{krok = h_2(KInt) = KInt \bmod Max + 1}$$

Vyhledání – Search

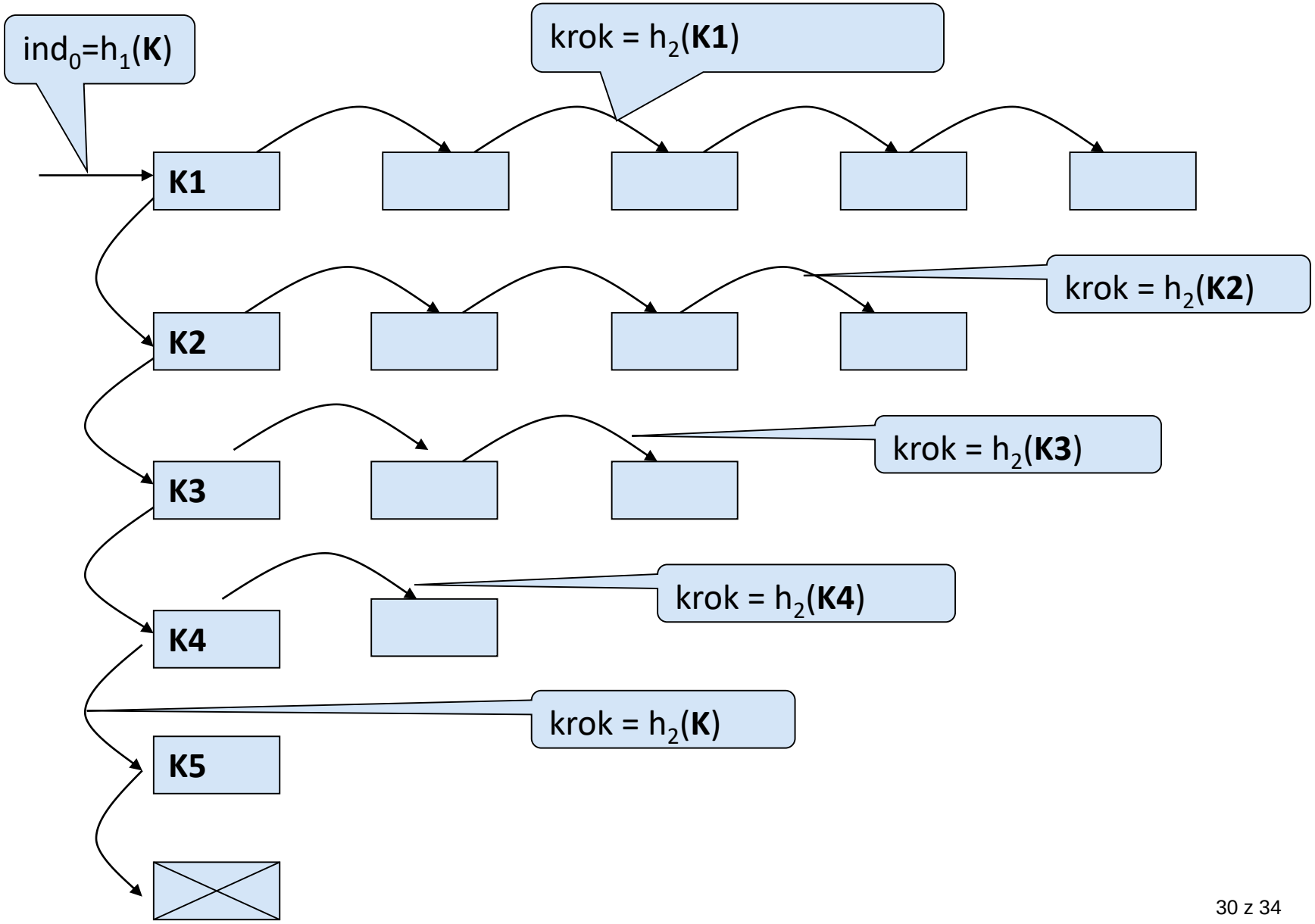
- **Vyhledávací cyklus:**
  - Začíná na indexu získaném první rozptylovací funkcí.
  - Končí úspěšně nalezením prvku s hledaným klíčem, neúspěšně při dosažení konce seznamu synonym (prázdným prvkem).
  - Index následujícího prvku je dán přičtením kroku k aktuálnímu prvku při respektování kruhovosti pole:

$$\mathbf{ind_{i+1} = (ind_i + krok) \bmod (Max+1)}$$

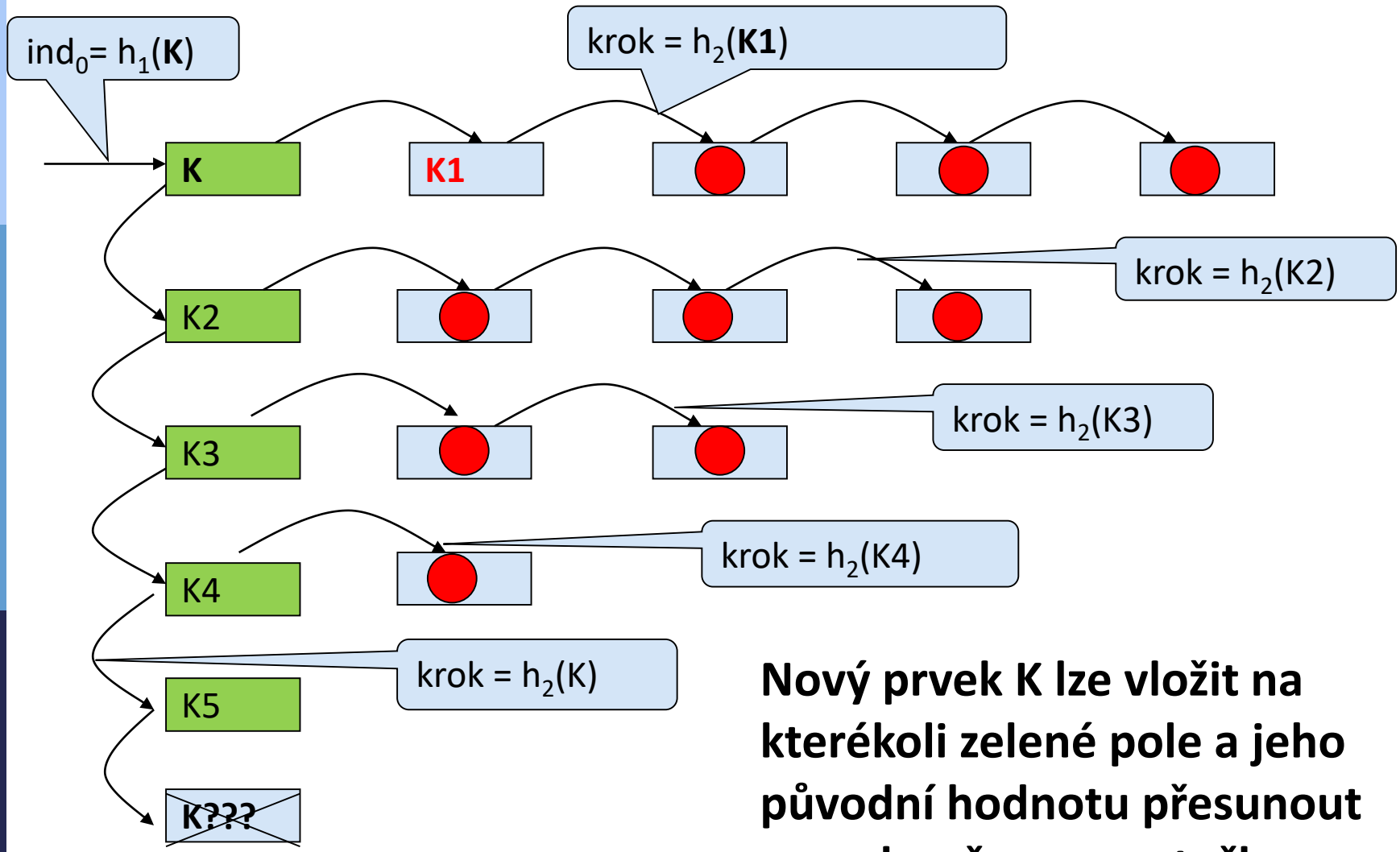
- Operace Insert vloží nový prvek na místo prvního prázdného prvku.

Brentova varianta

- Brentova varianta je varianta metody TRP se dvěma rozptylovacími funkcemi.
- Princip vyhledávání (Search) je shodný s TRP se dvěma rozptylovacími funkcemi.
- Brentova varianta je vhodná za podmínky, že počet případů úspěšného vyhledávání je častější než počet neúspěšného vyhledání s následným vkládáním.
- Brentova varianta provádí při vkládání rekonfiguraci prvků pole s cílem investovat do vkládání a získat lepší průměrnou dobu vyhledání.


ind0=h1(K)
krok = h2(K1)
K1
krok = h2(K2)
K2
krok = h2(K3)
K3
krok = h2(K4)
K4
krok = h2(K)
K5

Prvek K1 se přesune na první volné místo s krokem $h_2(K1)$ a na jeho místo se vloží prvek K.



Nový prvek K lze vložit na kterékoli zelené pole a jeho původní hodnotu přesunout na pole s červenou tečkou.

Brentova varianta

- Normální metoda by nový klíč vložila na první volné místo s krokem $h_2(K)$ – zde po 6 krocích.
- Brentova varianta hledá první volné místo mezi červeně zakroužkovanými poli s krokem $h_2(K1)$, resp. $h_2(K2)$ atd. Na toto místo vloží prvek K1, resp. K2 atd. a na uvolněné místo vloží prvek K.
- Protože posun čelního prvku je menší než zde 5, je celková průměrná hodnota délky vyhledávání menší, než kdyby byl prvek K vložen na 6 pozici shora.

Hodnocení TRP s implicitním zřetězením

- **Operaci Delete** lze řešit pomocí ***zaslepení*** – vložením klíče, který nebude nikdy vyhledáván.
- TRP s implicitním zřetězením je vhodná v aplikacích, v nichž se operace **Delete** nepoužívá příliš často.
- **Maximální kapacita** TRP pro rozsah pole <0..Max> je Max (o 1 menší než počet prvků pole) – alespoň jeden prvek musí zůstat jako *zarážka* vyhledávání.
- Pro efektivní použití TRP s implicitním zřetězením se pole tabulky dimenzuje tak, aby její maximální zaplnění (dané poměrem $\mathbf{N_{akt}/Max}$) nebylo větší než cca 0,6-0,7.

Hodnocení metod vyhledávání

- Sekvenční vyhledávání — n
- Binární vyhledávání v seřazeném poli — $\log_2 n$
- Binární vyhledávácí strom — $\log_2 n$
  - při degradaci na seznam — n
- Vyvažovaný BVS (např. AVL) — $\log_2 n$
- TRP — 1
  - při maximální kolizi klíčů — n
  - při maximální kolizi a vyváženém stromu — $\log_2 n$

Hashovací funkce

- Též hešovací, hašovací, rozptylovací (u TRP) funkce
- Pro danou zprávu vrací její otisk (hash)
- Je vhodným kandidátem pro mapovací funkci TRP

Vlastnosti:

- Vstup **libovolné délky** transformuje na výstup **fixní délky**
- **Determinismus** – pro stejný vstup vrací vždy stejný výstup
- **Rychlost** – není výpočetně náročné funkci vyčíslit
- **Malá změna na vstupu** (např. jednoho bitu) způsobí **velkou změnu na výstupu** (tzv. Avalanche Effect – efekt laviny)
- Navržena tak, aby měla **co nejméně kolizí**

Hashovací funkce - využití

Využití hashovacích funkcí:

- **Zajištění integrity dat** – kontrolní součty (sítě, archivy aj.)
- **Zajištění nepopiratelnosti dat** – elektronický podpis
  = hash zprávy zašifrovaný privátním klíčem podepisujícího
- **Zajištění důvěrnosti dat**
  - Ukládání hesel – operační systémy, informační systémy, aplikace
    (při přihlášení se hash zadaného hesla porovná s uloženým hashem)
  - Součást kryptografických protokolů (šifrovací protokoly TLS/SSL)
- **Rychlá identifikace souborů** – souborové systémy, distribuované systémy, forenzní analýza digitálních dat
- **Tabulky s rozptýlenými položkami** (hashovací tabulky)

Kryptografické hashovací funkce

Aby byla funkce použitelná pro kryptografické účely, musí být výpočetně nezvládnutelné v „rozumném čase“:

- Z výstupu spočítat původní vstup (**1st Preimage Resistance**)
- Pro daný hash najít další vstup, který povede na stejný hash (**2nd Preimage Resistance**)
- Najít dva vstupy, které povedou ke kolizi – stejnému hashi (**Collision resistence**)

Příklady hashovacích funkcí vhodné pro kryptografické účely: ~~MD5~~*, ~~SHA-1~~*, rodina SHA-2, rodina SHA-3, bcrypt, scrypt, …

* - dnes již považovány za jednoduše prolomitelné

Další typy hashovacích funkcí

- **Fuzzy hashing / Similarity hashing** – analýza podobnosti: Je naopak žádoucí, aby dva podobné vstupy měly podobný hash
  - SSDEEP, sdhash, TLSH
- **Klouzavé hashovací funkce** (rolling hash functions) – Efektivní výpočet hodnot posouvajícího se okna nad vstupními daty
  - Adler32, CRC, Rabin-Karpův hash, Spamsum
- **Percepční hashování** (perceptual hashing) – detekce podobných multimediálních souborů (obrázky, zvuk)
  - pHash, dHash, aHash