Algoritmizace Hashing

June 5, 2012

Hashovací tabulky jsou datové struktury, které používají hashovací funkci k vyhledávání dat. Hashovací funkce $h(k)$ zobrazuje množinu klíčů k do intervalu adres $\langle a_{min}, a_{max} \rangle$.

- Volba hashovací funkce pro celá čísla:
  - Multiplikativní (m je prvočíslo)

$$h(k) = round(\frac{k}{2^w \cdot m}) \tag{1}$$

  - Modulární

$$h(k) = k\%n \tag{2}$$

  - Kombinovaná

$$h(k) = round(c \cdot k)\%m, c \in \langle 0, 1 \rangle \tag{3}$$

- Pro řetězce:
  - Hornerovo schéma

$$h(k) = k_n \cdot a^n + ... + k_0 \cdot a^0 \tag{4}$$

0.1 Řešení kolizí

Kolize vzniká když $h(k)$ vrací pro různé klíče stejnou hodnotu.

0.1.1 Hashování se separovanými řetězci

Hlavní myšlenka je, že pro každou možnou adresu vytvoříme spojový seznam a do něj přidáváme všechny prvky s danou adresou. Demonstrace v A.1.

- Výhody
  - Nemusíme znát předem počet prvků
- Nevýhody
  - Potřebujeme dynamické přidělování paměti
  - Potřebuje navíc paměť na ukazatele a na tabulku (heads v Figure A.1)

Složitost operací:

- Vkládání $O(1)$ (nebo složitost vložení do struktury kterou používáme místo spojového seznamu)
- Vyhledání $O(n)$ (nejhorší případ kdy jsou všechny prvky v kolizi)

0.1.2 Otevřené hashování

Hlavní myšlenka je, že pokud mám zhruba odhad počtu prvků které budu vkládat, tak mohu ušetřit na zbytečném ukládání ukazatelů tím, že budu ukládat kolize rovnou do tabulky. To může být provedeno následujícími způsoby:

- Lineární přidávání (když nastane kolize prvek se vloží na další volnou pozici). Demonstrace v Figure A.2.

  Složitost operací:

  - Vkládání $O(n)$ (nejhorší případ kdy jsou všechny prvky v kolizi)
  - Vyhledání $O(n)$ (nejhorší případ kdy jsou všechny prvky v kolizi)

- Dvojité hashování (když nastane kolize použije se druhá hashovací funkce, pomocí níž se spočítá offset a prvek se vloží na pozici kam měl být dán + offset). Demonstrace v Figure A.3.

  Složitost operací:

  - Vkládání $O(n)$ (nejhorší případ kdy jsou všechny prvky v kolizi v první i druhé hash funkci)
  - Vyhledání $O(n)$ (nejhorší případ kdy jsou všechny prvky v kolizi v první i druhé hash funkci)

0.1.3 Srůstající hashování

Hlavní myšlenka je omezení clusterování otevřeného hashování přidáním ukazatele pro každý prvek v tabulce. Možnosti provedení:

- Bez pomocné paměti (prvky se vkládají na pozice v tabulce)
  - LISCH (late insert standard coalesced hashing)
  - EISCH (early insert standard coalesced hashing)
- S pomocnou pamětí (prvky se vkládají na k tomu určené místo tzv. sklep za adresovým prostorem tabulky)
  - LICH (late insert coalesced hashing)
  - EICH (early insert coalesced hashing)
  - VICH (variable insert coalesced hashing)

LISCH

Přidává myšlenku ukazatele na poslední volné místo v tabulce (inicializován na poslední pozici). Při kolizi se prvek přidá na místo kam ukazuje tento ukazatel a ukazatel se opět přesune na poslední volné místo. Prvek na pozici kde vznikla kolize si drží ukazatel na pozici kam byl vložen kolidující prvek, ten si zase drží odkaz na další prvek který má stejný hash atd. Demonstrace ve Figurách A.4, A.5, A.6 a A.7. Porovnání s EISCH v Figure A.11

EISCH

Liší se od LISCH pouze tím, že ukazatele vedou odshora dolů (tzn ukazatel prvku který je na správné pozici vede vždy na jeho poslední kolizi, ta si zase drží odkaz na předchozí kolizi atd). Demonstrace ve Figurách A.8, A.9 a A.10. Porovnání s LISCH v Figure A.11

LICH

Stejný princip jako LISCH s tím rozdílem, že ukazatel začíná v tzv. sklepě což je speciální místo vyhrazené pro ukládání kolizí mimo adresní prostor tabulky. Porovnání ve Figure A.12.

EICH

Stejný princip jako EISCH s tím rozdílem, že ukazatel začíná v tzv. sklepě což je speciální místo vyhrazené pro ukládání kolizí mimo adresní prostor tabulky. Porovnání ve Figure A.12.

VICH

Algoritmus VICH připojuje prvek na konec řetězce, pokud řetězec končí ve sklepě, jinak na místo, kde řetězec opustil sklep. Porovnání ve Figure A.12.

0.1.4 Univerzální hashování

Místo jedná hashovací funkce máme konečnou množinu hashovacích funkcí H mapujících klíče do množiny $\{0, ..., m-1\}$. Množina funkcí H je univerzální, pokud pro každou dvojici různých klíčů $x, y \in U$ je počet hashovacích funkcí z množiny H, pro které $h(x) = h(y)$, nejvýše $|H|/m$. Pravděpodobnost kolize při náhodném výběru funkce $h(k)$ z množiny univerzálních hashovacích funkci H tedy není vyšší než pravděpodobnost kolize při náhodném a nezávislém výběru dvou stejných hodnot z intervalu $\{0, 1, \ldots, m-1\}$ tedy $1/m$. Při prvním spuštění programu jednu náhodně zvolíme. Funkci pak náhodně měníme jen v případě, že počet kolizí převyšuje přípustnou mez. V tomto případě je samozřejmě potřeba přehashovat celou tabulku.

Appendix A

Obrázky

A.1 Hashování se separovanými řetězci

- $h(k) = k \bmod 3$
- posloupnost : 1, 5, 21, 10, 7

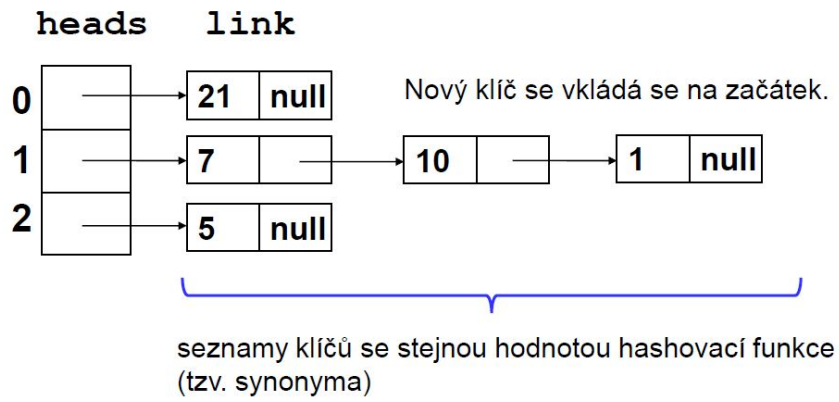

Figure A.1: Hashování se separovanými řetězci

A.2 Otevřená hashování

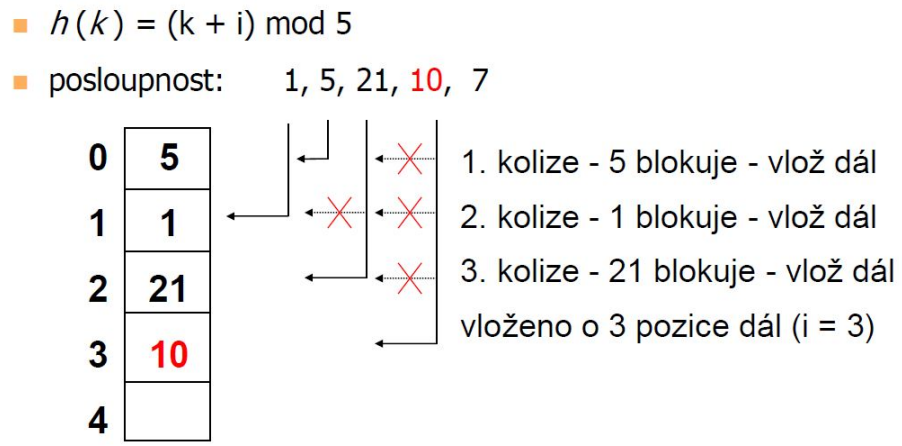

Figure A.2: Lineární přidávání

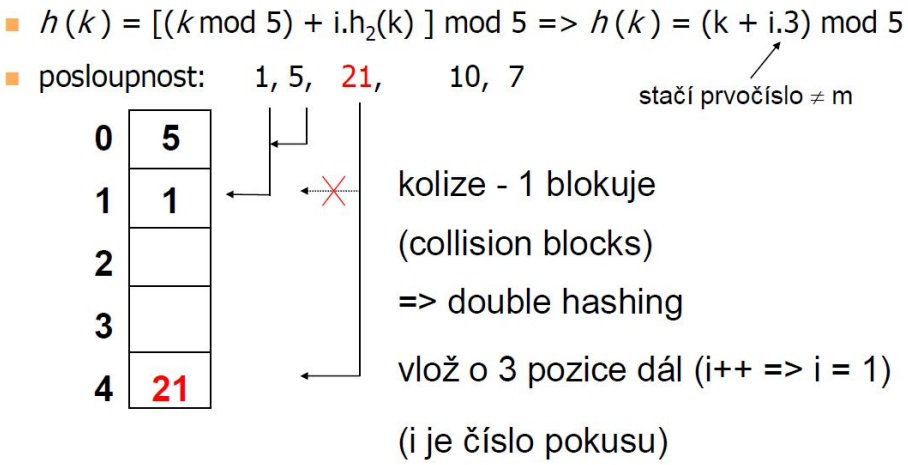

Figure A.3: Dvojité hashování

A.3 LISCH

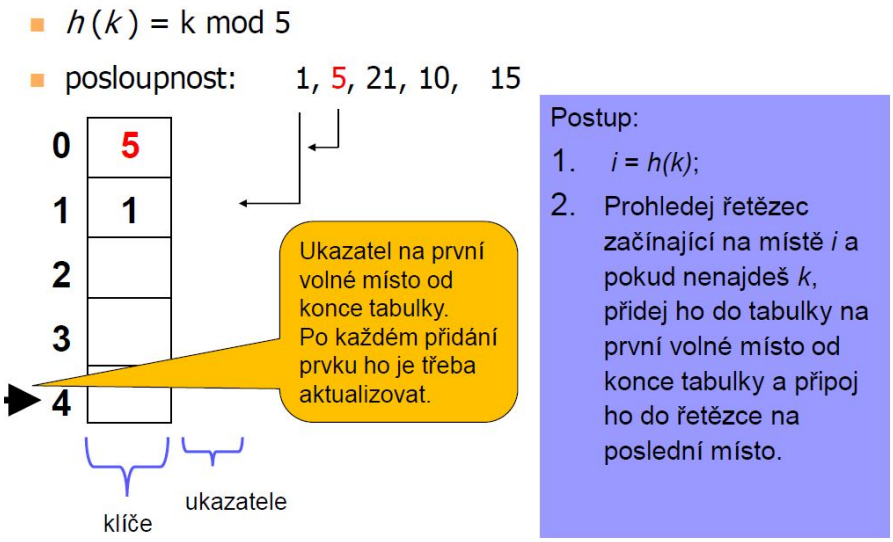

Figure A.4: LISCH 1

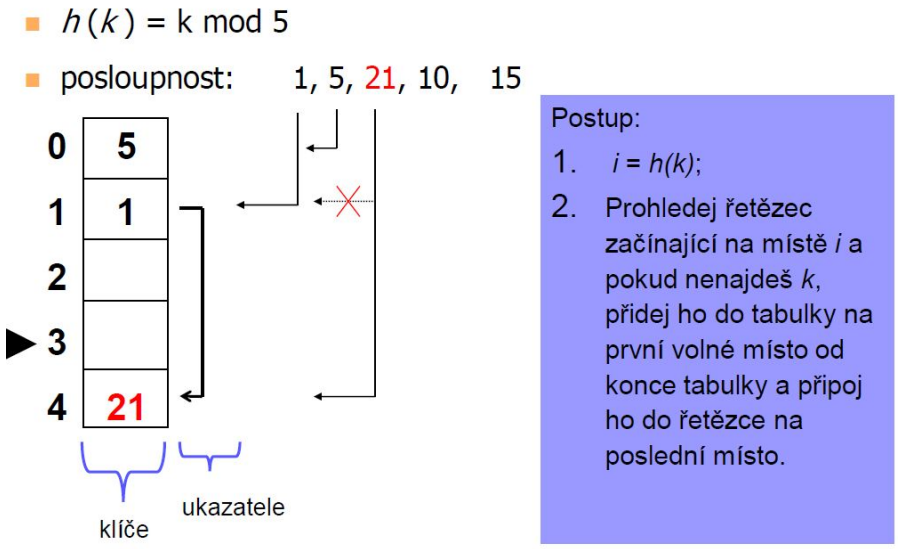

Figure A.5: LISCH 2

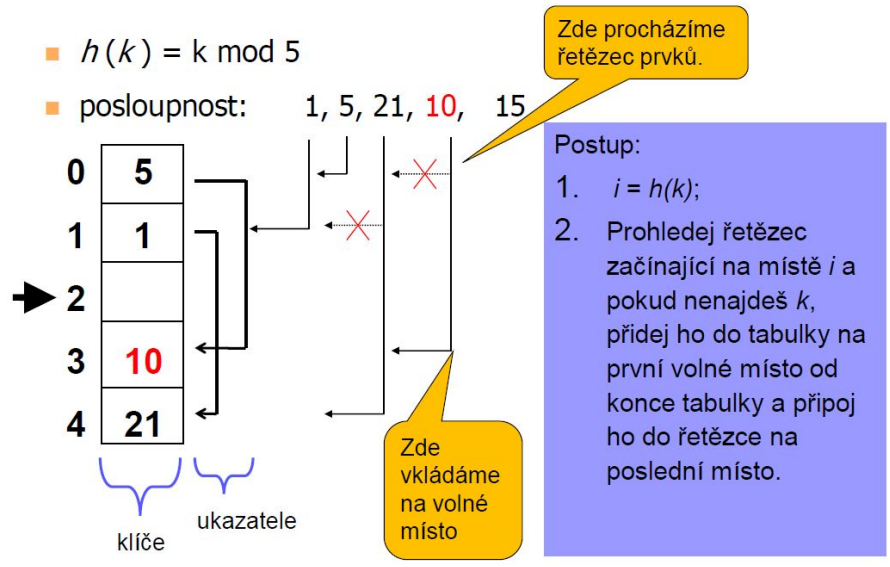

Figure A.6: LISCH 3

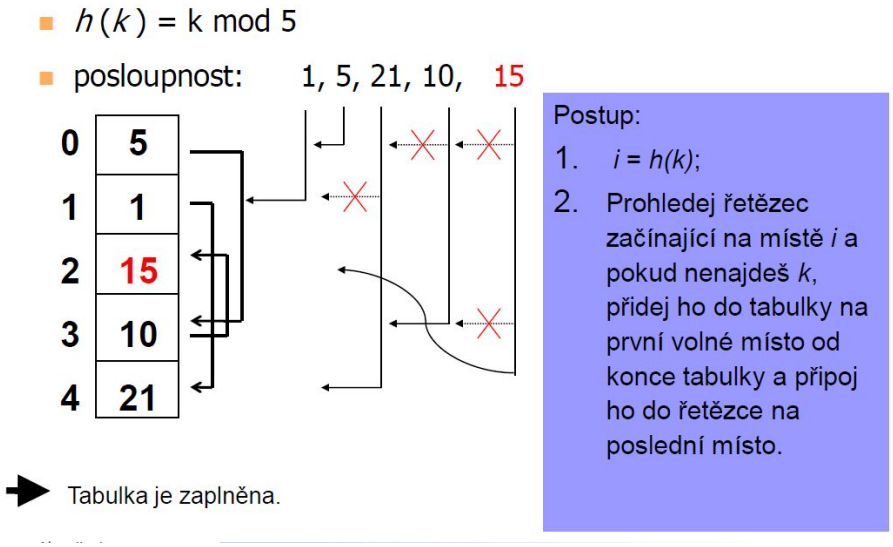

Figure A.7: LISCH 4

A.4 EISCH

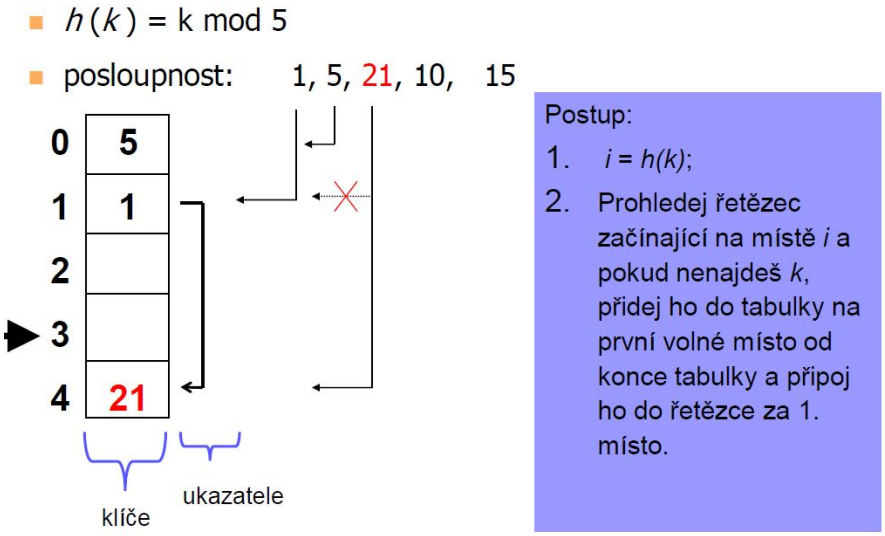

Figure A.8: EISCH 1

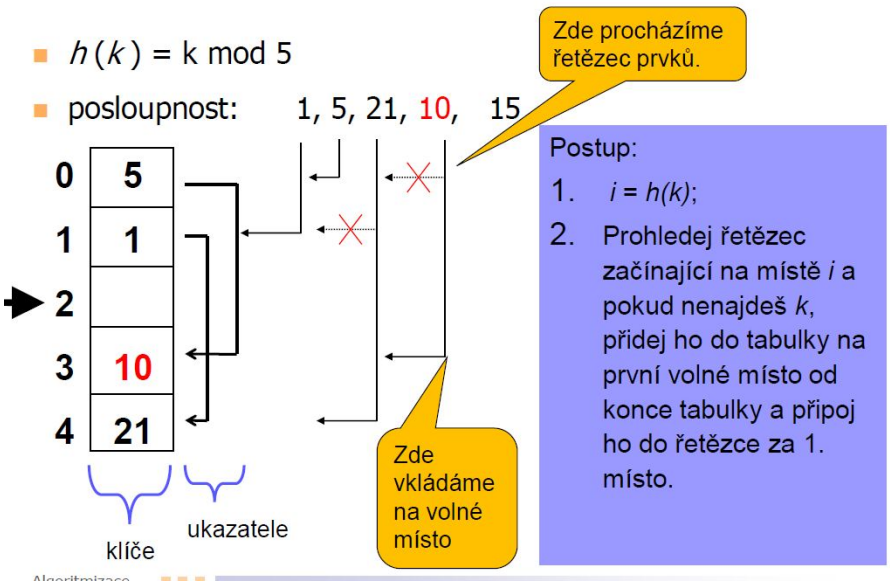

Figure A.9: EISCH 2

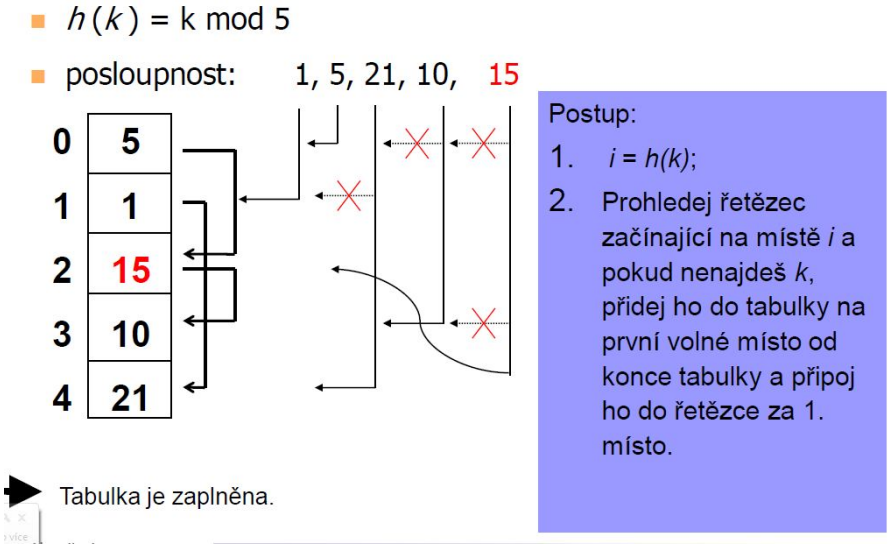

Figure A.10: EISCH 3

A.5 LISCH vs EISCH

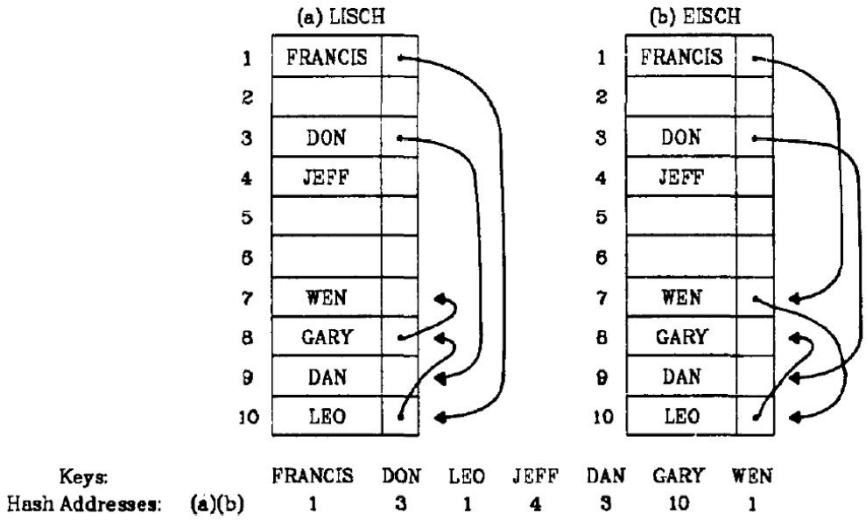

Figure A.11: LISCH vs EISCH

A.6 LICH vs EICH vs VICH

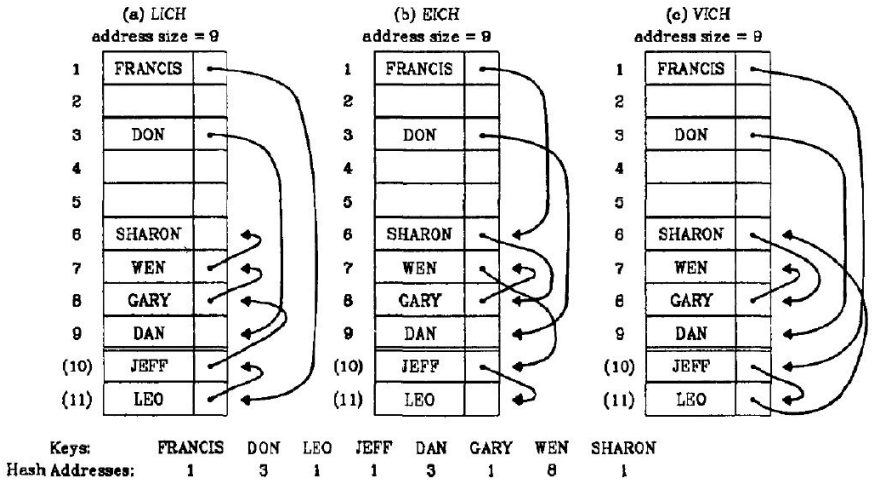

Figure A.12: LICH vs EICH vs VICH