# Perfektní hešování

P. Bezstarosti*
K. Burešová**
M. Pokorný***

*Gymnázium Dobruška, **Gymnázium Česká Lípa, ***Střední škola aplikované kybernetiky, s.r.o.

***mp@decin.cz

**Abstrakt:**

V případech, kdy je potřeba uložit velké množství málo se měnících dat, ke kterým je potřeba rychlý přístup, je neefektivní vyhledávat procházením v O(n). Často používaný způsob urychlení přístupu k těmto datům je hešovací tabulka. Problémem hešovacích tabulek jsou kolize, které zpomalují vyhledávání a zvyšují paměťovou náročnost implementace. V našem miniprojektu jsme prozkoumali běžné způsoby generování perfektních hešovacích tabulek - takových, ve kterých nevznikají kolize.

Klíčová slova: Neměnná data, perfektní hešovací tabulky, rychlý přístup k datům, databáze, algoritmus

# 1 Úvod

Hešování je dnes běžně používáno především k rychlému vyloučení shodnosti různých dat nebo zrychlení přístupu k předem daným, nebo málo se měnícím datům, jako například klíčových slov programovacích jazyků, URL ve webových vyhledávačích nebo datových sad v data miningu. V naší práci jsme se zaměřili na obvyklé algoritmy používané k vytvoření perfektní nebo minimální perfektní hešovací funkce, které jsou vhodné především pro poměrně malé množiny klíčů. Implementovali jsme tři takovéto algoritmy v C++ a PHP. Knihovna C Minimal Perfect Hashing Library (domovská stránka `http://cmph.sourceforge.net`) umožňuje jednoduché použití efektivních hešovacích algoritmů pod licencí LGPL.

# 2 Perfektní hešovací tabulka (perfect hash table)

Definujme si několik důležitých pojmů.

*Tabulkou* rozumíme třísloupcovou strukturu. První sloupec obsahuje indexy v rozmezí $\langle 0; Q-1 \rangle$, kde Q je počet záznamů. Druhý sloupec obsahuje celočíselné klíče, třetí obsahuje datový offset klíče.

*Hešovací funkce* slouží k převádění klíčů na relativně malé indexy.

*Kolize* je situace, kdy hešovací funkce má pro různé klíče stejnou hodnotu index. Kolize je třeba nějak ošetřit, nejčastěji pomocnou tabulkou, do které se ukládají kolidující klíče.

*Perfektní hešovací tabulka (PHT)* je navržená tak, aby v ní nedocházelo ke kolizím, tedy se dá na jeden dotaz zjistit, zda klíč v tabulce obsažen je nebo ne. Využívá se pro dlouhodobě neměnná data, např. jízdní řády.

*Minimální PHT* je taková PHT, která má počet záznamů stejný jako počet klíčů. Tato tabulka je nejoptimálnější výsledek při hešování kvůli menším nárokům.

# 3 Efektivní hešovací funkce

Efektivní hešovací funkce využíváme k tvorbě PHT. Nesmí v nich tedy docházet ke kolizím, to znamená, že funkce musí každému unikátnímu klíči přiřadit jeho unikátní index, tedy musí být prostá.

Při tvorbě minimální PHT je navíc potřeba, aby hešovací funkce byla surjektivním zobrazením, protože ke každému záznamu z tabulky musí být přiřazen nejméně jeden klíč.

# 4 Hledání hešovacích funkcí

Při hledání hešovacích funkcí jsme vycházeli z následujících obecných vzorců.

$$h(key) = A \cdot key \; mod \; Q$$

$$h(key) = \left\lfloor \frac{A}{key} \right\rfloor mod \; Q$$

Pokud u prvního vzorce dostaneme mezi klíči dva, které dávají při dělení $Q_0$ = n (kde n je počet klíčů) stejný zbytek, nemůžeme nalézt takové *A*, abychom dostali minimální PHT.

Více jsme se tedy věnovali druhému vzorci, pro který jsme vytvořili algoritmus, který jsme dále optimalizovali. V [1] jsme našli vzorec pro nejmenší možné $A$:$A \geq \left\lfloor \frac{(n-2)w_1 w_n}{w_n - w_1} \right\rfloor$.
Při optimalizaci jsme se zaměřili na rychlé určení a přeskakování těch *A*, která nejsou řešením.
Na internetu jsme také našli algoritmus CHM.

## Algoritmus CHM

CHM (Czech, Havas, Majewski) je algoritmus prezentovaný v [2] vytvářející minimální perfektní hešovací tabulky zachovávající pořadí klíčů používající náhodné grafy. Pracuje v lineárním čase a k reprezentaci hešovací funkce potřebuje lineární množství dat. Oproti algoritmům, které jsme implementovali před ním, je velice rychlý, a narozdíl od AK mod Q vygeneruje minimální hešovací funkci vždy.

Algoritmus předpokládá, že klíče jsou slova, tedy posloupnosti znaků z určité abecedy.

První fáze algoritmu je fáze mapování. Nejprve se vygenerují tabulky $T_1$ a $T_2$ obsahující náhodná čísla pro znaky a jejich pozice ve slovech. „Otisk“ klíče z tabulky je potom součet všech prvků tabulky odpovídajících znakům klíče a jejich pozicím. Zvolí se *N* větší než množství slov a vytvoří se prázdný neorientovaný graf G. Dále se pro každý klíč vygeneruje

jeho „otisk“ z obou tabulek $T_1$ a $T_2$ modulo N a do grafu G se přidá hrana G a případně vrcholy z těchto „otisků“ (opět modulo N).

První fáze algoritmu se pak opakuje, dokud není graf G acyklický.

Druhá fáze algoritmu je fáze přiřazování. Každý vrchol grafu se označí za nenavštívený a přiřadí se mu hodnota G nula.

Poté se najde první nenavštívený vrchol N a provede se následující operace („navštívení“): pro každého nenavštíveného souseda S navštěvovaného vrcholu $V$ se provede přiřazení $G(S) \leftarrow \big(H(N,S) - G(N)\big) \bmod N$ a rekurzivně se navštíví soused $S$. Funkce $H(N,S)$ je pak číslo klíče, který má "otisky" $N$ a $S$, tedy klíče příslušejícího hraně $N$-$S$.

Když neexistuje žádný nenavštívený vrchol, algoritmus končí.

Heš slova L se pak spočítá jako:

$$\big(G(otisk\ v\ 1.tabulce) + G(otisk\ v\ 2.tabulce)\big) \bmod počet\ klíčů.$$

# 5 Simulační experimenty

## Významné letopočty v české historii

Ukládané letopočty: 863, 929, 995, 1212, 1348, 1415, 1609, 1620, 1781, 1848, 1914, 1918, 1939, 1945, 1968, 1989, 1993.

Hešovací funkce: $index = \left\lfloor \frac{38685101}{key} \right\rfloor mod\ 17$

| **Index** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| **Klíč** | 995 | 1989 | 1348 | 1415 | 1609 | 1968 | 1848 | 1918 | 929 |
| **Index** | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | |
| **Klíč** | 1212 | 1939 | 1620 | 1781 | 1993 | 863 | 1914 | 1945 | |

## Malá prvočísla

Ukládali jsme prvočísla do 50: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47

Hešovací funkce: $index = \left\lfloor \frac{804885}{key} \right\rfloor mod\ 15$

| **Index** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Klíč** | 23 | 11 | 19 | 37 | 29 | 3 | 17 | 2 | 7 | 13 | 47 | 41 | 5 | 43 | 31 |

## „Nešťastná“ prvočísla

Zadefinovali jsme si pojem „nešťastná“ prvočísla množinou D:

$$P = \{n \in N | \forall m; 1 < m < n : m \nmid n\}$$

$$D = \{n \in P | n \bmod 100 = 13\}$$

Pak jsme vytvářeli hešovací tabulku pro ta z nich, která jsou menší než 5000 (13, 113, 313, 613, 1013, 1213, 1613, 1913, 2113, 2213, 2713, 3313, 3413, 3613, 4013, 4513, 4813).

Hešovací funkce: $index = \left\lfloor \frac{16973294}{key} \right\rfloor mod\ 17$

| **Index** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| **Klíč** | 2713 | 1213 | 2213 | 4513 | 13 | 3613 | 3313 | 4813 | 2113 |
| **Index** | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | |
| **Klíč** | 3413 | 1013 | 113 | 613 | 413 | 313 | 1913 | 1613 | |

# 6 Shrnutí

Podařilo se nám napsat a optimalizovat vlastní algoritmy pro hledání minimálních perfektních hešovacích funkcí. Na internetu jsme našli algoritmus CHM, který hešovací funkce hledá velmi dobře.
Algoritmy jsme testovali na třech úlohách s pevně stanovenými daty. Pro všechny tři se nám podařilo najít vhodnou hešovací funkci a vytvořit minimální PHT.

# Poděkování

Děkujeme Jaromíru Kukalovi za ochotnou pomoc při realizaci našeho miniprojektu a za užitečné informace, náměty a rady.

Dále děkujeme FJFI ČVUT za poskytnutí zázemí a možnosti zúčastnit se tohoto projektu.

# Reference:

[1] Jeschke, G.: Reciprocal hashing: a method for generating minimal perfect hashing function, *Communication of the ACM*, 1981, 24(12):829-833

[2] Czech, Z. J. - Havas, G. - Majewski, B. S.: An optimal algorithm for generating minimal perfect hash functions, *Information Processing Letters*, 1992, 43(5):257-264