Kukačkové hešování

Jaroslav Kňap, Gymnázium Turnov, knap.jaroslav@gmail.com
Matěj Plch, Gymnázium Blansko, plch.matej@gmail.com
Lukáš Černý, Gymnázium Turnov, lukascerny00@gmail.com

Abstrakt:

Pro uložení velkých objemů dat se často používají tzv. hešovací funkce. Hešovací funkce poskytují značně lepší časy v přístupech dat než lineární prohledávání, ovšem jejich problém je možnost kolizí hodnot hešovacích funkcí. V našem projektu jsme se zabývali teorií a posléze i implementací algoritmu známého jako kukačkové hešování, který řeší tyto kolize.

1 Úvod

Nejprve si definujme důležité pojmy:

Hešovací funkce slouží k převádění vstupních dat (v našem případě čísel) na jednoduché indexy.

Kolize je určitý moment, kdy jedna hešovací funkce přidělí různým číslům stejný index. Tuto situaci je nutno nějak ošetřit a právě k tomuto účelu se dá využít právě kukačkové hešování.

Kukačkové hešovaní je algoritmus který využívá dvou různých hešovacích funkcí a tabulek k urychlení vyhledávání, přidávání a odstraňování elementů. Také velmi elegantně řeší problém kolizí a využívá ho coby základ jak pro svůj princip, tak i název. Vyznačuje se konstantními přístupovými časy k datům, kdy nezávisle na objemu dat se k požadovaným informacím dostaneme nejvýše na dva pokusy, přičemž paměťová náročnost je v průměru dvojnásobná v poměru k uloženým datům.

2 Princip kukačkového hešování

K ukládání dat slouží dvě hešovací tabulky o velikosti r, kde r náleží do množiny prvočísel a dle potřeby se zvětšuje. Při pokusu o zápis čísla x do první tabulky se jeho pozice v tabulce určí pomocí hešovací funkce $h_1(x)$. Pokud je určená pozice prázdná, hodnota se zapíše a práce je hotova. Pokud je však tato pozice obsazená, číslo se na tuto pozice zapíše a původní hodnota se zapíše do druhé tabulky na index určený hešovací funkcí $h_2(x)$, přičemž číslo, které se zde nacházelo předtím, je přesunuto do první tabulky podle hešovací funkce $h_1(x)$. Podle tohoto principu dostala tato technika název, neboť je analogií chování kukaček, které po vylíhnutí v cizím hnízdě si toto hnízdo přivlastní a vyhodí ostatní vajíčka pryč.
Problém nastane, jestliže dojde k zacyklení procesu přehazování prvků, kdy nelze najít místo v tabulce pro určitý prvek, případně hledání volného místa trvá příliš dlouho. Řešením je

přehešování obou tabulek, kdy se určí nová velikost tabulek podle nového prvočísla *r* a určí se nové hešovací funkce $h_1(x)$ a $h_2(x)$. Následně se již zapsaná data přepíší do nových tabulek podle nových hešovacích funkcí. V zájmu výpočetní a paměťové efektivity je vhodné, aby nové tabulky byly alespoň dvakrát větší než ty předcházející. Zabrání se tak příliš častému přehešovávání tabulek při zachování rozumné paměťové náročnosti.

3 Vlastní implementace

Kukačkové hešování jsme realizovali ve vývojovém prostředí Borland Delphi.

Uvedeme několik příkladů použitých algoritmů:

Funkce `lookup` - slouží k vyhledávání dat

```
function lookup(x: integer):boolean;
begin
        if (tabulka1[h1(x)] = x) or (tabulka2[h2(x)]=x) then lookup := true
        else lookup := false;
end;
```

Procedura `insert` – slouží k uložení zadané hodnoty

```
procedure insert(x: integer);
var i, p, hash: integer;
begin
        if lookup(x) then exit;
        for i := 1 to 100 do begin
                hash := h1(x);
                if  tabulka1[hash] = -1 then begin
                        pocet1:=pocet1+1;
                        tabulka1[hash] := x;
                        exit;
                end;
                p := x;
                x := tabulka1[hash];
                pole1[hash] := p;
                hash := h2(x);
                if  tabulka2[hash] = -1 then begin
                        pocet2:=pocet2+1;
                        tabulka2[hash] := x;
                        exit;
                end;
                p := x;
                x := tabulka2[hash];
                tabulka2[hash] := p;
        end;
        if hashing then begin
           uspech:=false;
           exit;
        end else begin
           rehash;
           insert(x);
        end;
end;
```

Použité hešovací funkce:

$$h_1(x) = x \bmod r \qquad h_2(x) = \left\lfloor \frac{x}{r} \right\rfloor \bmod r$$

4 Metodika testování

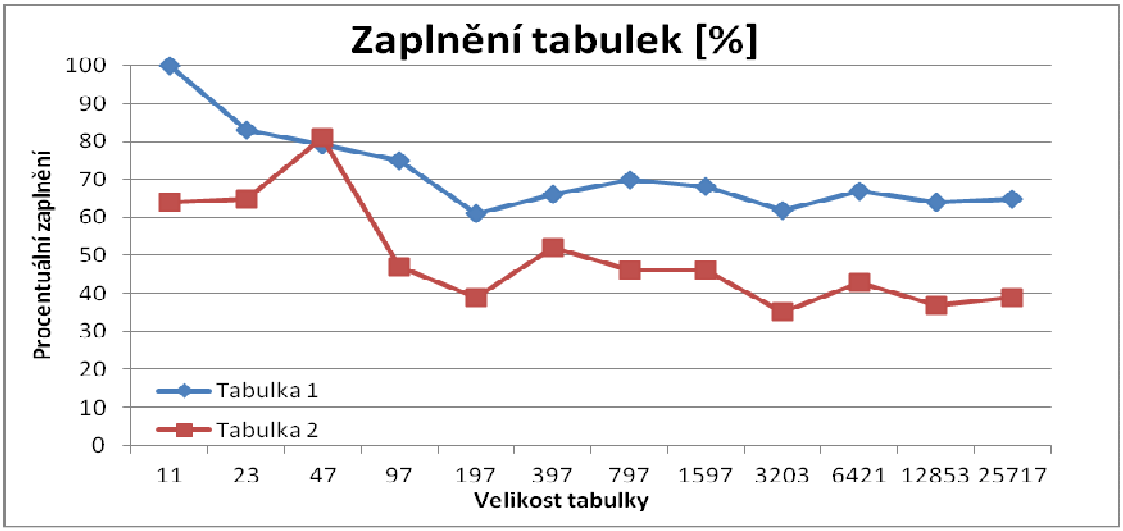
Jako hlavní testovací metodu výkonnosti tohoto algoritmu jsme použili generátor pseudonáhodných čísel, který generoval čísla z intervalu (0… 9 999 999 999), než se dosáhlo dvanáctého přehešování tabulek. Tato hodnota byla zvolena z důvodu přiměřené časové náročnosti. V průběhu chodu programu jsme zaznamenávali procentuální obsazenost tabulek před každým přehešováním.

Také jsme se zamysleli nad kvalitou generátoru pseudonáhodných čísel a pro zajímavost jsme vyzkoušeli efektivitu algoritmu i na jiném souboru dat, konkrétně na množině náhodných prvočísel.

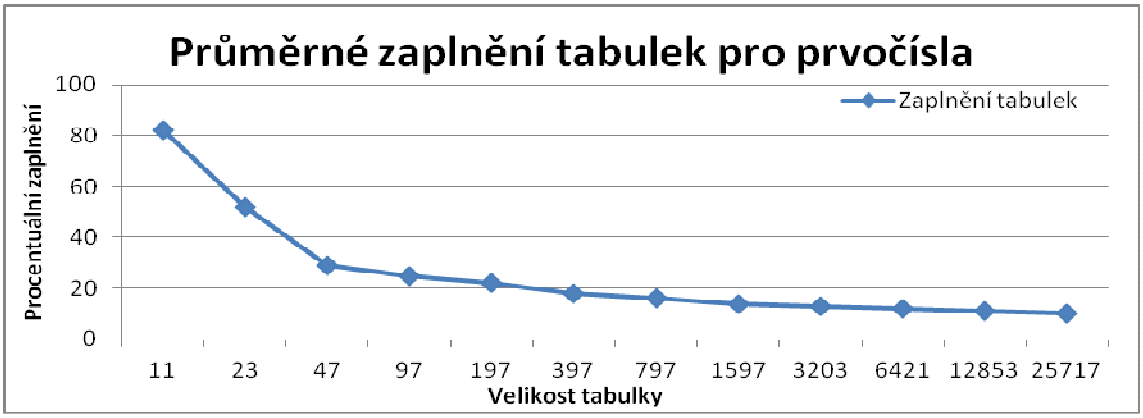
5 Naměřené výsledky

Počet hodnot, které byly zapsány do tabulek, osciloval okolo 25 000. Následující grafy ukazují procentuální využití tabulek.

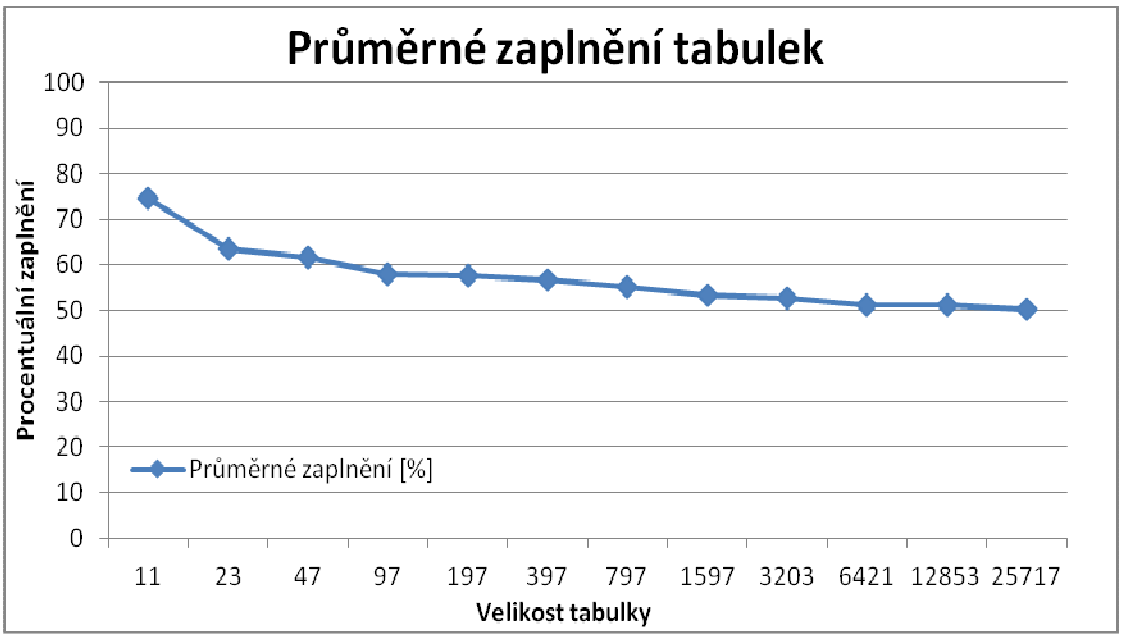
Graf 1: Ukázka konkrétního měření



Graf 2: Průměrné zaplnění tabulek u prvočísel



Graf 3: Průměrné zaplnění tabulek před přehešováním



Z výsledků vyplývá, že naše implementace kukačkového hešování je schopna využít průměrně 50 % dostupného místa v tabulkách, než je třeba tabulky přehešovat. Záleží však na použitém souboru dat, při použití prvočísel jako vstupních dat dochází k velkému plýtvání pamětí. Tento nedostatek je způsoben použitím nedokonalých hešovacích funkcí.

Výsledné hodnoty odpovídají naměřeným hodnotám v originální práci [1] autorů kukačkového hešování.

6 Shrnutí

Podařilo se nám pochopit a také vytvořit vlastní implementaci algoritmu kukačkového hešování s využitím vývojového prostředí Borland Delphi. K tomuto účelu jsme využili poměrně jednoduché hešovací funkce, které pracují s prvočísly.

Poděkování

Děkujeme FJFI za uspořádání Týdne vědy na Jaderce a poskytnutí zázemí nutného k uskutečnění našeho projektu.

Dále děkujeme našemu garantovi doc. Ing. Jaromíru Kukalovi, Ph. D. za ochotnou pomoc a rady při realizaci našeho projektu.

Reference:

[1] PAGH, R. – RODLER, F. F.: *Cuckoo Hashing,* [online], 2001, Dostupné z: http://www.it-c.dk/people/pagh/papers/cuckoo-jour.pdf, [citováno 18. června 2012]

[2] PAGH, R.: *Cuckoo Hashing for Undergraduates,* [online], 2006, Dostupné z: http://www.it-c.dk/people/pagh/papers/cuckoo-undergrad.pdf. [citováno 18. června 2012]