Úvod.
Základní problém: Reprezentace množin a operace s nimi. V řadě úloh a algoritmů je tento podproblém rozhodující pro složitost řešení, protože tyto operace se mnohokrát opakují. Proto je třeba navrhnout pro tyto úlohy co nejefektivnější algoritmy (každý ušetřený čas mnohonásobným opakováním začne hrát důležitou roli). To vede k detailní analýze složitosti v závislosti na vnějších okolnostech. Nelze říct, že některý algoritmus je nejlepší, protože za určitých okolností může být 'méně efektivní' algoritmus výhodnější.

Řešíme tzv. slovníkový problém: Dáno univerzum U, máme reprezentovat $S \subseteq U$ a navrhnout algoritmy pro následující operace
MEMBER(x) – zjistí, zda $x \in S$, a nalezne jeho uložení
INSERT(x) – když $x \notin S$, pak vloží x do struktury reprezentující S
DELETE(x) – když $x \in S$, pak odstraní x ze struktury reprezentující S.
Efektivita algoritmu: časová složitost, prostorová složitost;
vyšetřené buď v nejhorším případě nebo v průměrném případě nebo amortizovaně.

Literatura:
K. Mehlhorn: Data Structures and Algorithms 1: Sorting and Searching, Springer 1984
http://www.mpi-sb.mpg.de/~mehlhorn/DatAlgbooks.html

J. S. Vitter, W.-Ch. Chen: Design and Analysis of Coalesced Hashing, Oxford Univ. Press, 1987

Hašování

Pomocí bitového pole můžeme rychle implementovat operace **MEMBER**, **INSERT** a **DELETE**.

Nevýhoda: když je velké univerzum, pak je prostorová složitost v nejlepším případě ohromná, ve špatném případě nelze pole zadat do počítače.

Hašování chce zachovat rychlost operací, ale odstranit paměťovou náročnost. První publikovaný článek o hašování je od Dumney z roku 1956, první analýza hašování pochází od Petersona z roku 1957, ale existuje technická zpráva od IBM o hašování z roku 1953.

Základní idea: Dáno univerzum U a množina $S \subseteq U$ tak, že $|S| << |U|$. Máme funkci $h : U \to \{0, 1, \ldots, m-1\}$ a množinu S reprezentujeme tabulkou (polem) s m řádky tak, že $s \in S$ je uložen na řádku $h(s)$.

Nevýhoda: mohou existovat různá $s, t \in S$ taková, že $h(s) = h(t)$ - tento jev se nazývá kolize.

Hlavní problém: řešení kolizí.

Základní řešení: použijeme pole o velikosti $[0..m-1]$ a i-tá položka pole bude spojový seznam obsahující všechny prvky $s \in S$ takové, že $h(s) = i$. Toto řešení se nazývá hašování se separovanými řetězci.

Příklad: $U = \{1, 2, \ldots, 1000\}$, $S = \{1, 7, 11, 53, 73, 141, 161\}$ a funkce je $h(x) = x \bmod 10$. Pak

$$P(0) = P(2) = P(4) = P(5) = P(6) = P(8) = P(9) = \emptyset,$$
$$P(7) =< 7 >, \quad P(3) =< 53, 73 >, \quad P(1) =< 1, 141, 11, 161 > .$$

Seznamy nemusí být uspořádané.

Algoritmy operací.

MEMBER(x):

Spočítáme $i := h(x)$, $t := NIL$
if i-tý seznam je neprázdný **then**
  t :=první prvek i-tého seznamu
  while $t \neq x$ a $t \neq$poslední prvek i-tého seznamu **do**
    t :=následující prvek i-tého seznamu
  enddo
endif
if $t = x$ **then Výstup**: $x \in S$ **else Výstup**: $x \notin S$ **endif**

INSERT(x):

Spočítáme $i := h(x)$, $t := NIL$
if i-tý seznam je neprázdný **then**
  t :=první prvek i-tého seznamu
  while $t \neq x$ a $t \neq$poslední prvek i-tého seznamu **do**
    t :=následující prvek i-tého seznamu
  enddo
endif
if $t \neq x$ **then** vložíme x do i-tého seznamu **endif**

DELETE(x):
Spočítáme $i := h(x)$, $t := NIL$
if i-tý seznam je neprázdný **then**
  t :=první prvek i-tého seznamu
  while $t \neq x$ a $t \neq$poslední prvek i-tého seznamu **do**
    t :=následující prvek i-tého seznamu
  enddo
endif
if $t = x$ **then** odstraníme x z i-tého seznamu **endif**

Předpoklad: V následující analýze předpokládáme, že hodnota funkce $h(x)$ je spočitatelná v čase $O(1)$.

Složitost v nejhorším případě: V nejhorším případě operace vyžadují čas $O(|S|)$ (všechny prvky jsou v jednom seznamu). Požadovaná paměťová náročnost $O(m + |S|)$ (předpokládáme, že reprezentace prvku $s \in S$ vyžaduje paměť $O(1)$)
paměť není efektivně využitá 

Předpoklady analýzy oček. délky řetězců: Spočítáme očekávanou délku řetězců za předpokladů

(1) h je rychle spočitatelná (tj. $O(1)$) a neměnná během výpočtu;
(2) h rozděluje univerzum U rovnoměrně (tj. $-1 \leq |h^{-1}(i)| - |h^{-1}(j)| \leq 1$ pro $i, j \in \{0, 1, \ldots, m-1\}$);
(3) S je náhodně vybraná z univerza U (tj. pro dané $n = |S|$ jsou všechny podmnožiny U o velikosti n reprezentovanou množinou S se stejnou pravděpodobností);
(4) každý prvek z U má stejnou pravděpodobnost být argumentem operace;
(5) velikost reprezentované množiny je výrazně menší než velikost univerza.

Značení: Použité značení: $|S| = n$, m =počet řetězců, $|U| = N$,
$\ell(i)$ =délka i-tého řetězce, $\alpha = \frac{n}{m}$ faktor naplnění (load factor)

Důsledky předpokladů: Důsledky předpokladů:

$$\text{Prob}(h(x) = i) = \tfrac{1}{m} \text{ pro všechna } x \in U \text{ a všechna } i = 0, 1, \ldots, m-1;$$
$$\text{Prob}(\ell(i) = l) = p_{n,l} = \binom{n}{l} \left(\tfrac{1}{m}\right)^l \left(1 - \tfrac{1}{m}\right)^{n-l} \text{ pro všechna } i = 0, 1, \ldots, m-1.$$

Vysvětlení: i-tý řetězec má délku l, právě když existuje podmnožina $A \subseteq S$ taková, že $|A| = l$ (těchto možností je $\binom{n}{l}$), pro každé $x \in A$ platí $h(x) = i$ (pravděpodobnost tohoto jevu je $\left(\frac{1}{m}\right)^l$) a pro každé $x \in S \setminus A$ platí $h(x) \neq i$ (pravděpodobnost tohoto jevu je $\left(1 - \frac{1}{m}\right)^{n-l}$). To znamená, že jev má binomiální rozdělení.

Očekávaná délka řetězců.

$$E(l) = \sum_{l=0}^{n} l p_{n,l} = \sum_{l=0}^{n} l \binom{n}{l} \left(\frac{1}{m}\right)^{l} \left(1-\frac{1}{m}\right)^{n-l} =$$

$$\sum_{l=0}^{n} l \frac{n!}{l!\,(n-l)!} \left(\frac{1}{m}\right)^{l} \left(1-\frac{1}{m}\right)^{n-l} =$$

$$\frac{n}{m} \sum_{l=1}^{n} \frac{(n-1)!}{(l-1)!\,(n-l)!} \left(\frac{1}{m}\right)^{l-1} \left(1-\frac{1}{m}\right)^{n-l} =$$

$$\frac{n}{m} \sum_{l=1}^{n} \binom{n-1}{l-1} \left(\frac{1}{m}\right)^{l-1} \left(1-\frac{1}{m}\right)^{(n-1)-(l-1)} =$$

$$\frac{n}{m} \sum_{l=0}^{n-1} \binom{n-1}{l} \left(\frac{1}{m}\right)^{l} \left(1-\frac{1}{m}\right)^{n-1-l} =$$

$$\frac{n}{m} \left(\frac{1}{m} + 1 - \frac{1}{m}\right)^{n-1} = \frac{n}{m}.$$

[Tj. faktor naplnění \alpha]

Toto je standardní elementární výpočet očekávané hodnoty binomiálního rozdělení.

Výpočet druhého momentu.

$$E\left(l^2\right) = E\left(l\left(l-1\right)\right) + E\left(l\right),$$

$$E(l(l-1)) = \sum_{l=0}^{n} l\left(l-1\right) \binom{n}{l} \left(\frac{1}{m}\right)^{l} \left(1-\frac{1}{m}\right)^{n-l} =$$

$$\frac{n\left(n-1\right)}{m^2} \sum_{l=2}^{n} \binom{n-2}{l-2} \left(\frac{1}{m}\right)^{l-2} \left(1-\frac{1}{m}\right)^{(n-2)-(l-2)} =$$

$$\frac{n\left(n-1\right)}{m^2} \sum_{l=0}^{n-2} \binom{n-2}{l} \left(\frac{1}{m}\right)^{l} \left(1-\frac{1}{m}\right)^{n-2-l} =$$

$$\frac{n\left(n-1\right)}{m^2},$$

Sečte se na 1 podle binomické věty.

$$E\left(l^2\right) = \frac{n\left(n-1\right)}{m^2} + \frac{n}{m} = \frac{n}{m}\left(1 + \frac{n-1}{m}\right).$$

Výpočet rozptylu.

$$\operatorname{var}\left(l\right) = E\left(l - E\left(l\right)\right)^2 = E\left(l^2\right) - \left(E\left(l\right)\right)^2 =$$

$$\frac{n}{m}\left(1 + \frac{n-1}{m}\right) - \left(\frac{n}{m}\right)^2 = \frac{n}{m}\left(1 - \frac{1}{m}\right).$$

Shrneme výsledky:
Očekávaná délka řetězců je $\frac{n}{m}$ a rozptyl délky řetězců je $\frac{n}{m}\left(1-\frac{1}{m}\right)$. Toto jsou standardní elementární odvození druhého momentu a rozptylu binomiálního rozdělení.

Očekávaný nejhorší případ.

Spočítáme $E(NP)$ očekávanou délku maximálního řetězce.
Označme $\ell(i)$ délku i-tého řetězce. Pak

$$\text{Prob}\left(\max_i \ell(i) = j\right) = \text{Prob}\left(\max_i \ell(i) \geq j\right) - \text{Prob}\left(\max_i \ell(i) \geq j+1\right). \qquad \text{(\#1)}$$

Pak můžeme počítat:

$$\begin{aligned}
E(NP) = &\sum_j j\,\text{Prob}\left(\max_i \ell(i) = j\right) \overset{\text{(\#1)}}{=} \\
&\sum_j j\left(\text{Prob}\left(\max_i \ell(i) \geq j\right) - \text{Prob}\left(\max_i \ell(i) \geq j+1\right)\right) = \quad \text{[Roztrhnu na dvě sumy]} \\
&\sum_j j\,\text{Prob}\left(\max_i \ell(i) \geq j\right) - \sum_j j\,\text{Prob}\left(\max_i \ell(i) \geq j+1\right) = \\
&\sum_j j\,\text{Prob}\left(\max_i \ell(i) \geq j\right) - \sum_j (j-1)\,\text{Prob}\left(\max_i \ell(i) \geq j\right) = \quad \text{[Přeindexoval jsem]} \\
&\sum_j (j-j+1)\,\text{Prob}\left(\max_i \ell(i) \geq j\right) = \\
&\sum_j \text{Prob}\left(\max_i \ell(i) \geq j\right).
\end{aligned}$$

Vysvětlení: Při čtvrté rovnosti se v druhé sumě zvětšil index, přes který sčítáme, o 1, v páté rovnosti se k sobě daly koeficienty při stejných pravděpodobnostech ve dvou sumách.
Odtud

$$\begin{aligned}
&\text{Prob}(\max_i(\ell(i)) \geq j) = \\
&\qquad \text{Prob}\left(\ell(1) \geq j \vee \ell(2) \geq j \vee \cdots \vee \ell(m-1) \geq j\right) \leq \\
&\qquad \sum_i \text{Prob}\left(\ell(i) \geq j\right) \leq m\binom{n}{j}\left(\frac{1}{m}\right)^j = \\
&\qquad \frac{\prod_{k=0}^{j-1}(n-k)}{j!}\left(\frac{1}{m}\right)^{j-1} \leq n\left(\frac{n}{m}\right)^{j-1}\frac{1}{j!}.
\end{aligned}$$

Vysvětlení: První nerovnost plyne z toho, že pravděpodobnost disjunkce jevů je menší než součet pravděpodobností jevů, druhá nerovnost plyne z toho, že i-tý řetězec má délku alespoň j, jakmile existuje podmnožina $A \subseteq S$ taková, že $|A| = j$ (těchto možností je $\binom{n}{j}$) a pro každé $x \in A$ platí $h(x) = i$ (pravděpodobnost tohoto jevu je $\left(\frac{1}{m}\right)^j$).

Důsledek.

$$\text{Prob}\left(\max_i\left(\ell\left(i\right)\right)\geq j\right)\leq\min\left\{1,n\left(\frac{n}{m}\right)^{j-1}\frac{1}{j!}\right\}. \qquad \text{(\#2)}$$

Předpoklad: $\alpha=\frac{n}{m}\leq 1$. Ukážeme, že pro dostatečně velká n pro

$$j_0=\min\left\{j\mid n\left(\frac{n}{m}\right)^{j-1}\frac{1}{j!}\leq 1\right\}$$

platí $j_0\leq\frac{8\log n}{\log\log n}$. Z $\frac{n}{m}\leq 1$ a z $\left(\frac{j}{2}\right)^{\frac{j}{2}}\leq j!$ plyne

$$\min\left\{j\mid n\left(\frac{n}{m}\right)^{j-1}\frac{1}{j!}\leq 1\right\}\leq\min\left\{j\mid\frac{n}{j!}\leq 1\right\}\leq\min\left\{j\mid n\leq\left(\frac{j}{2}\right)^{\frac{j}{2}}\right\}$$

Výraz zanedbáváme, protože <= 1. (#3)

pro každé $n\geq 1$, kde j probíhá přirozená čísla. Pro pevné n označme

$$k+1=\min\left\{j\mid n\leq\left(\frac{j}{2}\right)^{\frac{j}{2}}\right\},$$

pak

$$\left(\frac{k}{2}\right)^{\frac{k}{2}}<n\leq\left(\frac{k+1}{2}\right)^{\frac{k+1}{2}}.$$

Nyní toto dvakrát zlogaritmujeme a protože logaritmus je rostoucí funkce, dostáváme

$$\left(\frac{k}{2}\right)\log\left(\frac{k}{2}\right)<\log n\leq\frac{k+1}{2}\log\left(\frac{k+1}{2}\right)$$

$$\log\left(\frac{k}{2}\right)+\log\log\left(\frac{k}{2}\right)<\log\log n\leq\log\left(\frac{k+1}{2}\right)+\log\log\left(\frac{k+1}{2}\right).$$

Jde odhadout shora výrazem 2 log((k+1)/2)

Za předpokladu, že $k\geq 3$, tak dostáváme, že $\log\log n<2\log\left(\frac{k+1}{2}\right)$, a odtud plyne

$$\frac{k}{8}<\frac{k}{4}\frac{\log\left(\frac{k}{2}\right)}{\log\left(\frac{k+1}{2}\right)}<\frac{\log n}{\log\log n},$$

1/2 < ^^

protože pro $k\geq 3$ je $\frac{1}{2}<\frac{\log\left(\frac{k}{2}\right)}{\log\left(\frac{k+1}{2}\right)}$. ~~Při sofistikovanější metodě, když se použije Stirlingova aproximace $\log j!$, lze dokázat, že $j_0<(1+a_j)\frac{\log n}{\log\log n}$, kde $\lim_{j\mapsto\infty}a_j=0$.~~

Toto použijeme při odhadu $E(NP)$. [Počítáme stále očekávanou délku nejdelšího řetězce]

$$E(NP) = \sum_j \text{Prob}\left(\max_i(\ell(i)) \geq j\right) \leq$$

$$\sum_j \min\left\{1, n\left(\frac{n}{m}\right)^{j-1}\frac{1}{j!}\right\} =$$

$$\sum_{j=1}^{j_0} 1 + \sum_{j=j_0+1}^{\infty}\left(n\left(\frac{n}{m}\right)^{j-1}\frac{1}{j!}\right) \leq j_0 + \sum_{j=j_0+1}^{\infty}\frac{n}{j!} =$$

$$j_0 + \frac{n}{j_0!}\sum_{j=j_0+1}^{\infty}\frac{j_0!}{j!} \leq j_0 + \sum_{j=j_0+1}\left(\frac{1}{j_0+1}\right)^{j-j_0} =$$

$$j_0 + \frac{\frac{1}{j_0+1}}{-\frac{1}{j_0+1}+1} = j_0 + \frac{1}{j_0} = O(j_0).$$

Vysvětlení: Při druhé nerovnosti jsme použili, že $\frac{n}{m} \leq 1$, při třetí nerovnosti jsme použili, že $\frac{n}{j_0!} \leq 1$ a

$$\frac{j_0!}{j!} = \frac{1}{\prod_{k=j_0+1}^{j} k} \leq \left(\frac{1}{j_0+1}\right)^{j-j_0}.$$

Shrneme získaný výsledek

Věta. *Za předpokladu $\alpha = \frac{n}{m} \leq 1$ je při hašování se separovanými řetězci horní odhad očekávané délky maximálního řetězce $O\left(\frac{\log n}{\log\log n}\right)$.*
Když $0.5 \leq \alpha \leq 1$, je to zároveň i dolní odhad. [Bez důkazu]

Očekávaný počet testů.

Def: Test je porovnání argumentu operace s prvkem na daném místě řetězce nebo zjištění, že vyšetřovaný řetězec je prázdný.

Budeme rozlišovat dva případy:
úspěšné vyhledávání – argument operace je mezi prvky reprezentované množiny,
neúspěšné vyhledávání – argument operace není mezi prvky reprezentované množiny.

Neúspěšné vyhledávání.

Očekávaný počet testů:

$$E(T) = \text{Prob}(\ell(i) = 0) + \sum_l l\,\text{Prob}(\ell(i) = l) =$$

$$p_{n,0} + \sum_l l p_{n,l} =$$

$$\left(1 - \frac{1}{m}\right)^n + \frac{n}{m} \approx e^{-\alpha} + \alpha.$$

Vysvětlení: Zjištění, zda řetězec je prázdný, vyžaduje jeden test, tj. $\text{Prob}(\ell(i)=0)$ není s koeficientem 0, ale 1. Protože pravděpodobnosti jsou stejné pro všechny řetězce, nemusíme specifikovat řetězec, který vyšetřujeme, stačí psát obecně i. $\sum_l lp_{n,l}$ jsme spočítali při výpočtu očekávané délky řetězce.

Úspěšné vyhledávání.

Zvolme jeden řetězec prvků o délce l. Počet testů při vyhledání všech prvků v tomto řetězci je

$$1+2+\cdots+l=\binom{l+1}{2}.$$

- Očekávaný počet testů při vyhledání všech prvků v nějakém řetězci je

$$\sum_l \binom{l+1}{2}\text{Prob}(\ell(i)=l)=\sum_l \binom{l+1}{2}p_{n,l}.$$

- Očekáný počet testů při vyhledání všech prvků v tabulce je $m\sum_l \binom{l+1}{2}p_{n,l}$.
- Očekávaný počet testů pro vyhledání jednoho prvku je

$$\begin{aligned}\frac{m}{n}\sum_{l=0}^{n}\binom{l+1}{2}p_{n,l} &= \frac{m}{2n}\left(\sum_{l=0}^{n}l^2p_{n,l}+\sum_{l=0}^{n}lp_{n,l}\right)=\\ &\frac{m}{2n}\left(\sum_{l=1}^{n}l(l-1)p_{n,l}+2\sum_{l=1}^{n}lp_{n,l}\right)=\\ &\frac{m}{2n}\left(\frac{n(n-1)}{m^2}+\frac{2n}{m}\right)=\frac{n-1}{2m}+1\approx\\ &1+\frac{\alpha}{2}.\end{aligned}$$

1

Jiný postup: Počet testů při úspěšném vyhledávání prvku $x\in S$ je 1+počet porovnání klíčů při neúspěšném vyhledávání x v operaci **INSERT**(x). Počet porovnání klíčů je délka řetězce, a proto očekávaný počet porovnání klíčů je očekávaná délka řetězce. Tedy očekávaný počet testů při úspěšném vyhledávání x je 1+očekávaná délka řetězce v okamžiku vkládání x, neboli

$$\frac{1}{n}\sum_{i=0}^{n-1}\left(1+\frac{i}{m}\right)=1+\frac{n-1}{2m}.$$

Věta. *Při hašování se separovanými řetězci je očekávaný počet testů při neúspešném vyhledávaní přibližně $e^{-\alpha}+\alpha$ a při úspěšném vyhledávání přibližně $1+\frac{\alpha}{2}$.*

Následující tabulka dává přehled očekávaného počtu testů pro různé hodnoty α

α	0	0.1	0.2	0.3	0.4	0.5	0.6
neúsp. vyh.	1	1.005	1.019	1.041	1.07	1.107	1.149
úspěš. vyh.	1	1.05	1.1	1.15	1.2	1.25	1.3

α	0.7	0.8	0.9	1	2	3
neúsp. vyh.	1.196	1.249	1.307	1.368	2.135	3.05
úspěš. vyh.	1.35	1.4	1.45	1.5	2	2.5

Všimněte si, že očekávaný počet testů při neúspěšném vyhledávání je menší než očekávaný počet testů při úspěšném vyhledávání, když $\alpha \leq 1$. Na první pohled vypadá tento výsledek nesmyslně, ale důvod je, že počet testů při úspěšném vyhledávání průměrujeme proti n, kdežto při neúspěšném vyhledávání proti m. Ilustrujeme to na následujícím příkladu:

Příklad: Nechť $n = \frac{m}{2}$ a nechť polovina neprázdných řetězců má délku 1 a polovina má délku 2.

Očekávaný počet testů při neúspěšném vyhledávaní:

1 test pro prázdné řetězce a řetězce délky 1 – těchto případů je $\frac{5m}{6}$
2 testy pro řetězce délky 2 – těchto případů je $\frac{m}{6}$.

Očekávaný počet testů je $\frac{1}{m}\left(1\frac{5m}{6} + 2\frac{m}{6}\right) = \frac{7}{6}$.

Očekávaný počet testů při úspěšném vyhledávání:

1 test pro prvky na prvním místě řetězce – těchto případů je $\frac{2n}{3}$
2 testy pro prvky, které jsou na druhém místě řetězce – těchto případů je $\frac{n}{3}$.

Očekávaný počet testů je $\frac{1}{n}\left(1\frac{2n}{3} + 2\frac{n}{3}\right) = \frac{4}{3}$.

Volba \alpha: Velikost α je doporučována menší než 1, ale nemá být hodně malá, protože by paměť nebyla efektivně využita.

Hašování s uspořádanými separovanými řetězci

Vylepšení metody: hašování s uspořádanými řetězci. Rozdíl proti původní metodě – řetězce jsou uspořádané ve vzrůstajícím pořadí. Protože řetězce obsahují tytéž prvky, je počet očekávaných testů při úspěšném vyhledávání stejný jako u neuspořádaných řetězců. Při neúspěšném vyhledávání končíme, když argument operace je menší než vyšetřovaný prvek v řetězci, tedy končíme dřív. Následující věta (bez důkazu) uvádí očekavaný počet testů v neúspěšném případě.

Věta. *Očekávaný počet testů při neúspěšném vyhledávání pro hašování s uspořádanými řetězci je přibližně* $e^{-\alpha} + 1 + \frac{\alpha}{2} - \frac{1}{\alpha}\left(1 - e^{-\alpha}\right)$. *Očekávaný počet testů při úspěšném vyhledávání pro hašování s uspořádanými řetězci je přibližně* $1 + \frac{\alpha}{2}$. [bez důkazu]

Uvedeme algoritmy pro operace s uspořádanými řetězci.

Algoritmy.

MEMBER(x):

Spočítáme $i := h(x)$, $t := NIL$
if i-tý seznam je neprázdný **then**
  t :=první prvek i-tého seznamu
  while $t < x$ a $t \neq$poslední prvek i-tého seznamu **do**
    t :=následující prvek i-tého seznamu
  enddo
endif
if $t = x$ **then** $x \in S$ **else** $x \notin S$ **endif**

INSERT(x):

Spočítáme $i := h(x)$, $t := NIL$
if i-tý seznam je neprázdný **then**
  t :=první prvek i-tého seznamu

while $t < x$ a $t \neq$poslední prvek i-tého seznamu **do**
$t :=$následující prvek i-tého seznamu
enddo
endif
if $t \neq x$ **then**
if $x < t$ **then** [Může se stát jen když t je poslední prvek]
vložíme x do i-tého seznamu před prvek t
else
vložíme x do i-tého seznamu za prvek t
endif
endif

DELETE(x):
Spočítáme $i := h(x)$, $t := NIL$ **if** i-tý seznam je neprázdný **then**
$t :=$první prvek i-tého seznamu
while $t < x$ a $t \neq$poslední prvek i-tého seznamu **do**
$t :=$následující prvek i-tého seznamu
enddo
endif
if $t = x$ **then** odstraníme x z i-tého seznamu **endif**

Nevýhody hašovaní se separovanými řetězci –
nevyužití alokované paměti (nehospodárné)
používání ukazatelů (cache).
Řešení: využít pro řetězce původní tabulku.

Položky tabulky:
key,
odkaz na uložená data,
položky pro práci s tabulkou.

Předpoklad o datech: Předpokládáme, že data jsou velká, v tom případě se ukládají mimo tabulku. V tabulce je jen odkaz na uložená data. Při popisu práce s tabulkou tuto část budeme vynechávat (tj. data budou pouze klíč).

Druhy hašování: Podle řešení kolize dělíme dál hašování:
- hašování s přemísťováním, hašování s dvěma ukazateli,
- srůstající hašování,
- dvojité hašování a hašování s lineárním přidáváním.

Hašování s přemísťováním

Položky pro práci s tabulkou: next, previous
- položka next – číslo řádku tabulky obsahující následující položku seznamu
- položka previous – číslo řádku tabulky obsahující předcházejí položku seznamu.

Příklad: $U = \{1, 2, \ldots, 1000\}$, $h(x) = x \bmod 10$,
uložená množina $S = \{1, 7, 11, 53, 73, 141, 161\}$,

řetězce: $P(1) = (1, 141, 11, 161)$, $P(3) = (73, 53)$, $P(7) = (7)$. [Jednotlivé seznamy]
Hašovací tabulka:

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

Tabulka vznikla následující posloupností operací:
INSERT(1), **INSERT**(141), **INSERT**(11), **INSERT**(73), **INSERT**(53), **INSERT**(7), **INSERT**(161).

Algoritmy.

MEMBER(x):

Spočítáme $i := h(x)$
if $i.previous \neq$prázdné nebo $i.key =$prázdné **then Výstup**: $x \notin S$, stop **endif**
while $i.next \neq$prázdné a $i.key \neq x$ **do** $i := i.next$ **enddo**
if $i.key = x$ **then Výstup**: $x \in S$ **else Výstup**: $x \notin S$ **endif**

DELETE(x):

Spočítáme $i := h(x)$
if $i.previous \neq$prázdné nebo $i.key =$prázdné **then** stop **endif** [Nejsem na začátku seznamu]
while $i.next \neq$prázdné a $i.key \neq x$ **do** $i := i.next$ **enddo** [Hledám prvek x]
if $i.key = x$ **then** [Nalezl jsem prvek x]
  if $i.previous \neq$prázdné **then**
    $(i.previous).next := i.next$ [Můj předchůdce musí ukazovat na mého následníka]
    if $i.next \neq$prázdné **then** $(i.next).previous := i.previous$ **endif** [Můj následník musí ukazovat na mého předchůdce]
    $i.key := i.next := i.previous :=$ prázdné [Prvek x po sobě smaže svůj řádek v tabulce]
  else [Předchůdce prvku x není, takže x je první prvek v seznamu]
    if $i.next \neq$prázdné **then**
      $i.key := (i.next).key$, $i.next := (i.next).next$ [Zkopíruju následníka na řádek, kde je nyní x]
      if $((i.next).next) \neq$prázdné **then** $((i.next).next).previous := i$ **endif** [Po přesunutí následníka na pozici x by mohl mít následník-následníka špatnou hodnotu *previous*]
      $(i.next).key := (i.next).next := (i.next).previous :=$ prázdné
    else
      $i.key :=$ prázdné [Prvek x nemá ani předchůdce ani následníka, mažu klíč]
    endif
  endif
endif

INSERT(x):

Spočítáme $i := h(x)$
if $i.previous \neq NIL$ nebo $i.key = NIL$ **then** [Prázdný seznam nebo neprázdný předek]
  if $i.key = NIL$ **then**
    $i.key := x$ [Do prázdného seznamu jen přidám klíč a hotovo]
  else [Buď mám již nějaký prvek v seznamu nebo hůř: do mého seznamu prorostl jiný seznam]
    if neexistuje prázdný řádek tabulky **then**
      Výstup: přeplnění, stop
    else
      nechť j je volný řádek tabulky
      $j.key := i.key$, $j.previous := i.previous$, $j.next := i.next$
      $(j.previous).next := j$
      if $j.next \neq NIL$ **then** $(j.next).previous := j$ **endif**
      $i.key := x$, $i.next := i.previous$:=prázdné
    endif
  endif
  stop
endif
while $i.next \neq NIL$ a $i.key \neq x$ **do** $i := i.next$ **enddo**
if $i.key \neq x$ **then**
  if neexistuje prázdný řádek tabulky **then**
    Výstup: přeplnění, stop
  else
    nechť j je volný řádek tabulky
    $i.next := j$, $j.key := x$, $j.previous := i$, ~~stop~~
  endif
endif
endif

První případ.

Všimněte si příkazu STOP, tedy toto je jedna větev algritmu. Z nějakého důvodu není použito ELSE, což by bylo přehlednější.

[Přemísťujeme prvek i, protože patří do jiného seznamu, musíme tedy udělat místo pro prvek našeho nového jednoprvkového seznamu.]

[Dojedu si na konec seznamu /nebo najdu vkládaný prvek a končím.]

V příkladu provedeme **INSERT**(28), nový řádek je 4. řádek – výsledná hašovací tabulka

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	1

Očekávaný počet testů:

Očekávaný počet testů je stejný jako pro hašování se separovanými řetězci:
- úspěšné vyhledávání: $\frac{n-1}{2m} + 1 \approx 1 + \frac{\alpha}{2}$
- neúspěšné vyhledávání: $\left(1 - \frac{1}{m}\right)^n + \frac{n}{m} \approx e^{-\alpha} + \alpha$,

kde m = velikost tabulky, n = velikost S tj. počet uložených prvků, $\alpha = \frac{n}{m}$ = faktor zaplnění.

Hašování s dvěma ukazateli

Nevýhoda hašování s přemísť. je přemístování:

Nevýhoda hašování s přemísťováním je krok 5) v operaci **INSERT**. Vyžaduje více času – operace s přemístěním položky. Toto odstraňuje další implementace hašování se separujícími řetězci.

:-)

Položky pro práci s tabulkou – next, begin

Položka next – číslo řádku tabulky obsahující následující položku seznamu

Položka begin – číslo řádku tabulky obsahující první položku seznamu s touto adresou

Pozn: Stejná data jako v minulém případě

Hašovací tabulka:

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

Tabulka vznikla následující posloupností operací:
INSERT(1), **INSERT**(141), **INSERT**(11), **INSERT**(73), **INSERT**(53), **INSERT**(7), **INSERT**(161).

Algoritmy.

MEMBER(x):

Spočítáme $i := h(x)$
if $i.begin =$ prázdné **then**
  Výstup: $x \notin S$, stop
else
  $i := i.begin$
endif
while $i.next \neq$ prázdné a $i.key \neq x$ **do**
  $i := i.next$
enddo
if $i.key = x$ **then**

Výstup: $x \in S$
else
Výstup: $x \notin S$
endif

DELETE(x):

Spočítáme $i := h(x)$
if $i.begin$ =prázdné **then** stop **else** $j := i$, $i := i.begin$ **endif** [j je jakýsi předchozí prvek pro i]
while $i.next \neq$prázdné a $i.key \neq x$ **do** $j := i$, $i := i.next$ **enddo** [Hledám klíč x]
if $i.key = x$ **then**
  if $i = j.begin$ **then**
    if $i.next \neq$prázdné **then**
      $j.begin := i.next$
    else
      $j.begin :=$prázdné
    endif
  else
    $j.next := i.next$
  endif
  $i.key := i.next :=$prázdné [Promažeme řádku i]
endif

[Chceme správně nastavit hodnotu *begin*, pokud by byla smazáním prvku *i* ovlivněna]

INSERT(x):

Spočítáme $i := h(x)$
if $i.begin$ =prázdné **then**
  if $i.key$ =prázdné **then**
    $i.key := x$, $i.begin := i$
  else
    if neexistuje prázdný řádek tabulky **then**
      Výstup: přeplnění, stop
    else
      nechť j je volný řádek tabulky
      $j.key = x$, $i.begin := j$
    endif
  endif
else
  $i := i.begin$
  while $i.next \neq$prázdné a $i.key \neq x$ **do** $i := i.next$ **enddo**
  if $i.key \neq x$ **then**
    if neexistuje prázdný řádek tabulky **then**
      Výstup: přeplnění, stop
    else
      nechť j je volný řádek tabulky, $i.next := j$, $j.key := x$, ~~stop~~
    endif
  endif
endif

Přidáváme do prázdného seznamu.

Příklad: V příkladu provedeme **INSERT**(28), nový řádek je 4. řádek – výsledná hašovací tabulka

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

Algoritmus při práci s položkami je rychlejší než při hašování s přemísťováním, ale začátek řetězce v jiném místě tabulky přidává jeden test. Výsledek bez odvozování:

Očekávaný počet testů:

úspěšný případ: $1+\frac{(n-1)(n-2)}{6m^2}+\frac{n-1}{2m}\approx 1+\frac{\alpha^2}{6}+\frac{\alpha}{2}$

neúspěšný případ: $\approx 1+\frac{\alpha^2}{2}+\alpha+e^{-\alpha}(2+\alpha)-2.$

Srůstající hašování

Terminologie: Srůstající hašování se dělí podle práce s pamětí na standardní a na srůstající hašování s pomocnou pamětí (které se nazýva jen srůstající hašování) a podle způsobu přidávání dalšího prvku.

Popíšeme metody:

Standardní srůstající hašování: LISCH, EISCH,

Srůstající hašování: LICH, VICH, EICH.

Všechny metody pro práci s tabulkou používají jen položku next – číslo řádku tabulky obsahující následující položku seznamu.

Základní idea: řetězec začíná na svém místě, ale pokud už tam byl uložen nějaký údaj, pak řetězec tohoto údaje sroste s řetězcem začínajícím na tomto řádku. To znamená, že prvky řetězce, který začíná na tomto místě budou uloženy v řetězci, který už je uložen na tomto místě, ale jen od tohoto místa dál.

Metody EISCH a LISCH.

EISCH – early-insertion standard coalesced hashing

LISCH – late-insertion standard coalesced hashing.

Organizace tabulky je stejná jako v předchozích případech.

Základní ideje: LISCH přidává nový prvek na konec řetězce,

EISCH přidává nový prvek x do řetězce za řádkem $h(x)$.

Příklad:

Příklad: $U = \{1, 2, \ldots, 1000\}$, $h(x) = x \bmod 10$
množina $S = \{1, 7, 11, 53, 73, 141, 171\}$ je uložena v hašovací tabulce

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

Tabulka pro metodu LISCH vznikla následující posloupností operací:
INSERT(1), **INSERT**(141), **INSERT**(11), **INSERT**(73), **INSERT**(53), **INSERT**(161), **INSERT**(7).
Pro metodu EISCH tabulka vznikla následující posloupností operací:
INSERT(1), **INSERT**(161), **INSERT**(11), **INSERT**(73), **INSERT**(53), **INSERT**(7), **INSERT**(141).
Provedeme **INSERT**(28), přidáváme do čvrtého rádku, výsledná tabulka vlevo je pro metodu LISCH, vpravo pro metodu EISCH.

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

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

Algoritmy.

Algoritmus operace **MEMBER** je pro obě metody stejný.

MEMBER(x):
Spočítáme $i := h(x)$
while $i.next \neq$prázdné a $i.key \neq x$ **do** $i := i.next$ **enddo**
if $i.key = x$ **then Výstup**: $x \in S$ **else Výstup**: $x \notin S$ **endif**

Metoda LISCH – **INSERT**(x):

Spočítáme $i := h(x)$
while $i.next \neq$prázdné a $i.key \neq x$ **do** $i := i.next$ **enddo** [Dojedu na konec seznamu]
if $i.key \neq x$ **then**
  if neexistuje prázdný řádek tabulky **then**
    Výstup: přeplnění, stop
  else
    nechť j je prázdný řádek, $j.key := x$, $i.next := j$ [Přidám prvek na konec seznamu]
  endif
endif

Metoda EISCH – **INSERT**(x):

Spočítáme $k := i := h(x)$
while $i.next \neq$prázdné a $i.key \neq x$ **do** $i := i.next$ **enddo** [Dojedu na konec seznamu]
if $i.key \neq x$ **then**
  if neexistuje prázdný řádek tabulky **then**
    Výstup: přeplnění, stop
  else
    nechť j je volný řádek tabulky
    $j.next := k.next$, $k.next := j$, $j.key := x$ [Posledním prvkem je nyní druhý prvek a druhým prvkem je vkládaný prvek x]
  endif
endif

Operace DELETE:

Efektivní operace **DELETE** není známá, ale i primitivní algoritmy pro operaci **DELETE** mají rozumnou očekávanou časovou složitost.

Analýza složitosti těchto algoritmů..

Popis situace: Uložena množina $S = \{s_1, s_2, \ldots, s_n\}$ do tabulky velikosti m, je dán prvek s_{n+1} a máme zjistit, zda $s_{n+1} \in S$. Označme $a_i = h(s_i)$ pro $i = 1, 2, \ldots, n+1$, kde h je použitá hašovací funkce.
Předpoklad: všechny posloupnosti $a_1, a_2, \ldots, a_{n+1}$ jsou **stejně pravděpodobné**. Výběr prázdného řádku je pevně daný, to znamená, že při stejně obsazených řadcích dostaneme vždy stejný prázdný řádek.

Neúspěšné vyhledávání ($s_{n+1} \notin S$)..

Označení: $C(a_1, a_2, \ldots, a_n; a_{n+1})$ označuje počet testů pro zjištění, že $s_{n+1} \notin S$. Platí: očekávaný počet testů při neúspěšném vyhledávání v množině S je

$$\frac{\sum C(a_1, a_2, \ldots, a_n; a_{n+1})}{m^{n+1}},$$

kde se sčítá přes všechny posloupnosti $a_1, a_2, \ldots, a_{n+1}$ – a těch je m^{n+1}.

Def:

Řetězec délky l v množině S je maximální posloupnost adres $(b_1, b_2, \ldots, b_l)$ taková, že $b_i.next = b_{i+1}$ pro $i = 1, 2, \ldots, l-1$. Když adresa a_{n+1} je i-tý prvek v řetězci, pak počet testů je $l - i + 1$. Proto řetězec délky l přispěl k součtu $\sum C(a_1, a_2, \ldots, a_n; a_{n+1})$ celkem počtem testů $1 + 2 + \cdots + l = l + \binom{l}{2}$. [Součet aritmetické posloupnosti: L/2 (1+L) = (L over 2) + L]

Pozn: l = L, jen rozdíl mezi l a 1 není vidět, tak používám L.

Def:

$c_n(l)$ = počet všech řetězců délky l ve všech reprezentacích n-prvkových množin (ztotožňujeme dvě množiny, které měly stejnou posloupnost adres při ukládaní prvků), pak

Výpočet pro počet testů v neúsp. př.

$$\sum C(a_1, a_2, \ldots, a_n; a_{n+1}) = c_n(0) + \sum_{l=1}^{n} \left(l + \binom{l}{2}\right) c_n(l) \qquad \text{[Rozdělím na dvě sumy]}$$

$$= c_n(0) + \sum_{l=1}^{n} l c_n(l) + \sum_{l=1}^{n} \binom{l}{2} c_n(l),$$

Počet testů pro řetězec délky l.

kde $c_n(0)$ je počet prázdných řádků ve všech reprezentacích.

Reprezentace S má $m-n$ prázdných řádků, [Nemůžu mít n > m prvků, protože by se mi do tabulky nevešly] všech posloupností n-adres je m^n, proto

$$c_n(0) = (m-n)\, m^n.$$

[m^n je počet možností, jak se mohou prvky S zahašovat do tabulky, každá taková tabulka má m-n prázdných řádků; to že se prázdné řádky opakují nám nevadí, protože počítáme stále součet (SUMU) a ne průměr.]

$\sum_{l=1}^{n} l c_n(l)$ je celková délka všech řetězců ve všech tabulkách reprezentujících všechny n-prvkové množiny a proto

$$\sum_{l=1}^{n} l c_n(l) = n m^n.$$

Spočítáme $S_n = \sum_{l=1}^{n} \binom{l}{2} c_n(l)$. Nejprve rekurentní vztah pro $c_n(l)$. Přidáváme prvek s adresou a_{n+1}. Pak řetězec délky l v reprezentaci S zůstal stejný, když adresa a_{n+1} neležela v tomto řetězci, v opačném případě se délka řetězce zvětšila na $l+1$. Proto přidání jednoho prvku vytvořilo z řetězce délky l celkem $m-l$ řetězců délky l a l řetězců délky $l+1$. Vysčítáním přes všechny n-prvkové posloupnosti adres dostáváme

$$c_{n+1}(l) = (m-l)\, c_n(l) + (l-1)\, c_n(l-1).$$

Odtud

$$S_n = \sum_{l=1}^{n} \binom{l}{2} c_n(l) =$$

$$\sum_{l=1}^{n} \left(\binom{l}{2} (m-l)\, c_{n-1}(l) + \binom{l}{2} (l-1)\, c_{n-1}(l-1) \right) = \qquad \text{[Použili jsme rekurentní vztah]}$$

$$\left(\sum_{l=1}^{n} \binom{l}{2} (m-l)\, c_{n-1}(l) \right) + \left(\sum_{l=0}^{n-1} \binom{l+1}{2} l c_{n-1}(l) \right) = \qquad \text{[Rozdělili jsme na dvě sumy]}$$

Poslední člen první sumy

$$\binom{n}{2} (m-n) \underbrace{c_{n-1}(n)}_{0} +$$

první člen druhé sumy

$$\left(\sum_{l=1}^{n-1} \left(\binom{l}{2} (m-l) + \binom{l+1}{2} l \right) c_{n-1}(l) \right) + \binom{1}{2} 0 c_{n-1}(0) =$$

$$\sum_{l=1}^{n-1} \binom{l}{2} (m+2)\, c_{n-1}(l) + \sum_{l=1}^{n-1} l c_{n-1}(l) =$$

$$(m+2)\, S_{n-1} + (n-1)\, m^{n-1},$$

kde jsme použili, že $c_{n-1}(n) = 0$, a identitu

$$(m-l)\binom{l}{2} + l\binom{l+1}{2} = \frac{1}{2}\left(l^2m - lm - l^3 + l^2 + l^3 + l^2\right) =$$
$$\frac{1}{2}\left(l^2m - lm + 2l^2\right) =$$
$$\frac{1}{2}\left(l^2m - lm + 2\left(l^2 - l\right)\right) + l =$$
$$(m+2)\binom{l}{2} + l.$$

Rekurence pro S_n dává

$$S_n = (m+2)S_{n-1} + (n-1)m^{n-1} =$$
$$(m+2)^2 S_{n-2} + (m+2)(n-2)m^{n-2} + (n-1)m^{n-1} =$$
$$(m+2)^3 S_{n-3} + (m+2)^2(n-3)m^{n-3} +$$
$$(m+2)(n-2)m^{n-2} + (n-1)m^{n-1} =$$
$$0 \quad (m+2)^{n-1}S_0 + \sum_{i=0}^{n-1}(m+2)^i(n-1-i)m^{n-1-i} =$$
$$(m+2)^{n-1}\sum_{i=0}^{n-1}(n-1-i)\left(\frac{m}{m+2}\right)^{n-1-i} =$$
$$(m+2)^{n-1}\sum_{i=1}^{n-1} i\left(\frac{m}{m+2}\right)^{i},$$

[Je jedno zda suma pojede od "začátku" či "od konce".]

kde jsme využili, že $S_0 \neq 0$. Spočítáme součet $T_c^n = \sum_{i=1}^n ic^i$ pro $n = 1, 2, \ldots$ a $c \neq 1$. Z $cT_c^n = \sum_{i=1}^n ic^{i+1}$ plyne

$$(c-1)T_c^n = cT_c^n - T_c^n = \sum_{i=2}^{n+1}(i-1)c^i - \sum_{i=1}^{n} ic^i =$$
$$nc^{n+1} + \left(\sum_{i=2}^{n}\left((i-1)c^i - ic^i\right)\right) - c =$$
$$nc^{n+1} + \left(\sum_{i=2}^{n} -c^i\right) - c =$$
$$nc^{n+1} - \sum_{i=1}^{n} c^i = nc^{n+1} - \frac{c^{n+1} - c}{c-1} =$$
$$\frac{nc^{n+2} - (n+1)c^{n+1} + c}{c-1}.$$

Sečetl jsem pár členů geom. řady

Tedy platí

$$T_c^n = \frac{nc^{n+2} - (n+1)\,c^{n+1} + c}{(c-1)^2}.$$

Protože $\frac{m}{m+2} \neq 1$, dostáváme, že

$$S_n = (m+2)^{n-1} \frac{(n-1)\left(\frac{m}{m+2}\right)^{n+1} - n\left(\frac{m}{m+2}\right)^n + \frac{m}{m+2}}{\left(\frac{m}{m+2} - 1\right)^2} =$$

[Zůstalo stejné]

$$\frac{1}{4}(m+2)^{n+1}\left[(n-1)\left(\frac{m}{m+2}\right)^{n+1} - n\left(\frac{m}{m+2}\right)^n + \frac{m}{m+2}\right] =$$

$$\frac{1}{4}\left[(n-1)\,m^{n+1} - n\,(m+2)\,m^n + m\,(m+2)^n\right] =$$

$$\frac{1}{4}\left(m\,(m+2)^n - m^{n+1} - 2nm^n\right).$$

Výsledný odhad počtu testů při neúspěšném vyhledávání:

Očekávaný počet testů při neúspěšném vyhledávání je

$$\frac{(m-n)\,m^n + nm^n + \frac{1}{4}\left(m\,(m+2)^n - m^{n+1} - 2nm^n\right)}{m^{n+1}} =$$

$$\frac{m^{n+1} + \frac{1}{4}\left(m\,(m+2)^n - m^{n+1} - 2nm^n\right)}{m^{n+1}} =$$

$$1 + \frac{1}{4}\left(\left(1 + \frac{2}{m}\right)^n - 1 - \frac{2n}{m}\right) \sim 1 + \frac{1}{4}\left(e^{2\alpha} - 1 - 2\alpha\right).$$

Tento odhad je stejný pro obě metody – LISCH i EISCH, protože mají stejné posloupnosti adres (liší se jen pořadím prvků v jednotlivých řetězcích).

Úspěšný případ ($s_{n+1} \in S$).

Očekávaný počet testů při úspěšném vyhledávání v modelu LISCH spočítáme stejnou metodou jako pro hašování se separujícími řetězci. Pro vyhledání prvku $s_{n+1} \in S$ je počet testů roven 1+počet porovnání klíčů při operaci **INSERT**(s_{n+1})**.** Když s_{n+1} je vložen na místo $h(s_{n+1})$, nebyl porovnáván žádný klíč a test bude 1, když $h(s_{n+1})$ byl na na i-tém místě v řetězci délky l, pak bylo při operaci **INSERT**(s_{n+1}) použito $l - i + 1$ porovnání klíčů a teď se použije $l - i + 2$ testů. Podle předchozí části analýzy dostaneme, že očekávaný počet porovnání klíčů při neúspěšném vyhledávání je

$$\frac{1}{m^{n+1}}\left(\sum_{l=1}^{n}\left(l + \binom{l}{2}\right)c_n(l)\right) =$$

$$\frac{1}{m^{n+1}}\left(nm^n + \frac{1}{4}\left(m\,(m+2)^n - m^{n+1} - 2nm^n\right)\right) =$$

$$\frac{1}{4}\left(\left(1 + \frac{2}{m}\right)^n - 1 + \frac{2n}{m}\right).$$

Očekávaný počet testů v úspěšném případě pro metodu LISCH:

Tedy očekávaný počet testů při úspěšném vyhledávání v n-prvkové množině je podle předchozí analýzy roven $1+n$-tina součtu očekávaného počtu porovnání klíčů při neúspěšném vyhledávání v i-prvkové množině, kde i probíhá čísla $0, 1, \ldots, n-1$. Podle předchozích výsledků je hledaný součet

$$\sum_{i=0}^{n-1} \frac{1}{4}\left[\left(1+\frac{2}{m}\right)^{i} - 1 + \frac{2i}{m}\right] = \frac{1}{4}\frac{\left(1+\frac{2}{m}\right)^{n}-1}{1+\frac{2}{m}-1} - \frac{n}{4} + \frac{\binom{n}{2}}{2m} = \frac{m}{8}\left(\left(1+\frac{2}{m}\right)^{n} - 1 - \frac{2n}{m}\right) + \frac{n^2-n}{4m}.$$

Tedy očekávaný počet testů v úspěšném případě pro n-prvkovou množinu je

$$1 + \frac{m}{8n}\left(\left(1+\frac{2}{m}\right)^{n} - 1 - \frac{2n}{m}\right) + \frac{n-1}{4m} \sim 1 + \frac{1}{8\alpha}\left(e^{2\alpha} - 1 - 2\alpha\right) + \frac{\alpha}{4}.$$

Dtto pro metodu EISCH:

Pro metodu EISCH je očekáváný počet testů v úspěšném případě

$$\frac{m}{n}\left(\left(1+\frac{1}{m}\right)^{n} - 1\right) \sim \frac{1}{\alpha}\left(e^{\alpha} - 1\right).$$

Výpočet je ale komplikovanější musí se použít složitější metoda (metoda EISCH dává nový prvek hned za místo, kde má být uložen). Chyba aproximace pro tyto odhady je $O\left(\frac{1}{m}\right)$.

Metody LICH, EICH, VICH.

LICH – late-insertion coalesced hashing
EICH – early-insertion coalesced hashing
VICH – varied-insertion coalesced hashing.

Základní idea: Metody používají pomocnou paměť. Tabulka je rozdělená na adresovací část a na pomocnou paměť, která není dostupná pomocí hašovací funkce, ale pomáhá při řešení kolizí. Metody se liší operací **INSERT**. Všechny metody při kolizi nejprve použijí řádek tabulky z pomocné části a teprve, když je pomocná část zaplněna, používají adresovací část.
Metoda LICH: při **INSERT**u vkládá prvek vždy na konec řetězce.
Metoda EICH: při **INSERT**u vkládá prvek x do řetězce vždy na místo hned za řádkem $h(x)$.
Metoda VICH: Při **INSERT**u, když nový řádek je z pomocné části, tak je vložen s novým prvkem na konec řetězce, když je pomocná část paměti vyčerpána, tak se řádek s novým prvkem vkládá do řetězce za poslední řádek z pomocné části tabulky. Když řetězec neobsahuje žádný řádek z pomocné paměti, tak se řádek s novým prvkem x vkládá hned za řádek $h(x)$.

Idea: pomocná část má zabránit rychlému srůstání řetězců.

DELETE:

Tyto metody nepodporují přirozené efektivní algoritmy pro operaci **DELETE**.

Příklad: $U = \{1, 2, \ldots, 1000\}$, $h(x) = x \bmod 10$,
$S = \{1, 7, 11, 53, 73, 141, 161\}$. Tabulka má 12 řádků a má tvar

řádek	key	next
P(0)		
P(1)	1	10
P(2)		
P(3)	73	11
P(4)		
P(5)	7	
P(6)		
P(7)	161	5
P(8)	11	7
P(9)		
P(10)	141	8
P(11)	53	

Nepřístupná oblast pro hashovací fci: x mod 10

Hašovací tabulka vznikla posloupnostmi operací:
Pro metodu LICH:
INSERT(1), **INSERT**(73), **INSERT**(141), **INSERT**(53), **INSERT**(11), **INSERT**(161), **INSERT**(7).
Pro metodu EICH:
INSERT(1), **INSERT**(73), **INSERT**(161), **INSERT**(53), **INSERT**(11), **INSERT**(141), **INSERT**(7),
ale nedodržovalo se, že se nejdřív zaplňují řádky z pomocné části. Při dodržování tohoto pravidla takováto tabulka nemůže vzniknout.
Pro metodu VICH:
INSERT(1), **INSERT**(73), **INSERT**(141), **INSERT**(53), **INSERT**(161), **INSERT**(11), **INSERT**(7).

Aplikujeme operace **INSERT**(28) a **INSERT**(31), nové řádky budou řádky číslo 4 a 9. Tabulka vytvořená pomocí metody LICH je na levé straně, metodou VICH je v prostředku a metodou EICH je na pravé straně.

LICH

řádek	key	next
P(0)		
P(1)	1	10
P(2)		
P(3)	73	11
P(4)	28	9
P(5)	7	4
P(6)		
P(7)	161	5
P(8)	11	7
P(9)	31	
P(10)	141	8
P(11)	53	

VICH

řádek	key	next
P(0)		
P(1)	1	10
P(2)		
P(3)	73	11
P(4)	28	7
P(5)	7	
P(6)		
P(7)	161	5
P(8)	11	4
P(9)	31	8
P(10)	141	9
P(11)	53	

EICH

řádek	key	next
P(0)		
P(1)	1	9
P(2)		
P(3)	73	11
P(4)	28	7
P(5)	7	
P(6)		
P(7)	161	5
P(8)	11	4
P(9)	31	10
P(10)	141	8
P(11)	53	

Algoritmy.

Algoritmus operace **MEMBER** je pro tyto metody stejný jako pro LISCH a EISCH

MEMBER(x):
Spočítáme $i := h(x)$
while $i.next \neq$prázdné a $i.key \neq x$ **do** $i := i.next$ **enddo**
if $i.key = x$ **then Výstup**: $x \in S$ **else Výstup**: $x \notin S$ **endif**

Algoritmus operace **INSERT** je pro metodu LICH stejný jako pro metodu LISCH a pro metodu EICH je stejný jako pro metodu EISCH s jediným doplňkem, pokud existuje prázdný řádek v pomocné části, tak j-tý řádek je z pomocné části. Tento předpoklad je i pro algoritmus **INSERT** pro metodu VICH.

Metoda LICH – **INSERT**(x):
Spočítáme $i := h(x)$
if $i.next = NIL$ **then** $i.next = x$, stop **endif**
while $i.next \neq NIL$ a $i.key \neq x$ **do** $i := i.next$ **enddo**
if $i.key \neq x$ **then**
  if neexistuje prázdný řádek tabulky **then**
    přeplnění, stop
  else
    nechť j je prázdný řádek, $j.key := x$, $i.next := j$
  endif
endif

Metoda EICH – **Insert**(x):
Spočítáme $k := i := h(x)$
if $i.next = NIL$ **then** $i.next = x$, stop **endif**
while $i.next \neq NIL$ a $i.key \neq x$ **do** $i := i.next$ **enddo**
if $i.key \neq x$ **then**
  if neexistuje prázdný řádek tabulky **then**

přeplnění, stop
else
nechť j je volný řádek tabulky
$j.next := k.next$, $k.next := j$, $j.key := x$
endif
endif

Metoda VICH – **INSERT**(x):

Spočítáme $i := h(x)$
if $i.next = NIL$ **then** $i.next = x$, stop **endif**
while $i.next \neq NIL$ a $i.key \neq x$ **do**
if k není definováno a $i.next < m$ **then** $k := i$ **endif** [Zapamatuju si prvek, který není v pomocné části.]
Poznámka: Podmínka pro k je splněna, když jsme byli na začátku nebo v pomocné části, podmínka na $i.next$ je splněna, když $i.next$ není v pomocné části.
$i := i.next$
enddo
if $i.key \neq x$ **then**
if neexistuje prázdný řádek **then**
přeplnění, stop
else
nechť j je volný řádek, $j.key := x$
if k není definováno **then** [Když poslední prvek seznamu je v pomocné části]
$i.next := j$
else
$j.next := k.next$, $k.next := j$
endif
endif
endif

Složitost algoritmů pro srůstající hašování.
Značení: n – velikost uložené množiny,
m – velikost adresovací části tabulky,
m' – velikost tabulky,
$\alpha = \frac{n}{m'}$ – faktor zaplnění,
$\beta = \frac{m}{m'}$ – adresovací faktor,
λ – jediné nezáporné řešení rovnice $e^{-\lambda} + \lambda = \frac{1}{\beta}$.

Očekávaný počet testů pro metodu LICH
neúspěšný případ:
$e^{-\frac{\alpha}{\beta}} + \frac{\alpha}{\beta}$, když $\alpha \leq \lambda\beta$,
$\frac{1}{\beta} + \frac{1}{4}\left(e^{2\left(\frac{\alpha}{\beta}-\lambda\right)} - 1\right)\left(3 - \frac{2}{\beta} + 2\lambda\right) - \frac{1}{2}\left(\frac{\alpha}{\beta} - \lambda\right)$, když $\alpha \geq \lambda\beta$
úspěšný případ:
$1 + \frac{\alpha}{2\beta}$, když $\alpha \leq \lambda\beta$,
$1 + \frac{\beta}{8\alpha}\left(e^{2\left(\frac{\alpha}{\beta}-\lambda\right)} - 1 - 2\left(\frac{\alpha}{\beta} - \lambda\right)\right)\left(3 - \frac{2}{\beta} + 2\lambda\right) + \frac{1}{4}\left(\frac{\alpha}{\beta} + \lambda\right) + \frac{\lambda}{4}\left(1 - \frac{\lambda\beta}{\alpha}\right)$, když $\alpha \geq \lambda\beta$.

Očekávaný počet testů pro metodu EICH
neúspěšný případ:
$e^{-\frac{\alpha}{\beta}} + \frac{\alpha}{\beta}$, když $\alpha \leq \lambda\beta$,
$e^{2\left(\frac{\alpha}{\beta}-\lambda\right)}\left(\frac{3}{4}+\frac{\lambda}{2}-\frac{1}{2\beta}\right)+e^{\frac{\alpha}{\beta}-\lambda}\left(\frac{1}{\beta}-1\right)+\left(\frac{1}{4}-\frac{\alpha}{2\beta}+\frac{1}{2\beta}\right)$, když $\alpha \geq \lambda\beta$
úspěšný případ:
$1+\frac{\alpha}{2\beta}$, když $\alpha \leq \lambda\beta$,
$1+\frac{\alpha}{2\beta}+\frac{\beta}{\alpha}\left(\left(e^{\frac{\alpha}{\beta}-\lambda}-1\right)(1+\lambda)-\left(\frac{\alpha}{\beta}-\lambda\right)\right)\left(1+\frac{\lambda}{2}+\frac{\alpha}{2\beta}\right)$), když $\alpha \geq \lambda\beta$.

Očekávaný počet testů pro metodu VICH
neúspěšný případ:
$e^{-\frac{\alpha}{\beta}} + \frac{\alpha}{\beta}$, když $\alpha \leq \lambda\beta$,
$\frac{1}{\beta}+\frac{1}{4}\left(e^{2\left(\frac{\alpha}{\beta}-\lambda\right)}-1\right)\left(3-\frac{2}{\beta}+2\lambda\right)-\frac{1}{2}\left(\frac{\alpha}{\beta}-\lambda\right)$, když $\alpha \geq \lambda\beta$
úspěšný případ:
$1+\frac{\alpha}{2\beta}$, když $\alpha \leq \lambda\beta$,
$1+\frac{\alpha}{2\beta}+\frac{\beta}{\alpha}\left(\left(e^{\frac{\alpha}{\beta}-\lambda}-1\right)(1+\lambda)-\left(\frac{\alpha}{\beta}-\lambda\right)\right)\left(1+\frac{\lambda}{2}+\frac{\alpha}{2\beta}\right)+\frac{1-\beta}{\alpha}\left(\frac{\alpha}{\beta}-\lambda-e^{\frac{\alpha}{\beta}-\lambda}+1\right)$, když $\alpha \geq \lambda\beta$.

Chyba aproximace pro tyto odhady je $O\left(\log \frac{m'}{\sqrt{m'}}\right)$.

Hašování s lineárním přidáváním

Tabulka má jedinou položku – key

Základní idea: Při operaci **INSERT**(x) vložíme x na řádek $h(x)$, když je prázdný, v opačném případě nalezneme nejmenší i takové, že řádek $h(x)+i \bmod m$ je prázdný, a tam vložíme x. Tato metoda byla motivována snahou o co největší využití paměti.

Komentář: Metoda vyžaduje minimální velikost paměti. V tabulce se vytvářejí shluky použitých řádků, a proto při velkém zaplnění metoda vyžaduje velké množství času. Metoda nepodporuje efektivní implementaci operace **DELETE**. Při vyhledávání je třeba testovat, zda nevyšetřujeme podruhé první vyšetřovaný řádek a pro zjištění přeplnění je vhodné mít uložen počet vyplněných řádků v tabulce. Pro standarní paměti není výhodná. Při použiti cache-paměti se výrazně mění její ohodnocení, protože minimalizuje počet přechodů mezi různými typy pamětí. Proto se tato metoda doporučuje pro počítače s cache-pamětí.

MEMBER(x):
Spočítáme $i := h(x)$, $h := i$
if $i.key = x$ **then Výstup** $x \in S$, stop **endif**
if $i.key =$prázdný **then** Výstup: $x \notin S$, stop **endif**
$i := i+1$
while $i.key \neq$prázdný a $i.key \neq x$ a $i \neq h$ **do** $i := i+1 \bmod m$ **enddo**
if $i.key = x$ **then Výstup**: $x \in S$ **else Výstup**: $x \notin S$ **endif**

INSERT(x):
Spočítáme $i := h(x)$, $j := 0$

while $i.key \neq$prázdný a $i.key \neq x$ a $j < m$ **do** $i := i + 1 \bmod m$, $j := j + 1$ **enddo**
if $j = m$ **then Výstup**: přeplnění, stop **endif**
if $i.key =$prázdný **then** $i.key := x$ **endif**

Příklad: Máme universum $U = \{1, 2, \ldots, 1000\}$, hašovací funkci $h(x) = x \bmod 10$ a množinu $S = \{1, 7, 11, 53, 73, 141, 161\}$. Tato množina je uložena v levé tabulce. Provedeme operaci **INSERT**(35). Výsledek je uložen v pravé tabulce.

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

Insert 35

řádek	key
P(0)	
P(1)	1
P(2)	11
P(3)	73
P(4)	141
P(5)	161
P(6)	53
P(7)	7
P(8)	35
P(9)	

Tabulka vznikla posloupností operací:
INSERT(1), **INSERT**(11), **INSERT**(73), **INSERT**(141), **INSERT**(161), **INSERT**(53), **INSERT**(7).
Na závěr uvedeme složitost této metody. Očekávaný počet testů:

neúspěšný případ: $\approx \frac{1}{2}\left(1 + \left(\frac{1}{1-\alpha}\right)^2\right)$,

úspěšný případ: $\approx \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right)$.

Dvojité hašování

Nevýhoda lin. haš.: Základní nevýhoda předchozí metody je způsob výběru dalšího řádku. Je velmi determinován a důsledkem je vznik shluku řádků, který vede k výraznému zpomalení metody.
Idea jak odstranit tuto nevýhodu: Použijeme dvě hašovací funkce h_1 a h_2 a při operaci **INSERT**(x) nalezneme nejmenší $i = 0, 1, \ldots$ takové, že řádek $(h_1(x) + ih_2(x)) \bmod m$ je prázdný, a tam uložíme prvek x.

Tabulka: Tabulka má jedinou položku – key.
Požadavky na korektnost: Pro každé x musí být $h_2(x)$ a m nesoudělné (jinak prvek x nemůže být uložen na libovolném řádku tabulky).
Předpoklad pro výpočet očekavaného počtu testů: posloupnost $\{h_1(x) + ih_2(x)\}_{i=0}^{m-1}$ je náhodná permutace množiny řádků tabulky.
Nevýhoda: Uvedená metoda nepodporuje operaci **DELETE**.

Pozn: Přeplnění se řeší stejným způsobem jako v metodě hašování s lineárním přidáváním.
Poznámka: Metoda hašování s lineárním přidáváním je speciální případ dvojitého hašování, kde $h_2(x) = 1$ pro každé $x \in U$.

Algoritmy.

MEMBER(x):

Spočítáme $i := h_1(x)$, $h := h_2(x)$, $j := 0$
while $i.key \neq$prázdný a $i.key \neq x$ a $j < m$ **do** $i := i + h \bmod m$, $j := j + 1$ **enddo**
if $i.key = x$ **then Výstup**: $x \in S$ **else Výstup**: $x \notin S$ **endif**

INSERT(x):

Spočítáme $i := h_1(x)$, $h := h_2(x)$, $j := 0$
while $i.key \neq$prázdný a $i.key \neq x$ a $j < m$ **do** $i := i + h \bmod m$, $j := j + 1$ **enddo**
if $j = m$ **then Výstup**: přeplnění, stop **endif**
if $i.key =$prázdný **then** $i.key := x$ **endif**

Komplik. předpis funkce h_2:

Příklad: Mějme universum $U = \{1, 2, \ldots, 1000\}$. Hašovací funkce jsou $h_1(x) = x \bmod 10$ a $h_2(x) = 1 + 2(x \bmod 4)$, když $x \bmod 4 \in \{0, 1\}$, $h_2(x) = 3 + 2(x \bmod 4)$, když $x \bmod 4 \in \{2, 3\}$. Množina je $S = \{1, 7, 11, 53, 73, 141, 161\}$. Tato množina je uložena v levé tabulce. Aplikujme **INSERT**(35). Pak $h_2(35) = 9$, tedy posloupnost pro $x = 35$ je

$$(5, 4, 3, 2, 1, 0, 9, 8, 7, 6).$$

Výsledek je uložen v pravé tabulce.

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

Insert 35

řádek	key
P(0)	11
P(1)	1
P(2)	35
P(3)	73
P(4)	141
P(5)	7
P(6)	53
P(7)	161
P(8)	
P(9)	

Tabulka vznikla posloupností operací:
INSERT(1), **INSERT**(73), **INSERT**(53), **INSERT**(141), **INSERT**(161), **INSERT**(11), **INSERT**(7).
Analýza vyhledávání v dvojitém hašování.

Neúspěšný případ.

Značení: $q_i(n, m)$ – když tabulka má m řádků a je v ní obsazeno n řádků, tak je to pravděpodobnost, že pro každé $j = 0, 1, \ldots, i-1$ je řádek $h_1(x) + jh_2(x)$ obsazen. Pak $q_0(n, m) = 1$, $q_1(n, m) = \frac{n}{m}$, $q_2(n, m) = \frac{n(n-1)}{m(m-1)}$ a obecně

$$q_i(n, m) = \frac{\prod_{j=0}^{i-1}(n - j)}{\prod_{j=0}^{i-1}(m - j)}.$$

$C(n,m)$ – očekávaný počet testů v neúspěšném vyhledávání, když tabulka má m řádků a n jich je obsazeno. Podle definice platí:

$$C(n,m) = \sum_{j=0}^{n} (j+1)\left(q_j(n,m) - q_{j+1}(n,m)\right) = \sum_{j=0}^{n} q_j(n,m).$$

Dále platí $C(0,m) = 1$ pro každé m a $q_j(n,m) = \frac{n}{m} q_{j-1}(n-1,m-1)$ pro všechna $j, n > 0$ a $m > 1$. Odtud

[q_{j-1}(n-1,m-1) ... nějak je závorka uskočená :-( ]

$$C(n,m) = \sum_{j=0}^{n} q_j(n,m) = 1 + \frac{n}{m}\left(\sum_{j=0}^{n-1} q_j(n-1,m-1)\right) = 1 + \frac{n}{m} C(n-1,m-1).$$

Indukcí ukážeme, že $C(n,m) = \frac{m+1}{m-n+1}$. Když $n = 0$, pak $C(0,m) = \frac{m+1}{m-0+1} = 1$ a tvrzení platí. Předpokládáme, že tvrzení platí pro $n - 1 \geq 0$ a pro každé $m \geq n-1$ a dokážeme tvrzení pro n a $m \geq n$. Platí

$$\begin{aligned} C(n,m) &= 1 + \frac{n}{m} C(n-1,m-1) = \\ &1 + \frac{n((m-1)+1)}{m((m-1)-(n-1)+1)} = \\ &1 + \frac{n}{m-n+1} = \frac{m+1}{m-n+1}. \end{aligned}$$

Očekávaný počet dotazů při neúspěšném vyhledávání v tabulce s m řádky, z nichž n je obsazeno, je $\frac{m+1}{m-n+1}$.

Úspěšný případ.

Použijeme metodu ze separujících řetězců. Počet dotazů při vyhledávání x pro $x \in S$ je stejný jako byl počet dotazů při vkládání x do tabulky. Tedy očekávaný počet dotazů při úspěšném vyhledávání v tabulce s m řádky, z nichž n je obsazeno, je

$$\begin{aligned} \frac{1}{n}\sum_{i=0}^{n-1} C(i,m) &= \frac{1}{n}\sum_{i=0}^{n-1} \frac{m+1}{m-i+1} = \\ &\frac{m+1}{n}\left(\sum_{j=1}^{m+1}\frac{1}{j} - \sum_{j=1}^{m-n+1}\frac{1}{j}\right) \approx \\ &\frac{1}{\alpha}\ln\left(\frac{m+1}{m-n+1}\right) \approx \frac{1}{\alpha}\ln\left(\frac{1}{1-\alpha}\right). \end{aligned}$$

Následující tabulka ukazuje tyto hodnoty v závislosti na velikosti α.

hodnota α	0.5	0.7	0.9	0.95	0.99	0.999
$\frac{1}{1-\alpha}$	2	3.3	10	20	100	1000
$\frac{1}{\alpha}\ln(\frac{1}{1-\alpha})$	1.38	1.70	2.55	3.15	4.65	6.9

Porovnání efektivity

Pořadí metod hašování podle očekávaného počtu testů:

Neúspěšné vyhledávání.

Hašování s uspořádanými řetězci,
Hašovaní s řetězci=Hašování s přemísťováním,
Hašování s dvěma ukazateli,
VICH=LICH,
EICH,
LISCH=EISCH,
Dvojité hašování,
Hašování s lineárním přidáváním.

Úspěšné vyhledávání.

Hašování s uspořádanými řetězci=Hašování s řetězci=Hašování s přemísťováním,
Hašování s dvěma ukazateli,
VICH,
LICH,
EICH,
EISCH,
LISCH,
Dvojité hašování,
Hašování s lineárním přidáváním.

Poznámka: Metoda VICH při neúspěšném vyhledávání pro $\alpha < 0.72$ a při úspěšném vyhledávání pro $\alpha < 0.92$ vyžaduje menší očekávaný počet testů než metoda s dvěma ukazateli. Při neúspěšném vyhledávání jsou metody VICH a LICH stejné a jsou o 8% lepší než EICH a o 15% než metody LISCH a EISCH. Při úspěšném vyhledávání je VICH nepatrně lepší než LICH a EICH o 3% lepší než EISCH a o 7% lepší než LISCH.

Očekávaný počet testů při úplně zaplněné tabulce.

Metoda s přemísťováním: neúspěšné vyhledávání 1.5, úspěšné vyhledávání 1.4.
Metoda s dvěma ukazateli: úspěšné i neúspěšné vyhledávání 1.6.
VICH: neúspěšné vyhledávání 1.79, úspěšné vyhledávání 1.67.
LICH: neúspěšné vyhledávání 1.79, úspěšné vyhledávání 1.69.
EICH: neúspěšné vyhledávání 1.93, úspěšné vyhledávání 1.69.
EISCH: neúspěšné vyhledávání 2.1, úspěšné vyhledávání 1.72.
LISCH: neúspěšné vyhledávání 2.1, úspěšné vyhledávání 1.8.

Metodu s lineárním přidáváním je dobré použít jen pro $\alpha < 0.7$, metodu s dvojitým hašováním pro $\alpha < 0.9$, pak čas pro neúspěšné vyhledávání rychle narůstá.

Vliv $\beta = \frac{m}{m'}$ při srůstajícím hašování.
Při úspěšném vyhledávání je optimální hodnota $\beta = 0.85$, při neúspěšném vyhledávání je optimální hodnota $\beta = 0.78$. V praxi se doporučuje použít hodnotu $\beta = 0.86$ (uvedené výsledky byly pro tuto hodnotu β).

Komentář: Metody se separujícími řetězci a srůstající hašování používají více paměti (při srůstajícím hašování součet adresovací a pomocné části). Metoda s přemísťováním a metoda dvojitého hašování vyžaduji více času – na přemístění prvku a na výpočet druhé hašovací funkce.

Další otázky

Jak nalézt volný řádek.
Za nejlepší metodu se považuje mít seznam (zásobník) volných řádků a z jeho vrcholu brát volný řádek a po úspěšné operaci **DELETE** tam zase řádek vložit (pozor při operaci **DELETE** ve strukturách které nepodporují **DELETE**).

Jak řešit přeplnění.
Standardní model: Dána základní velikost tabulky m a pracuje se s tabulkami s $2^i m$ řádky pro vhodné $i = 0, 1, \ldots$. Vhodné i znamená, že faktor zaplnění α je v intervalu $< \frac{1}{4}, 1 >$ (s vyjímkou $i = 0$, kde se uvažuje pouze horní mez). Při překročení meze se zvětší nebo zmenší i a všechna data se přehašují do nové tabulky.
Výhoda: Po přehašování do nové tabulky, počet operací, které vedou k novému přehašování, je alespoň polovina velikosti uložené množiny.
Praktické použití: Nedržet se striktně mezí, používat malé pomocné tabulky při přeplnění a posunout velké přehašování na dobu klidu (aby systém nenechal uživatele v normální době čekat).

Jak řešit **DELETE** v metodách, které ho nepodporují.
Použít ideu tzv. 'falešného **DELETE**'. Odstranit prvek, ale řádek neuvolnit (i v klíči nechat nějakou hodnotu, která bude znamenat, že řádek je prázdný, položky podporující práci s tabulkami neměnit). Řádek nebude v seznamu volných řádků, ale operace **INSERT**, když testuje tento řádek, tak tam může vložit nový prvek. Když je alespoň polovina použitých řádků takto blokována, je vhodné celou strukturu přehašovat. Pravděpodobnostní analýzu tohoto modelu neznám.

Otevřené problémy.

Jak využít ideje z hašování s uspořádanými řetězci pro ostatní metody řešení kolizí (jmenovitě pro srůstající hašování).

Jakou metodu použít pro operaci **DELETE** ve srůstajícím hašování (problém je zachovat náhodnost uložené množiny a tím platnost odhadu na složitost operací).

Jak nalézt druhou hašovací funkci pro metodu dvojitého hašování, aby vzniklé posloupnosti adres při operaci **INSERT** se chovaly jako náhodné?

Závěr.

Připomeňme si předpoklady pro předchozí uvedené výsledky o hašování:

(1) Hašovací funkce se rychle spočítá (v čase $O(1)$);
(2) Hašovací funkce rovnoměrně rozděluje univerzum (to znamená, že pro dvě různé hodnoty i a j hašovací funkce platí $-1 \leq |h^{-1}(i)| - |h^{-1}(j)| \leq 1$);
(3) Vstupní data jsou rovnoměrně rozdělená.

Diskutujme splnitelnost těchto předpokladů.
Předpoklad 1) je jasný.
Předpoklad 2) – je výhodné, když rozdělení univerza hašovací funkcí kopíruje známé rozdělení vstupních dat. Použilo se při návrhu překladače pro FORTRAN (Lum 1971). V následující tabulce jsou uvedené spočítané a naměřené výsledky. Použila se metoda separovaných řetězců. Byly teoreticky spočítané za našich předpokladů. Experiment byl prováděn pomocí hašovací funkce, která preferovala obvyklé názvy identifikátorů. Výsledky byly měřeny, když se překladač FORTRANu použil pro standardní výpočet. Porovnání výsledku:

hodnota α	0.5	0.6	0.7	0.8	0.9
experiment	1.19	1.25	1.28	1.34	1.38
teorie	1.25	1.30	1.35	1.40	1.45

Závěr: Podmínky 1) a 2) můžeme splnit, když známe rozložení vstupních dat, můžeme dosáhnout ještě lepších výsledků.

Nevýhoda: Rozložení vstupních dat nemůžeme ovlivnit a obvykle ho ani neznáme. Je reálné, že rozdělení vstupních dat bude nevhodné pro použitou hašovací funkci. Důsledek – na počátku 70. let se začalo ustupovat od hašování. Hledal se postup, který by se vyhnul uvedenému problému s bodem 3). Nalezenému řešení je věnován následující text.

Univerzální hašování

Řešení navrhli Carter a Wegman (1977), když přišli s metodou univerzálního hašování, která obchází požadavek 3). To vedlo k novému rozsáhlému používání hašování.

Základní idea: Místo jedné funkce máme množinu H funkcí z univerza do tabulky velikosti m takových, že pro každou množinu $S \subseteq U$, $|S| \leq m$ se většina funkcí chová dobře vůči S (tj. S splňuje požadavek 3)). Hašovací funkci zvolíme náhodně z H (s rovnoměrným rozdělením) a hašujeme pomocí takto zvolené funkce.

Modifikace ideje. Ověřování vlastností vyžaduje znalost velikosti množiny H. Rychlá vyčíslitelnost $h(x)$ vyžaduje analytické zadání funkcí v H, ale zjištění rovnosti dvou analyticky zadaných funkcí na univerzu U je problematické. Řešením problému je použití indexové množiny. To znamená, že $H = \{h_i \mid i \in I\}$ a dvě funkce jsou různé, když mají různé indexy. Pak velikost systému bude velikost indexové množiny. Místo zvolení hašovací funkce budeme volit náhodně index s rovnoměrným rozložením a když zvolíme index i, pak budeme pracovat s hašovací funkcí h_i. Očekávaná hodnota náhodné proměnné f z množiny I do reálných čísel bude průměr přes I, tj. $\frac{\sum_{i \in I} f(i)}{|I|}$.

Def:

Formálně: Nechť U je univerzum. Soubor funkcí $H = \{h_i \mid i \in I\}$ z univerza U do množiny $\{0, 1, \ldots, m-1\}$ se nazývá c-univerzální (c je kladné reálné číslo), když

$$\forall x, y \in U,\ x \neq y \text{ platí } |\{i \in I \mid h_i(x) = h_i(y)\}| \leq \frac{c|I|}{m}.$$

Ekviv. def:

Jako ekvivalentní definici lze použít toto tvrzení: systém funkcí H z univerza U do množiny $\{0,1,\ldots,m-1\}$ je c-univerzální, když vybíráme funkci $h \in H$ s rovnoměrným rozdělením, pak pro každé dvě různá $x, y \in U$, platí

$$\text{Prob}\left(h\left(x\right) = h\left(y\right)\right) \leq \frac{c}{m}.$$

Problémy: existence c-univerzálních systémů,
vlastnosti c-univerzálních systémů (zda splňují požadované ideje).

Existence univerzálních systémů.

Def. mn. hašovacích funkcí:

Univerzum $U = \{0,1,\ldots,N-1\}$ pro prvočíslo N,
$H = \{h_{a,b} \mid (a,b) \in U \times U\}$,
kde $h_{a,b}\left(x\right) = \left(\left(ax+b\right) \bmod N\right) \bmod m$
(tj. indexová množina je $U \times U$ a její velikost je N^2).

Výhoda: funkce z množiny H umíme rychle vyčíslit.

Zvolme $x, y \in U$ taková, že $x \neq y$. Chceme nalézt $(a,b) \in U \times U$ takové, že $h_{a,b}\left(x\right) = h_{a,b}\left(y\right)$.

Musí existovat $i \in \{0,1,\ldots,m-1\}$ a $r, s \in \left\{0,1,\ldots,\left\lceil \frac{N}{m} \right\rceil - 1\right\}$ tak, že platí

$$\begin{aligned} \left(ax+b \equiv i + rm\right) \bmod N \\ \left(ay+b \equiv i + sm\right) \bmod N \end{aligned}$$

Když x, y, i, r a s jsou konstanty a a a b jsou proměnné, je to systém lineárních rovnic v tělese $\mathbb{Z}/ \bmod N$, kde $\mathbb{Z}$ jsou celá čísla. Matice soustavy

Zn

$$\begin{pmatrix} x & 1 \\ y & 1 \end{pmatrix}$$

Dle Frobeniovy věty

je regulární, protože $x \neq y$. Tedy existuje jediné řešení této soustavy pro fixovaná x, y, i, r a s. Pro daná x a y, i nabývá m hodnot, r a s nabývají $\left\lceil \frac{N}{m} \right\rceil$ hodnot.

Závěr: pro každá $x, y \in U$ taková, že $x \neq y$, existuje $m\left(\left\lceil \frac{N}{m} \right\rceil\right)^2$ dvojic $(a,b) \in U \times U$ takových, že $h_{a,b}\left(x\right) = h_{a,b}\left(y\right)$.

Věta. *Množina H je c-univerzální pro*

$$c = \frac{\left(\left\lceil \frac{N}{m} \right\rceil\right)^2}{\left(\frac{N}{m}\right)^2}.$$

Skutečně, pro každé $x, y \in U$, $x \neq y$, je počet $(a,b) \in U \times U$ takových, že $h_{a,b}\left(x\right) = h_{a,b}\left(y\right)$, nejvýše roven

$$m\left(\left\lceil \frac{N}{m} \right\rceil\right)^2 = \frac{\left(\left\lceil \frac{N}{m} \right\rceil\right)^2}{\left(\frac{N}{m}\right)^2} \frac{N^2}{m} = \frac{\left(\left\lceil \frac{N}{m} \right\rceil\right)^2}{\left(\frac{N}{m}\right)^2} \frac{|I|}{m}.$$

Závěr: Dokázali jsme existenci c-univerzálních systémů pro c blízké 1. Stačí si uvědomit, že každé univerzum můžeme považovat za univerzum tvaru $\{0,1,\ldots,N-1\}$ pro nějaké N a že mezi čísly N a $2N$ vždy existuje nějaké prvočíslo.

Vlastnosti univerzálního hašování.

Předpoklad: $H = \{h_i \mid i \in I\}$ je c-univerzální systém funkcí

Def:

Označení: Pro $i \in I$ a prvky $x, y \in U$ označme

$$\delta_i(x,y) = \begin{cases} 1 & \text{když } x \neq y \text{ a } h_i(x) = h_i(y), \\ 0 & \text{když } x = y \text{ nebo } h_i(x) \neq h_i(y). \end{cases}$$

[Na stejné hašovací funkci kolize mezi x a y]

Def:

Pro množinu $S \subseteq U$, $x \in U$ a $i \in I$ definujme

$$\delta_i(x,S) = \sum_{y \in S} \delta_i(x,y).$$

Pro fixovanou množinu $S \subseteq U$ a pro fixované $x \in U$ sečteme $\delta_i(x,S)$ přes všechna $i \in I$:

$$\sum_{i \in I} \delta_i(x,S) = \sum_{i \in I} \sum_{y \in S} \delta_i(x,y) = \sum_{y \in S} \sum_{i \in I} \delta_i(x,y) =$$

$$\sum_{y \in S, y \neq x} |\{i \in I \mid h_i(x) = h_i(y)\}| \leq$$

$$\sum_{y \in S, y \neq x} c\frac{|I|}{m} = \begin{cases} (|S|-1)\, c\frac{|I|}{m} & \text{když } x \in S, \\ |S| c\frac{|I|}{m} & \text{když } x \notin S. \end{cases}$$

Protože $\delta_i(x,S)$ dává odhad na velikost řetězce $h_i(x)$ při reprezentaci množiny S pomocí funkce h_i, dostáváme, že očekávaná délka řetězce pro fixovanou množinu $S \subseteq U$ a fixované $x \in U$ přes $i \in I$ s rovnoměrným rozdělením je nejvýše

$$\frac{1}{|I|} \sum_{i \in I} \delta_i(x,S) \leq \begin{cases} c\frac{|S|-1}{m} & \text{když} \quad x \in S, \\ c\frac{|S|}{m} & \text{když} \quad x \notin S. \end{cases}$$

Jen vydělím |I|

Věta. *Očekávaný čas operací* **MEMBER**, **INSERT** *a* **DELETE** *při c-univerzálním hašování je $O(1 + c\alpha)$, kde α je faktor naplnění (tj. $\alpha = \frac{|S|}{m}$).*
Očekávaný čas pro pevnou posloupnost n operací **MEMBER**, **INSERT** *a* **DELETE** *aplikovaných na prázdnou tabulku pro c-univerzální hašování je $O\left(\left(1 + \frac{c}{2}\alpha\right) n\right)$, kde $\alpha = \frac{n}{m}$.*

Význam výsledku: Vzorec se jen o multiplikativní konstantu c liší od vzorce pro hašování se separovanými řetězci. Přitom c může být jen o málo menší než 1 a ve všech známých příkladech je $c \geq 1$. Takže, co jsme dosáhli? Rozdíl je v předpokladech. Zde je předpoklad 3) nahrazen předpokladem, že index $i \in I$ je vybrán s rovnoměrným rozdělením, a není žádný předpoklad na vstupní data. **Výběr indexu i můžeme ovlivnit, ale výběr vstupních dat nikoliv.** Můžeme zajistit rovnoměrné rozdělení výběru i z I nebo se k tomuto rozdělení hodně přiblížit.

Markovova nerovnost.

Předpoklady: Je dána množina $S \subseteq U$, prvek $x \in U$. Očekávaná velikost $\delta_i(x, S)$ je μ, a $t \geq 1$.

Ukážeme pro $t > 1$, že pravděpodobnost, že $\delta_i(x, S) \geq t\mu$ pro $i \in I$, je menší než $\frac{1}{t}$ (předpokládáme, že i je z I vybráno s rovnoměrným rozdělením).

Označme $I' = \{i \in I \mid \delta_i(x, S) \geq t\mu\}$. Pak platí

$$\mu = \frac{\sum_{i \in I} \delta_i(x, S)}{|I|} > \frac{\sum_{i \in I'} \delta_i(x, S)}{|I|} \geq \frac{\sum_{i \in I'} t\mu}{|I|} = \frac{|I'|}{|I|} t\mu$$

Kratím μ na obou stranách rovnice.

Odtud

$$|I'| < \frac{|I|}{t}.$$

Závěr: Pravděpodobnost, že $\delta_i(x, S) \geq t\mu$, je menší než $\frac{1}{t}$, a odtud plyne požadované tvrzení.

Poznámka: Toto tvrzení platí obecně a nazývá se Markovova nerovnost. Uvedený důkaz ilustruje jednoduché tvrzení pro konečný případ.

Problémy.

Hlavní problém: Zajištění rovnoměrného rozdělení výběru i z I.

Provedení výběru: Zakódovat indexy z množiny I do čísel $0, 1, \ldots, |I| - 1$. Zvolit náhodně číslo i z tohoto intervalu s rovnoměrným rozdělením a pak použít funkci s indexem, jehož kód je i. Abychom vybrali i, nalezneme nejmenší j takové, že $2^j - 1 \geq |I| - 1$. Pak čísla v intervalu $\{0, 1, \ldots, 2^j - 1\}$ jednoznačně korespondují s posloupnostmi 0 a 1 délky j. Budeme vybírat náhodně posloupnost 0 a 1 délky j. K výběru posloupnosti použijeme náhodný generátor rovnoměrného rozdělení.

Závada: Skutečný náhodný generátor pro rovnoměrné rozdělení je prakticky nedosažitelný (některé fyzikální procesy). K dispozici je pouze pseudogenerátor.

Jeho nevýhoda: Čím je j větší, tím je posloupnost pravidelnější (tj. méně náhodná).

Důsledky: Nalézt co nejmenší c-univerzální systémy. Nalézt dolní odhady na jejich velikost.

Dolní odhady na velikost.

Předpoklady: Nechť U je universum velikosti N a nechť $H = \{h_i \mid i \in I\}$ je c-univerzální systém funkcí hašujících do tabulky velikosti m. Můžeme předpokládat, že

$$I = \{0, 1, \ldots, |I| - 1\}.$$

Indukcí definujme množiny $U_0, U_1, \ldots$ tak, že: $U_0 = U$.
Nechť U_1 je největší podmnožina U_0 vzhledem k počtu prvků taková, že $h_0(U_1)$ je jednoprvková množina.
Nechť U_2 je největší podmnožina U_1 vzhledem k počtu prvků taková, že $h_1(U_2)$ je jednoprvková množina.

Nechť U_3 je největší podmnožina U_2 vzhledem k počtu prvků taková, že $h_2(U_3)$ je jednoprvková množina.
Obecně, nechť U_i je největší podmnožina U_{i-1} vzhledem k počtu prvků taková, že $h_{i-1}(U_i)$ je jednoprvková množina.

Protože hašujeme do tabulky velikosti m, platí $|U_i| \geq \left\lceil \frac{|U_{i-1}|}{m} \right\rceil$. Protože $|U_0| = N$, indukcí dostaneme, že $|U_i| \geq \lceil \frac{N}{m^i} \rceil$ pro každé i. Zvolme $i = \lceil \log_m N \rceil - 1$. Pak i je největší přirozené číslo takové, že $\frac{N}{m^i} > 1$. Tedy U_i má aspoň dva prvky, zvolme $x, y \in U_i$ taková, že $x \neq y$. Pak $h_j(x) = h_j(y)$ pro $j = 0, 1, \ldots, i-1$. Tedy

$$i \leq |\{j \in I \mid h_j(x) = h_j(y)\}| \leq \frac{c|I|}{m}. \qquad \text{(\#1)}$$

Věta. *Když $H = \{h_i \mid i \in I\}$ je c-univerzální systém pro univerzum U o velikosti N hašující do tabulky s m řádky, pak*

$$|I| \geq \frac{m}{c}\left(\lceil \log_m N \rceil - 1\right).$$

Posloupnosti 0 a 1 při náhodné volbě i z I musí mít délku alespoň

$$\lceil (\log m - \log c + \log\log N - \log\log m) \rceil$$

(zde všechny logaritmy jsou o základu 2).

Malý univerzální systém.

Zkonstruujeme c-univerzální systém takový, že logaritmus z velikosti jeho indexové množiny pro velká univerza je až na aditivní konstantu menší než $4(\log m + \log\log N)$, kde N je velikost univerza a m je počet řádků v tabulce.

Nechť $p_1, p_2, \ldots$ je rostoucí posloupnost všech prvočísel. Mějme velikost tabulky m a univerzum $U = \{0, 1, \ldots, N-1\}$ pro nějaké přirozené číslo N (nemusí být prvočíslo). Nechť t je nejmenší číslo takové, že $t \ln p_t \geq m \ln N$. Definujme

Def:

$$H_1 = \{g_{c,d}(h_\ell) \mid t < \ell \leq 2t,\ c, d \in \{0, 1, \ldots, p_{2t} - 1\}\},$$

kde $h_\ell(x) = x \bmod p_\ell$ a $g_{c,d}(x) = ((cx + d) \bmod p_{2t}) \bmod m$.
Ukážeme, že když $m(\ln m + \ln\ln m) < N$, pak H_1 je 3.25-univerzální systém.

Pozn: Nejprve si připomeneme známou větu o velikosti prvočísel (zde ln je přirozený logaritmus, tj. o základu e).

Věta. *Pro každé $i = 1, 2, \ldots$ platí $p_i > i \ln i$ a pro $i \geq 6$ platí $p_i < i(\ln i + \ln\ln i)$.* □

Tedy pro $i \geq 6$ platí $p_i < 2i \ln i$. (\#2)

Velikost indexové množiny H_1. Indexová množina H_1 je

$$I = \{(c, d, \ell) \mid c, d \in \{0, 1, \ldots, p_{2t} - 1\},\ t < \ell \leq 2t\}.$$

(\#3) Tedy $|I| = t p_{2t}^2$. Odtud plyne $|I| \leq 16 t^3 \ln^2 2t$ a tedy

$$\log(|I|) \leq 4 + 3\log t + 2\log\log t.$$

Pro dostatečně velké t (takové, že $\log t \geq 2\log\log t$, tj. $t \geq 16$) platí, že $\log(|I|) \leq 4 + 4\log t$.
Z definice t plyne, že $t \leq m \ln N$, když $\ln p_t \geq 1$ (tj. $p_t \geq 3$).
Závěr: $\log(|I|) \leq 4 + 4(\log m + \log\log N)$.

Univerzalita malého systému.

Zvolme různá x a y z univerza U. Označíme

$$G_1 = \{(c,d,\ell) \mid g_{c,d}(h_\ell(x)) = g_{c,d}(h_\ell(y)),\ h_\ell(x) \neq h_\ell(y)\},$$
$$G_2 = \{(c,d,\ell) \mid g_{c,d}(h_\ell(x)) = g_{c,d}(h_\ell(y)),\ h_\ell(x) = h_\ell(y)\}$$

a odhadneme velikost G_1 a G_2.

Odhad velikosti G_1. Když $(c,d,\ell) \in G_1$, pak existují $r,s \in \left\{0,1,\ldots,\left\lceil \frac{p_{2t}}{m} \right\rceil - 1\right\}$ a $i \in \{0,1,\ldots,m-1\}$ taková, že

$$(c\,(x \bmod p_\ell) + d \equiv i + rm) \bmod p_{2t}$$
$$(c\,(y \bmod p_\ell) + d \equiv i + sm) \bmod p_{2t}.$$

Když c a d považujeme za neznámé, pak je to soustava lineárních rovnic s regulární maticí (protože $x \bmod p_\ell \neq y \bmod p_\ell$), a tedy pro každé ℓ, i, r a s existuje nejvýše jedna dvojice (c,d). Proto

$$|G_1| \leq tm\left(\left\lceil \frac{p_{2t}}{m} \right\rceil\right)^2 \leq \frac{tp_{2t}^2}{m}\left(1+\frac{m}{p_{2t}}\right)^2 = \frac{|I|}{m}\left(1+\frac{m}{p_{2t}}\right)^2.$$

Odhad velikosti G_2. Označme $L = \{\ell \mid t < \ell \leq 2t,\ x \bmod p_\ell = y \bmod p_\ell\}$ a $P = \prod_{\ell \in L} p_\ell$. Protože P dělí $|x-y|$, dostáváme, že $P \leq N$. Protože $p_t < p_\ell$ pro každé $\ell \in L$, dostáváme, že $P > p_t^{|L|}$. Tedy $|L| \leq \frac{\ln N}{\ln p_t} \leq \frac{t}{m}$ z definice t. Protože $(c,d,\ell) \in G_2$, právě když $\ell \in L$ a $c,d \in \{0,1,\ldots,p_{2t}-1\}$, shrneme, že

$$|G_2| \leq |L|p_{2t}^2 \leq \frac{tp_{2t}^2}{m} = \frac{|I|}{m}.$$

Abychom odhadli $\left(1+\frac{m}{p_{2t}}\right)^2$, ukážeme si nejdřív pomocné lemma.

Lemma. *Když $t \geq 6$ a $m(\ln m + \ln\ln m) < N$, pak $m < \frac{p_t}{\ln t}$.*

Důkaz. Předpokládejme, že tvrzení neplatí. pak $m \geq \frac{p_t}{\ln t}$. Z Věty o velikosti prvočísel pak plyne $m \geq \frac{p_t}{\ln t} > \frac{t \ln t}{\ln t} = t$. Když použijeme, že $m(\ln m + \ln\ln m) < N$, tak dostaneme, že

$$\ln m + \ln(\ln m + \ln\ln m) < \ln N,$$

a odtud plyne, že

$$t \ln p_t < t \ln(t(\ln t + \ln\ln t)) \leq m(\ln m + \ln(\ln m + \ln\ln m)) < m \ln N$$

Definice t

a to je spor s definicí t. Tedy $m < \frac{p_t}{\ln t}$. □

Nyní zkombinujeme Větu o odhadu velikosti prvočísel, předchozí Lemma a fakt, že

$$\ln 2t \geq \ln t \geq \ln\ln t \qquad \text{pro všechna } t \geq 1$$

Odhadujeme $(1+m/p_{2t})^2$

a dostaneme, že

$$\frac{m}{p_{2t}} \leq \frac{\frac{p_t}{\ln t}}{2t \ln 2t} < \frac{t(\ln t + \ln\ln t)}{2t \ln t \ln 2t} < \frac{1}{\ln 2t}\left(1 + \frac{\ln\ln t}{\ln t}\right).$$

zanedbame

Je zřejmé, že tento výraz je menší než $\frac{1}{2}$ a když konverguje k $+\infty$ pak tento výraz konverguje k 0.

Z toho plyne, že $\left(1 + \frac{m}{p_{2t}}\right)^2 \leq 1.5^2 = 2.25$ a tedy

$$|\{i \in I \mid h_i(x) = h_i(y)\}| = |G_1| + |G_2| \leq$$
$$\frac{|I|}{m}\left(1 + \frac{m}{p_{2t}}\right)^2 + \frac{|I|}{m} \leq \frac{|I|}{m}(1 + 2.25) = 3.25\frac{|I|}{m}.$$

Shrnutí: Když $t \geq 6$ a $m \ln m \ln\ln m < N$, pak H_1 je 3.25-univerzální. Bez jakýchkoliv předpokladů lze ukázat, že H_1 je 5-univerzální.

Odhad na velikost c.

Věta. *Když H je c-univerzální systém univerza U o velikosti N hašující do tabulky s m řádky, pak $c \geq 1 - \frac{m}{N}$.*

Nejprve dokážeme technické lemma.

Lemma. *Mějme reálná čísla b_i pro $i = 0, 1, \ldots, m-1$ a nechť $b = \sum_{i=0}^{m-1} b_i$. Pak*

$$\sum_{i=0}^{m-1} b_i(b_i - 1) \geq b\left(\frac{b}{m} - 1\right).$$

Důkaz lemmatu. Z Cauchyho-Schwarzovy nerovnosti

$$\left(\sum_{i=0}^{m-1} x_i y_i\right)^2 \leq \left(\sum_{i=0}^{m-1} x_i^2\right)\left(\sum_{i=0}^{m-1} y_i^2\right)$$

b^2

nasčítáme m, protože každý člen je 1

plyne $\left(\sum_{i=0}^{m-1} b_i\right)^2 = b^2 \leq m\left(\sum_{i=0}^{m-1} b_i^2\right)$, stačí položit $x_i = b_i$ a $y_i = 1$, a tedy $\frac{b^2}{m} \leq \sum_{i=0}^{m-1} b_i^2$. Odtud

$$\sum_{i=0}^{m-1} b_i(b_i - 1) = \sum_{i=0}^{m-1} b_i^2 - \sum_{i=0}^{m-1} b_i = \sum_{i=0}^{m-1} b_i^2 - b \geq \frac{b^2}{m} - b = b\left(\frac{b}{m} - 1\right)$$

a lemma je dokázáno. □

Důkaz Věty. Mějme funkci $f : U \to S$, kde U má velikost N a S má velikost m. Označme A množinu uspořádaných dvojic $u, v \in U$ takových, že $u \neq v$ a $f(u) = f(v)$. Když pro $s \in S$ označíme $k_s = |f^{-1}(s)|$, pak $|A| = \sum_{s \in S} k_s (k_s - 1)$. Z lemmatu plyne, že

$$|A| = \sum_{s \in S} k_s (k_s - 1) \geq N \left(\frac{N}{m} - 1 \right) = N \left(\frac{N - m}{m} \right),$$

protože $\sum_{s \in S} k_s = N$.
Když $H = \{h_i \mid i \in I\}$ je c-univerzální systém funkcí z univerza U o velikosti N do tabulky o velikosti m, pak pomocí lemmatu dostáváme

$$\begin{aligned} |I| N \left(\frac{N - m}{m} \right) &\leq \\ \sum_{i \in I} |\{(x, y) \in U \times U \mid h_i(x) = h_i(y),\ x \neq y\}| &= \\ \sum_{(x,y) \in U \times U,\, x \neq y} |\{i \in I \mid h_i(x) = h_i(y)\}| &\leq \\ \sum_{(x,y) \in U \times U, x \neq y} c \frac{|I|}{m} &= N(N-1) c \frac{|I|}{m}. \end{aligned}$$

Odtud plyne, že $N - m \leq c(N - 1)$, a tedy

Porovnavame pouze obe strany nerovnice

$$c \geq \frac{N - m}{N - 1} > \frac{N - m}{N} = 1 - \frac{m}{N}. \quad \square$$

Problémy univerzálního hašování.

Použít jiné metody na řešení kolizí než separované řetězce. Jak to ovlivní použitelnost univerzálního hašování? Platí podobné vztahy jako pro pevně danou hašovací funkci? Jaký vliv na efektivnost má nepřítomnost operace **DELETE**?

Existuje c-univerzální hašovací systém pro $c < 1$? Jaký je vztah mezi velikostí c-univerzálního hašovacího systému a velikostí c? Lze zkonstruovat malý c-univerzální systém pro $c < 3.25$? Zde hraje roli fakt, že při $c = 3.25$ se očekávaná délka řetězce může pohybovat až kolem hodnoty 7.

Použití Čebyševovy nerovnosti místo Markovovy nerovnosti dává kvadratický odhad pravděpodobnosti, že délka řetězce je o t větší než očekávaná hodnota. Za jakých okolností dává lepší odhad? Lze použít i vyšších momentů?

Jak použít Markovou nerovnost a očekávanou délku maximálního řetězce pro odhad očekávaného počtu voleb hašovací funkce? Pro jaké parametry lze použít následující model?

Jiny model:

Je dána základní velikost tabulky m a dále pro $j = 0, 1, \ldots$ čísla (parametry) l_j a c-univerzální hašovací systémy $H_j = \{h_i \mid i \in I_j\}$ z univerza do tabulky s $m2^j$ řádky.
Množina $S \subseteq U$ je reprezentována následovně: je dáno j takové, že když $j > 0$, pak

$m2^{j-2} \leq |S| \leq m2^j$, když $j = 0$, pak $|S| \leq m$, a je zvolen index $i \in I_j$. Dále máme prosté řetězce $r_0, r_1, \ldots, r_{m2^j-1}$, jejichž délky jsou nejvýše l_j, a řetězec r_k obsahuje prvky $\{s \in S \mid h_i(s) = k\}$.
Operace **INSERT**(x) prohledá řetězec $r_{h_i(x)}$ a když tento řetězec neobsahuje prvek x, pak ho přidá. Když $m2^{j-2} \leq |S| \leq m2^j$ a délka řetězce $r_{h_i}(x)$ je nejvýše l_j, pak operace končí. Když $|S| > m2^j$, tak se nejdříve zvětší j o 1. Pak se náhodně zvolí $i \in I_j$ a zkonstruují se řetězce reprezentující S. Když některý z nich má délku větší než l_j, tak se volba a konstrukce řetězců opakuje tak dlouho, dokud se nepovede zvolit $i \in I_j$ takové, že všechny zkonstruované řetězce mají délku nejvýše l_j. Operace **DELETE** se řeší analogicky.
Problém: Jak volit parametry l_i?

V případě řešení kolizí dvojitým hašováním nebo hašováním s lineárním přidáváním je třeba dát silnější podmínky na velikost $|S|$. V poslední době se této tématice věnuje pozornost a byla dosažena řada zajímavých výsledků.

Perfektní hašování

Jiné řešení kolizí je perfektní hašování. Idea je nalézt pro danou množinu hašovací funkci, která nevytváří kolize.

Nevýhoda: Metoda nepřipouští operaci **INSERT** (pro nový vstup nemůžeme zaručit, že nevznikne kolize). Metodu lze prakticky použít pro úlohy, kde lze očekávat hodně operací **MEMBER** a operace **INSERT** se téměř nevyskytuje (kolize se řeší pomocí malé pomocné tabulky, kam se ukládají kolidující data). Tato metoda se používá při navrhování kompilátorů.

Zadání úlohy: Pro danou množinu $S \subseteq U$ chceme nalézt hašovací funkci h takovou, že

(1) pro $s, t \in S$ takové, že $s \neq t$, platí $h(s) \neq h(t)$ (tj. h je perfektní hašovací funkce pro S);
(2) h hašuje do tabulky s m řádky, kde m je přibližně stejně velké jako $|S|$ (není praktické hašovat do příliš velkých tabulek – ztrácí se jeden ze základních důvodů pro hašování);
(3) h musí být rychle spočitatelná – jinak hašování není rychlé;
(4) uložení h nesmí vyžadovat moc paměti, nejvýhodnější je analytické zadání (když zadání h bude vyžadovat moc paměti, např. když by byla dána tabulkou, pak se ztrácí důvod k použití stejně jako v bodě 2).

Kompenzace: Nalezení hašovací funkce může spotřebovat více času. Provádí se jen na začátku úlohy.

Uvedené požadavky motivují zavedení následujícího pojmu.

Def: Mějme univerzum $U = \{0, 1, \ldots, N-1\}$. Soubor funkcí H z U do množiny $\{0, 1, \ldots, m-1\}$ se nazývá (N, m, n)-perfektní, když pro každou $S \subseteq U$ takovou, že $|S| = n$, existuje $h \in H$ perfektní pro S (tj. $h(s) \neq h(t)$ pro každá dvě různá $s, t \in S$).

Protože nevíme, zda taková h existují, nejprve vyšetříme množiny perfektních hašovacích funkcí. Vyšetříme vlastnosti (N, m, n)-perfektních souborů funkcí.

Dolní odhady na velikost (N, m, n)-perfektního souboru.

Předpokládejme, že H je (N, m, n)-perfektní systém pro $U = \{0, 1, \ldots, N-1\}$ a nejprve nalezneme dolní odhady na velikost $|H|$.

Mějme funkci h z U do množiny $\{0, 1, \ldots, m-1\}$. Nalezneme počet množin $S \subseteq U$ takových, že h je perfektní funkce pro S a $|S| = n$. Funkce h je perfektní pro $S \subseteq U$, právě když pro každé $i = 0, 1, \ldots, m-1$ je $|h^{-1}(i) \cap S| \leq 1$. Odtud počet těchto množin je

$$\sum \left\{ \prod_{j=0}^{n-1} |h^{-1}(i_j)| \mid 0 \leq i_0 < i_1 < \cdots < i_{n-1} < m \right\}.$$

Vysvětlení: $h(S) = \{i_j \mid j = 0, 1, \ldots, n-1\}$.

Toto číslo je maximální, když $|h^{-1}(i)| = \frac{N}{m}$ pro každé i. Tedy h může být perfektní nejvýše pro $\binom{m}{n}\left(\frac{N}{m}\right)^n$ množin (číslo $\binom{m}{n}$ určuje počet posloupností $0 \leq i_0 < i_1 < \cdots < i_{n-1} < m$). Protože n-prvkových podmnožin universa je $\binom{N}{n}$, dostáváme, že

$$|H| \geq \frac{\binom{N}{n}}{\binom{m}{n}\left(\frac{N}{m}\right)^n}.$$

Jiný odhad velikosti (N, m, n)-perfektního souboru.

Předpokládejme, že $H = \{h_1, \ldots, h_t\}$ je (N, m, n)-perfektní soubor funkcí. Definujme indukcí soubor množin U_i:

Induktivní definice množin U_i:

$U_0 = U$ a pro $i > 0$ je U_i největší podmnožina U_{i-1}, co do počtu prvků, taková, že h_i je konstantní na U_i. Pak $|U_i| \geq \frac{|U_{i-1}|}{m}$ pro všechna $i > 0$. Z $|U_0| = N$ plyne $|U_i| \geq \frac{N}{m^i}$. Pro každé $i = 1, 2, \ldots, t$ je $h_j(U_i)$ jednobodová množina pro každé $j \leq i$. Proto žádná h_j pro $j \leq i$ není perfektní pro množinu $S \subseteq U$ takovou, že $|S \cap U_i| \geq 2$. Protože H je (N, m, n)-perfektní, musí být $|U_t| \leq 1$, a tedy $\frac{N}{m^t} \leq 1$. Proto $t \geq \frac{\log N}{\log m}$.

t je počet hashovacích funkcí.

Věta. *Když H je (N, m, n)-perfektní soubor funkcí, pak*

$$|H| \geq \max\left\{ \frac{\binom{N}{n}}{\binom{m}{n}\left(\frac{N}{m}\right)^n}, \frac{\log N}{\log m} \right\}.$$

Existence (N, m, n)-perfektního souboru.

Mějme univerzum $U = \{0, 1, \ldots, N-1\}$ a soubor funkcí $H = \{h_1, h_2, \ldots, h_t\}$ z univerza U do množiny $\{0, 1, \ldots, m-1\}$. Reprezentujeme tento soubor pomocí matice $M(H)$ typu $N \times t$ s hodnotami $\{0, 1, \ldots, m-1\}$ tak, že pro $x \in U$ a $i = 1, 2, \ldots, t$ je v x-tém řádku a i-tém sloupci matice $M(H)$ hodnota $h_i(x)$. Pak žádná funkce z H není perfektní pro množinu $S = \{s_1, s_2, \ldots, s_n\} \subseteq U$, právě když podmatice $M(H)$ tvořená řádky s_1, s_2, až s_n a všemi sloupci nemá prostý sloupec. Takových matic je nejvýše

$$\left(m^n - \prod_{i=0}^{n-1}(m-i)\right)^t m^{(N-n)t}.$$

t (pocet hash. fci)

N

h_i(x) prvky, kde

x \in U
i \in {1,..,t}

Doplneni matice na matici typu N x n

Vysvětlení: m^n je počet všech funkcí z S do $\{0,1,\ldots,m-1\}$, $\prod_{i=0}^{n-1}(m-i)$ je počet prostých funkcí z S do $\{0,1,\ldots,m-1\}$, a tedy počet všech podmatic s n řádky takových, že žádný jejich sloupec není prostý, je $\left(m^n-\prod_{i=0}^{n-1}(m-i)\right)^t$. Tyto podmatice můžeme libovolně doplnit na matici typu $N\times n$ a pro každou matici je těchto doplnění $m^{(N-n)t}$.

Podmnožin U velikosti n je $\binom{N}{n}$, tedy počet všech matic, které nereprezentují (N,m,n)-perfektní systém, je menší nebo rovno

$$\binom{N}{n}\left(m^n-\prod_{i=0}^{n-1}(m-i)\right)^t m^{(N-n)t}.$$

Všech matic je m^{Nt} a když

$$(*)\qquad \binom{N}{n}\left(m^n-\prod_{i=0}^{n-1}(m-i)\right)^t m^{(N-n)t}<m^{Nt},$$

pak nutně existuje (N,m,n)-perfektní systém. Následující výrazy jsou ekvivalentní s nerovností $(*)$

Vyraz (*) vydeleny m^{Nt}

$$\binom{N}{n}\left(1-\frac{\prod_{i=0}^{n-1}(m-i)}{m^n}\right)^t<1 \Leftrightarrow t\geq\frac{\ln\binom{N}{n}}{-\ln\left(1-\frac{\prod_{i=0}^{n-1}(m-i)}{m^n}\right)}.$$

Protože $\ln\binom{N}{n}\leq n\ln N$ a protože $-\ln(1-x)\geq x$ pro $x\in(0,1)$ dostáváme

$$-\ln\left(1-\frac{\prod_{i=0}^{n-1}(m-i)}{m^n}\right)\geq\prod_{i=0}^{n-1}\left(1-\frac{i}{m}\right)=e^{\sum_{i=0}^{n-1}\ln\left(1-\frac{i}{m}\right)}\geq$$
$$e^{\int_0^n\ln\left(1-\frac{x}{m}\right)dx}$$

kde integrál můžeme odhadnout

$$m\left[\left(1-\frac{n}{m}\right)\left(1-\ln\left(1-\frac{n}{m}\right)\right)-1\right]\geq m\left[\left(1-\frac{n}{m}\right)\left(1+\frac{n}{m}\right)-1\right]>-\frac{n^2}{m},$$

dostáváme, že když $t\geq n(\ln N)e^{\frac{n^2}{m}}$, pak (*) platí, a tedy existuje (N,m,n)-perfektní soubor funkcí.

Existence (N,m,n)-perfektního souboru funkcí ale nezaručuje splnění požadavků 2), 3) a 4). Abychom uspěli, použijeme ideu z metody univerzálního hašování.

Konstrukce perfektní hašovací funkce.

Předpoklady: $U = \{0, 1, \ldots, N-1\}$, kde N je prvočíslo. Mějme $S \subseteq U$ o velikosti n.

Hash. fce:

Budeme uvažovat funkce

$$h_k(x) = (kx \bmod N) \bmod m \qquad \text{pro } k = 1, 2, \ldots, N-1. \qquad \text{(\#1)}$$

Hodnoty b_i^k:

Pro $i = 0, 1, \ldots, m-1$ a $k = 1, 2, \ldots, N-1$ označme

$$b_i^k = |\{x \in S \mid \underbrace{(kx \bmod N) \bmod m = i}_{h\_k(x) = i}\}|.$$

Význam b_i^k: Hodnoty b_i^k lze považovat za veličiny, které ukazují odchylku od perfektnosti. Všimněme si, že

$$\text{když } b_i^k \geq 2, \text{ pak } \left(b_i^k\right)^2 - b_i^k \geq 2,$$

protože $a^2 - a \geq 2$, když $a \geq 2$. Na druhou stranu

$$b_i^k \leq 1 \text{ implikuje } \left(b_i^k\right)^2 - b_i^k = 0.$$

Tedy z $\sum_{i=0}^{m-1} b_i^k = n$ plyne

Věta. *Funkce h_k je perfektní, právě když $\sum_{i=0}^{m-1} \left(b_i^k\right)^2 - n < 2$.*

Nyní odhadneme výraz $\sum_{k=1}^{N-1} \left(\left(\sum_{i=0}^{m-1} \left(b_i^k\right)^2\right) - n\right)$.

$$\sum_{k=1}^{N-1} \left(\left(\sum_{i=0}^{m-1} \left(b_i^k\right)^2 \right) - n \right) =$$

$$\sum_{k=1}^{N-1} \left(\left(\sum_{i=0}^{m-1} |\{x \in S \mid h_k(x) = i\}|^2 \right) - n \right) =$$

$$\sum_{k=1}^{N-1} |\{(x, y) \mid x, y \in S,\ x \neq y,\ h_k(x) = h_k(y)\}| =$$

$$\sum_{x,y \in S, x \neq y} |\{k \mid 1 \leq k < N,\ h_k(x) = h_k(y)\}|.$$

Rozebirame h_k(x)=h_k(y):

Zvolme $x, y \in S$ taková, že $x \neq y$. Pak $h_k(x) = h_k(y)$, právě když existuje $i = 0, 1, \ldots, m-1$ a $r, s = 0, 1, \ldots, \lfloor \frac{N}{m} \rfloor$ taková, že $kx \equiv i + rm \bmod N$ a $ky \equiv i + sm \bmod N$ a $i + rm, i + sm < N$. Odtud dostáváme, že $h_k(x) = h_k(y)$ implikuje $kx - ky \equiv (r - s)\, m \bmod N$. Protože $0 < k < n$ a $x \neq y$, dostáváme, že $kx - ky \neq 0$, a tedy $h_k(x) = h_k(y)$ implikuje existenci $q = -\lfloor \frac{N}{m} \rfloor, -\lfloor \frac{N}{m} \rfloor + 1, \ldots, -1, 1, 2, \ldots, \lfloor \frac{N}{m} \rfloor$ takového, že $kx - ky \equiv qm \bmod N$, a to je ekvivalentní s tím, že $k(x - y) \equiv qm \bmod N$ pro nějaké $q = -\lfloor \frac{N}{m} \rfloor, -\lfloor \frac{N}{m} \rfloor + 1, \ldots, -1, 1, 2, \ldots, \lfloor \frac{N}{m} \rfloor$.

Pro $x > y$ a pro jedno $q = 0, 1, \ldots, \lfloor \frac{N}{m} \rfloor$ existuje právě jedno k takové, že $k(x - y) \equiv qm \bmod N$, protože $\mathbb{Z}_N$ je těleso (tato rovnice má jediné řešení). Protože pro $q = -\lfloor \frac{N}{m} \rfloor, \ldots, -1, 0$

je rovnice $k(x-y) \equiv qm \bmod N$ ekvivalentní s rovnicí $k(x-y) \equiv N + qm \bmod N$, tak dostáváme, že pro $x, y \in S$, $x < y$ existuje nejvýše $2\lfloor \frac{N}{m} \rfloor = 2\lfloor \frac{N-1}{m} \rfloor$ různých $k = 1, 2, \ldots, N-1$, že $h_k(x) = h_k(y)$ (není pravda, že když k splňuje rovnici $k(x-y) \equiv qm \bmod N$ pro nějaké $q = -\lfloor \frac{N}{m} \rfloor, \ldots, -1, 1, \ldots, \lfloor \frac{N}{m} \rfloor$, pak $h_k(x) = h_k(y)$). Stejný odhad analogicky dostaneme, když $x < y$ (ale dostáváme jiná řešení). Odtud

$$\sum_{k=1}^{N-1} \left(\left(\sum_{i=0}^{m-1} \left(b_i^k\right)^2 \right) - n \right) \leq \sum_{x,y \in S, x \neq y} 2 \left(\frac{N-1}{m} \right) \leq 2(N-1) \frac{n(n-1)}{m}.$$

Pocet dvojic {x,y}, kde x =/= y

Tedy existuje k takové, že $\sum_{i=0}^{m-1} \left(b_i^k\right)^2 \leq 2\frac{n(n-1)}{m} + n$. (#3)

Ukážeme, že existuje více než $\frac{N-1}{4}$ takových k, že platí

$$\sum_{i=0}^{m-1} \left(b_i^k\right)^2 < 3\frac{n(n-1)}{m} + n.$$

V opačném případě dostáváme, že

$$\sum_{k=1}^{N-1} \left(\left(\sum_{i=0}^{m-1} \left(b_i^k\right)^2 \right) - n \right) \geq \frac{3(N-1)}{4} \frac{3n(n-1)}{m} = \frac{9(N-1)n(n-1)}{4m} > \frac{2(N-1)n(n-1)}{m},$$

a to je spor s předchozím výsledkem. Tedy při náhodném rovnoměrném výběru k je

$$\text{Prob}\left\{ \sum_{i=0}^{m-1} \left(b_i^k\right)^2 < \frac{3n(n-1)}{m} + n \mid k \in \{1, 2, \ldots, N-1\} \right\} \geq \frac{1}{4}.$$

Tvrzení. *Když $n = m$, pak*

(a) *existuje deterministický algoritmus, jenž v čase $O(nN)$ nalezne k takové, že*

$$\sum_{i=0}^{m-1} \left(b_i^k\right)^2 < 3n;$$

(b) *existuje pravděpodobnostní algoritmus, který nalezne takové k, že $\sum_{i=0}^{m-1} \left(b_i^k\right)^2 < 4n$ v čase $O(n)$ – očekávaný počet iterací výpočtu je nejvýše 4.*

Dále

(c) *existuje deterministický algoritmus, jenž v čase $O(nN)$ pro $m = n(n-1)+1$ nalezne takové k, že h_k je perfektní;*

(d) *existuje pravděpodobnostní algoritmus, který pro $m = 2n(n-1)$ v čase $O(n)$ nalezne k takové, že h_k je perfektní – očekávaný počet iterací výpočtu je nejvýše 4.*

a) *Důkaz.* Mějme $n = m$. Protože spočítání $\sum_{i=0}^{m-1} \left(b_i^k\right)^2$ pro pevné k vyžaduje čas $O(n)$, prohledáním všech možností nalezneme k takové, že

$$\sum_{i=0}^{m-1} \left(b_i^k\right)^2 \leq \frac{2n(n-1)}{n} + n = 3n - 2 < 3n,$$

Viz #3 z min. str., navic n = m.

v čase $O(nN)$. Tím je dokázáno a). Pravděpodobnostní algoritmus dokazující b) volí

b) náhodně k a v čase $O(n)$ ověří, zda $\sum_{i=0}^{m-1} \left(b_i^k\right)^2 \leq 3\frac{n(n-1)}{n} + n = 4n - 3 < 4n$. Tuto akci opakuje dokud požadavek není splněn. Protože pravděpodobnost, že k splňuje požadavek, je alespoň $\frac{1}{4}$, tak očekávaný počet iterací akce je nejvýše

$$\sum_{i=0}^{\infty} i \left(\frac{3}{4}\right)^{i-1} \frac{1}{4} = \frac{1}{4} \frac{1}{\left(1 - \frac{3}{4}\right)^2} = 4$$

a odtud plyne b).

c) Když $m = n(n-1) + 1$, pak prohledáním všech možností nalezneme k takové, že

$$\sum_{i=0}^{m-1} \left(b_i^k\right)^2 \leq \frac{2n(n-1)}{n(n-1)+1} + n < n + 2,$$

v čase $O(nN)$ a c) plyne z předchozí věty. Když $m = 2n(n-1)$, pak pro náhodně zvolené k platí s pravděpodobností $\leq \frac{1}{4}$, že

$$\sum_{i=0}^{m-1} \left(b_i^k\right)^2 \leq \frac{3n(n-1)}{2n(n-1)} + n < n + 2.$$

Algoritmus splňující tvrzení d) je stejný jako v případě b) (jen $m = 2n(n-1)$). □

Pozn: Takto zkonstruované perfektní hašovací funkce nesplňují požadavek 2) (platí $m = \Theta\left(n^2\right)$). Použijeme následující postup.

1) Nalezneme k takové, že pro $m = n$ platí $\sum_{i=0}^{m-1} \left(b_i^k\right)^2 < 3n$ (respektive $\sum_{i=0}^{m-1} \left(b_i^k\right)^2 < 4n$). Pro $i = 0, 1, \ldots, m-1$ nalezneme množiny $S_i = \{s \in S \mid h_k(s) = i\}$;

2) Pro každé $i = 0, 1 \ldots, m-1$ takové, že $S_i \neq \emptyset$, nalezneme pro $m = 1 + |S_i|(|S_i| - 1)$ (respektive $m = 1 + 2|S_i|(|S_i| - 1)$) takové k_i, že h_{k_i} je perfektní na S_i. Definujme $c_i = 1 + |S_i|(|S_i| - 1)$ (respektive $c_i = 2|S_i|(|S_i| - 1)$), když $S_i \neq \emptyset$, a $c_i = 0$, když $S_i = \emptyset$.

Def c_i:

3) Pro $i = 0, 1, \ldots, m$ definujme $d_i = \sum_{j=0}^{i-1} c_j$ a pro $x \in U$ označme $h_k(x) = l$. Pak položíme $g(x) = d_l + h_{k_l}(x)$.

Věta. *Zkonstruovaná funkce g je perfektní, hodnota $g(x)$ se pro každé $x \in U$ spočítá v čase $O(1)$, v deterministickém případě hašuje do tabulky velikosti $< 3n$ a je nalezena v čase $O(nN)$, v pravděpodobnostním případě hašuje do tabulky velikosti $< 6n$ a je nalezena v čase $O(n)$. Pro její zakódování jsou třeba hodnoty k a k_i pro $i = 0, 1, \ldots, m-1$. Tyto hodnoty jsou v rozmezí $1, 2, \ldots, N-1$, a tedy vyžadují $O(n \log N)$ paměti.*

Důkaz. Protože $g(S_i)$ pro $i = 0, 1, \ldots, m-1$ jsou navzájem disjunktní a h_{k_i} je perfektní na S_i, dostáváme, že g je perfektní. Pro výpočet hodnoty $g(x)$ jsou třeba dvě násobení, dvojí výpočet zbytku při dělení a jedno sčítání (hodnoty d_i jsou uloženy v paměti). Proto výpočet $g(x)$ vyžaduje čas $O(1)$. Dále d_m je horní odhad na počet řadků v tabulce. Protože pro $S_i \neq \emptyset$, máme $|S_i|(|S_i|-1)+1 \leq |S_i|^2 = \left(b_i^k\right)^2$, dostáváme v deterministickém případě $d_m = \sum_{i=0}^{m-1} c_i \leq \sum_{i=0}^{m-1} \left(b_i^k\right)^2 < 3n$ a k nalezneme v čase $O(nN)$. Protože k_i nalezneme v čase $O(|S_i|N)$, lze g zkonstruovat v čase $O\left(nN + \sum_{i=0}^{m-1} |S_i|N\right) = O\left(nN + N\sum_{i=0}^{m-1} |S_i|\right) = O(2nN) = O(nN)$. V pravděpodobnostním případě je

$$d_m = \sum_{i=0}^{m-1} c_i \leq \sum_{i=0}^{m-1} \left(2|S_i|^2 - 2|S_i|\right) = 2\sum_{i=0}^{m-1} \left(b_i^k\right)^2 - 2\sum_{i=0}^{m-1} b_i^k < 8n - 2n = 6n$$

(protože $|S_i| = b_i^k$ a $\sum_{i=0}^{m-1} b_i^k = n$). Protože k nalezneme v čase $O(n)$ a k_i v čase $O(|S_i|)$ dostaneme, že ~~že~~ g nalezneme v čase $O(n)$. Zbytek je jasný. □

Pozn: Tedy zkonstruovaná hašovací funkce splňuje požadavky 1), 2) a 3), ale požadavek 4) není splněn.

Mějme přirozené číslo m a nechť q je počet všech prvočísel dělících m ($p_1, p_2, \ldots$ je rostoucí posloupnost všech prvočísel). Pak

$$m \geq \prod_{i=1}^{q} p_i > q! = e^{\sum_{i=1}^{q} \ln i} \geq e^{\int_1^q \ln x dx} = e^{q\ln\left(\frac{q}{e}\right)+1} \geq \left(\frac{q}{e}\right)^q.$$

Proto existuje konstanta c, že $q \leq c\frac{\ln m}{\ln\ln m}$. Platí tedy

Věta. *Nechť $\delta(m)$ =počet prvočísel, která dělí m. Pak $\delta(m) = O\left(\frac{\log m}{\log\log m}\right)$.*

Mějme $S = \{s_1 < s_2 < \cdots < s_n\} \subseteq U$. Označme $d_{i,j} = s_j - s_i$ pro $1 \leq i < j \leq n$. Pak $s_i \bmod p \neq s_j \bmod p$, právě když $d_{i,j} \neq 0 \bmod p$. Označme $D = \prod_{1\leq i<j\leq n} d_{i,j} \leq N^{\binom{n}{2}}$. (#1)
Pak počet prvočíselných dělitelů čísla D je nejvýše $c\frac{\ln D}{\ln\ln D}$, a tedy mezi prvními $1 + c\frac{\ln D}{\ln\ln D}$ prvočísly existuje prvočíslo p takové, že $s_i \bmod p \neq s_j \bmod p$ pro každé $1 \leq i < j \leq n$. To
Def φ_p(x): znamená, že funkce $\phi_p(x) = x \bmod p$ je perfektní pro S. Podle věty o velikosti prvočísel $p_t \leq 2t\ln t$ pro každé $t \geq 6$, tedy

$$\begin{aligned} p &\leq 2\left(1 + c\frac{\ln D}{\ln\ln D}\right)\ln\left(1 + c\frac{\ln D}{\ln\ln D}\right) \leq \\ &4c\frac{\ln D}{\ln\ln D}\ln\left(2c\frac{\ln D}{\ln\ln D}\right) \leq \\ &4c(\ln 2c)\frac{\ln D}{\ln\ln D} + 4c\frac{\ln D}{\ln\ln D}\ln\left(\frac{\ln D}{\ln\ln D}\right) = \\ &\sout{4c\ln D + o(\ln D)} = O(\ln D) = O\left(n^2\ln N\right). \end{aligned}$$

Věta. *Pro každou n-prvkovou množinu $S \subseteq U$ existuje prvočíslo p o velikosti $O\left(n^2 \ln N\right)$ takové, že funkce $\phi_p(x) = x \bmod p$ je perfektní pro S.*

Složitost determinist. algoritmu:

Test, zda funkce $\phi_p(x) = x \bmod p$ je perfektní pro S, vyžaduje čas $O(n \log n)$. Tedy systematické hledání nejmenšího p, že ϕ_p je perfektní pro S, vyžaduje čas $O\left(n^3 \log n \log N\right)$. Nejmenší p takové, že ϕ_p je perfektní pro S, je prvočíslo. Navrhneme pravděpodobnostní algoritmus pro nalezení p. Pro dostatečně velké n mezi prvními $9c \ln D$ prvočísly je alespoň polovina takových prvočísel p, že ϕ_p je perfektní pro S. Algoritmus pak opakuje následující krok, dokud nenalezne perfektní funkci

Krok pstního algoritmu:

> vyberme náhodně číslo p mezi prvními $9cn^2 \ln N$ čísly a otestujme, zda p je prvočíslo a ϕ_p je perfektní

Odhad očekávaného počtu neúspěšných kroků.

Náhodně zvolené číslo $p \leq 9cn^2 \ln N$ je prvočíslo s pravděpodobností $\Theta\left(\frac{1}{\ln(9cn^2 \ln N)}\right)$ a pro prvočíslo p je ϕ_p perfektní s pravděpodobností $\geq \frac{1}{2}$. Tedy náhodně zvolené číslo $p \leq 9cn^2 \ln N$ splňuje test s pravděpodobností $\Theta\left(\frac{1}{\ln(9cn^2 \ln N)}\right)$, a proto očekávaný počet neúspěšných testů je $O\left(\ln\left(9cn^2 \ln N\right)\right)$. Tedy očekávaný čas algoritmu je

Ocekavan pocet neuspesnych testu -- ma byt log n^2??

$$O\left(n \log n \left(\log n + \log\log N\right)\right).$$

Otestovani vybraneho p

Věta. *Pro danou množinu $S \subseteq U$ takovou, že $|S| = n$, deterministický algoritmus nalezne prvočíslo $p = O\left(n^2 \log N\right)$ takové, že $\phi_p(x) = x \bmod p$ je perfektní pro S, a pracuje v čase $O\left(n^3 \log n \log N\right)$. Pravděpodobnostní algoritmus nalezne prvočíslo $p = O\left(n^2 \log N\right)$ takové, že ϕ_p je perfektní, v očekávaném čase $O\left(n \log n \left(\log n + \log\log N\right)\right)$.*

Výsledná p pro deter. a nedeter. algoritmus:

Deterministický algoritmus nalezne nejmenší prvočíslo s požadovanou vlastností. Pravděpodobnostní algoritmus nalezne prvočíslo, které může být podstatně větší, ale jeho velikost je omezena $9cn^2 \log N$.

Nyní navrhneme postup na konstrukci perfektní hašovací funkce pro množinu $S \subseteq U$.

(1) Nalezneme prvočíslo $q_0 \in O\left(n^2 \log N\right)$ takové, že $\phi_{q_0}(x) = x \bmod q_0$ je perfektní funkce pro S. Položme $S_1 = \{\phi_{q_0}(s) \mid s \in S\}$.
(2) Nalezneme prvočíslo q_1 takové, že $n(n-1) < q_1 \leq 2n(n-1)$. Pak existuje $l \in \{1, 2, \ldots, q_0 - 1\}$ takové, že $h_l(x) = ((lx) \bmod q_0) \bmod q_1$ je perfektní pro $S_1 \subseteq \{0, 1, \ldots, q_0 - 1\}$. Položme $S_2 = \{h_l(s) \mid s \in S_1\}$.
(3) Dále zkonstruujme perfektní hašovací funkci g pro množinu $S_2 \subseteq \{0, 1, \ldots, q_1 - 1\}$ do tabulky s méně než $3n$ řádky. Položme $f(x) = g(h_l(\phi_{q_0}(x)))$. Konstruovaná hašovací funkce je f.

Výsledek: f je perfektní hašovací funkce pro S, protože složení perfektních hašovacích funkcí je zase perfektní funkce, a tedy požadavek 1) je splněn.
f hašuje S do tabulky s méně než $3n$ řádky, a tedy splňuje požadavek 2).
Protože každá z funkcí g, h_l, ϕ_{q_0} se vyčíslí v čase $O(1)$, i vyčíslení funkce f vyžaduje čas $O(1)$ a požadavek 3) je splněn.
Funkce ϕ_{q_0} je jednoznačně určena číslem $q_0 \in O\left(n^2 \log N\right)$. Funkce h_l je určena čísly

$q_1 \in O\left(n^2\right)$ a $l \in O\left(q_0\right)$. Funkce g je určena $n+1$ čísly velikosti $O\left(q_1\right)$. Tedy zadání f vyžaduje paměť o velikosti

$$O\left(\log n + \log\log N + n\log n\right) = O\left(n\log n + \log\log N\right).$$

Lze říct, že požadavek 4) je splněn.

Výpočet ϕ_{q_0} vyžaduje čas $O\left(n^3 \log n \log N\right)$. Výpočet h_l vyžaduje čas $O\left(n\left(n^2 \log N\right)\right) = O\left(n^3 \log N\right)$ (použité univerzum je $\{0,1,\ldots,q_0\}$). Výpočet g vyžaduje čas $O\left(nn^2\right) = O\left(n^3\right)$ (zde univerzum je $\{0,1,\ldots,q_1\}$). Celkově, výpočet f vyžaduje čas $O\left(n^3 \log n \log N\right)$.

Lze použít i pravděpodobnostní algoritmy pro nalezení g, h_l a ϕ_{q_0}. Pak hašujeme do tabulky s méně než $6n$ řádky, ale očekávaný čas pro nalezení f je $O\left(n\log n\left(\log n + \log\log N\right)\right)$.

Tuto metodu navrhli Fredman, Komlós a Szemerédi.

Dynamické perfektní hašování.

Jedna z velkých nevýhod perfektního hašovaní je neznalost efektivních aktualizačních operací. Existují sice obecné metody na dynamizaci deterministických operací – viz letní přednáška, ale tato metoda v tomto případě neposkytuje efektivní dynamizační operace, protože deterministický algoritmus pro řešení perfektního hašování je pro aktualizační operace příliš pomalý. To vedlo k návrhu, který kombinuje pravděpodobnostní algoritmus pro perfektní hašování s obecnou metodou dynamizace a tyto metody jsou upraveny pro konkrétní situaci.

Nejprve uvedeme modifikaci výsledků z předchozí části, na kterých je tato metoda založena. Předpokládáme, že $U = \{0,1,\ldots,N-1\}$ je univerzum, kde N je prvočíslo, a že je dáno číslo $s < N$. Označme $\mathcal{H}_s = \{h_k \mid k = 1,2,\ldots,N-1\}$ množinu funkcí z U do $\{0,1,\ldots,s-1\}$, kde $h_k(x) = (kx \bmod N) \bmod s$ pro každé $x \in U$. Když zvolíme náhodně $k = 1,2,\ldots,N-1$, pak s pravděpodobností alespoň $\frac{1}{2}$ platí

$$\sum_{i=0}^{s-1} \left(b_i^k\right)^2 < \frac{8n^2}{s} + 2n.$$

Budeme předpokládat, že takové k máme, a pak pro každé $i = 0,1,\ldots,s-1$ předpokládáme, že náhodně zvolíme $j_i \in \mathcal{H}_{2\left(b_i^k\right)^2}$ takové, že h_{j_i} je prostá na množině $S_i = \{s \in S \mid h_k(s) = i\}$ (z předchozího textu víme, že když zvolíme náhodně $j_i = 0,1,\ldots,N-1$, pak h_{j_i} je prostá na S_i s pravděpodobností alespoň $\frac{1}{2}$). Pro jednoduchost předpokládáme, že množiny S_i pro $i = 0,1,\ldots,s-1$ uložíme do tabulek T_i a tabulky $T_0, T_1, \ldots, T_{s-1}$ budou uloženy v tabulce T. Když $s = O\left(|S|\right)$, pak tato metoda vyžaduje $O\left(|S|\right)$ prostoru. Abychom určili s, zvolme $c > 1$ a položme $s = \sigma\left(|S|\right)$, kde $\sigma(n) = \frac{4}{3}\sqrt{6}\,(1+c)\,n$ pro každé n. Nyní popíšeme algoritmy.

Algoritmy.

INSERT(x):

$n := n + 1$
if $n \leq m$ **then**
  $j := h(x)$, $|S_j| := |S_j| + 1$
  if $|S_j| \leq m(j)$ a pozice $h_j(x)$ v T_j je prázdná **then**
    vložíme x do tabulky T_j na pozici $h_j(x)$
  else
    if $|S_j| \leq m(j)$ a pozice $h_j(x)$ v T_j je obsazená **then**
      vytvoříme seznam S_j prvků v tabulce T_j
      vyprázdníme tabulku T_j
      zvolíme náhodně funkci $h_j \in \mathcal{H}_{2m(j)^2}$ | Volíme funkci h_j
      while h_j není prostá na množině S_j **do**
        zvolíme náhodně funkci $h_j \in \mathcal{H}_{2m(j)^2}$
      enddo
      for every $y \in S_j$ **do** vložíme y do T_j na pozici $h_j(y)$ **enddo** [Přehashujeme prvky]
    else [Platí |S_j| > m(j)]
      $m(j) := 2m(j)$ [Zdvojnásobíme prostor pro tabulku T_j]
      if není dost prostoru pro tabulku T_j nebo

$$\sum_{i=0}^{\sigma(m)-1} 2\left(m(i)\right)^2 \geq \frac{8m^2}{\sigma(m)} + 2m$$

      then
        RehashAll
      else
        alokujeme prostor pro novou prázdnou tabulku T_j
        vytvoříme seznam S_j prvků ze staré tabulky T_j a zrušíme ji
        zvolíme náhodně funkci $h_j \in \mathcal{H}_{2m(j)^2}$ | Volíme náhodně funkci h_j
        while h_j není prostá na množině S_j **do**
          zvolíme náhodně funkci $h_j \in \mathcal{H}_{2m(j)^2}$
        enddo
        for every $y \in S_j$ **do** vložíme y do T_j na pozici $h_j(y)$ **enddo** [Přehashujeme prvky]
      endif
    endif
  else
    RehashAll
  endif
~~**endif**~~

RehashAll:

projdeme tabulku T a tabulky T_i a vytvoříme seznam prvků z množiny S
$m := (1 + c)|S|$
zvolme náhodně $h \in \mathcal{H}_{\sigma(m)}$

for every $i = 0, 1, \ldots, \sigma(m) - 1$ **do** $S_i := \{x \in S \mid h(x) = i\}$ **enddo**
while $\sum_{i=0}^{\sigma(m)-1} 2\left(|S_i|\right)^2 < \frac{8m^2}{\sigma(m)} + 2m$ **do** (#1)
  zvolme náhodně $h \in \mathcal{H}_{\sigma(m)}$
  for every $i = 0, 1, \ldots, \sigma(m) - 1$ **do** $S_i := \{x \in S \mid h(x) = i\}$ **enddo**
enddo

Dokud neni splnena podminka #1, tak pomoci nahodne hash. funkce rozdeluju do mnozin S_i.

Komentář: zde S_i jsou množiny vytvořené náhodně zvolenou funkci h
$n := 0$
for every $i = 0, 1, \ldots, \sigma(m) - 1$ **do**
  $m(i) := |S_i|$
  zvolíme náhodně $h_i \in \mathcal{H}_{2m(i)^2}$
  while h_i není prostá na množině S_i**do**
    zvolíme náhodně $h_i \in \mathcal{H}_{2m(i)^2}$
  enddo
enddo
for every $x \in S$ **do INSERT**(x) **enddo**

Volíme funkci h_i

DELETE(x):
$j := h(x)$, $n := n - 1$, $|S_j| := |S_j| - 1$
odstraníme x z pozice $h_j(x)$ v tabulce T_j, pozice bude prázdná
if $n < \frac{m}{1+2c}$ **then RehashAll endif**

MEMBER(x):
$j := h(x)$
if x je na $h_j(x)$-té pozici v tabulce T_j **then**
  Výstup: x je prvek S
else
  Výstup: x není prvkem S
endif

Algoritmy pědpokládají, že při operaci **INSERT**(x) prvek x nepatří do S a při operaci **DELETE**(x) x je prvkem S. Pak n znamená velikost reprezentované množiny.

Uvedu složitost této metody bez důkazu.

Věta. *Popsaná metoda vyžaduje lineární paměť (neuvažuje se paměť potřebná pro zakódování hašovacích funkcí), operace* **MEMBER** *v nejhorším případě vyžaduje čas* $O(1)$ *a očekávaná amortizovaná složitost operací* **INSERT** *a* **DELETE** *je také* $O(1)$.

Toto zobecnění Fredman-Komlós-Szemerédiho metody navrhli Dietzfelbinger, Karlin, Mehlhorn, Meyer auf der Heide, Rohnert a Tarjan.

Další nevýhoda Fredman-Komlós-Szemerédiho metody:
Navržená metoda pracuje pro $m < 3n$, ale nezajistí $m = n$. Lze říct, že paměť je efektivní využitá? Existuje metoda, která by umožnila návrh perfektní hašovací funkce pro $m = n$? Z výsledků pro (N, m, n)-perfektní soubory funkcí plyne existence (N, n, n)-perfektního souboru pro $n^N > e^{n+ln(n)} \ln(N)$. Zmíníme se orientačně o parametrizované metodě, která

navrhuje perfektní hašovací funkci pro $S \subseteq U$ a pro $|S| = n$. Parametr bude přirozené číslo r, které určuje, jaké hypergrafy jsou užity při konstrukci funkce. Proto nejdříve připomeneme několik definic.

Def: Dvojice (X, E), kde X je množina a E je systém r-prvkových podmnožin X, se nazývá r-hypergraf. Prvky v E se nazývají hrany r-hypergrafu. Cyklus je hypergraf (X, E), kde každý vrchol leží alespoň ve dvou různých hranách. Naopak r-hypergraf (X, E) se nazývá acyklický, když žádný jeho podhypergraf není cyklus.

Nyní popíšeme metodu, která je rozdělena do dvou kroků. Je dáno $S \subseteq U$ takové, že $|S| = n$.

Krok 1) Mějme r-hypergraf (V, E), kde $|E| = n$. Nalezneme zobrazení

$$g : V \to \{0, 1, \ldots, n-1\}$$

takové, že funkce $h : E \to \{0, 1, \ldots, n-1\}$ definovaná $h(e) = \sum_{i=1}^{r} g(v_i) \bmod n$, kde $e = \{v_1, v_2, \ldots, v_r\}$, je prostá (místo sčítání modulo n můžeme použít libovolnou grupovou operaci na množině $\{0, 1, \ldots, n-1\}$). Pro acyklický r-hypergraf lze funkci g zkonstruovat následujícím postupem. Zvolíme bijekci $h : E \to \{0, 1, \ldots, n-1\}$ a pak definujeme g následovně: když $e = \{v_1, v_2, \ldots, v_r\}$ a $g(v_i)$ je definováno pro $i = 2, 3, \ldots, r$, pak

$$g(v_1) = h(e) - \sum_{i=2}^{r} g(v_i) \bmod n.$$

Protože pro každý acyklický r-hypergraf existuje vrchol, který leží v jediné hraně, lze tento postup použít ke konstrukci g pomocí indukce (a tedy máme algoritmus pro konstrukci g).

Krok 2) Nalezneme r funkcí $f_1, f_2, \ldots, f_r : U \to V$ takových, že (V, E), kde

$$E = \{\{f_1(x), f_2(x), \ldots, f_r(x)\} \mid x \in S\},$$

je acyklický r-hypergraf. Pak hašovací funkce f je definována $f(x) = \sum_{i=1}^{r} g(f_i(x))$ pro každé $x \in U$. Z konstrukce vyplývá, že je perfektní na množině S.

Autoři dokázali, že nejvhodnější alternativa je, když zobrazení $f_1, f_2, \ldots, f_r$ jsou náhodně zvolená náhodná zobrazení. Bohužel taková zobrazení neumíme zkonstruovat, ale autoři ukázali, že pro tyto účely lze použít náhodný výběr funkcí z nějakého c-univerzálního souboru funkcí.

Autoři ukázali, že jejich algoritmus vyžaduje $O(rn + |V|)$ času a $O(n \log n + r \log |V|)$ paměti.

Tento metapostup navrhli Majewski, Wormald, Havas a Czech (1996).

Pro praktické použití je problematická reprezentace r-hypergrafu a i náhodná volba funkcí $f_1, f_2, \ldots, f_r$ (viz předchozí diskuze o c-univerzalitě). Z požadavků na perfektní hašovací funkci je opět problémem splnění požadavku 4). Nevím, jak je uvedená metoda prakticky použitelná a zda se někde používá.

Externí hašování

Navržený postup je také znám pod názvem Faginův algoritmus. Tímto problémem se první asi zabýval Larsson.

Řešíme jiný problém – uložení dat na externí paměť. Hlavní problém – minimalizovat počet přístupů na externí paměť.

Předpoklady: Externí paměť je rozdělena na stránky, každá stránka obsahuje b položek (dat) (předpokládáme, že $b > 1$, jinak to nemá smysl). Vždy v jednom kroku načteme celou stránku do interní paměti nebo celou stránku v interní paměti v jednom kroku zapíšeme na externí medium. Tyto operace jsou řádově pomalejší než oprace v interní paměti.
Náš cíl: Nalézt způsob ukládání dat do stránek externí paměti, aby se minimalizoval počet operací s externí pamětí.

Předpokládejme, že $h : U \to \{0,1\}^*$ je prosté zobrazení takové, že délka $h(u)$ je stejná pro všechny prvky univerza U. Označme k délku $h(u)$ pro $u \in U$. Pak h je hašovací funkce.

Def: Nechť $S \subseteq U$, pak pro slovo α délky menší než k definujme

$$h_S^{-1}(\alpha) = \{s \in S \mid \alpha \text{ je prefix } h(s)\}.$$

Def: Řekneme, že α je kritické slovo, když $0 < |h_S^{-1}(\alpha)| \leq b$ a pro každý vlastní prefix α' slova α platí $|h_S^{-1}(\alpha')| > b$. Pro každé $s \in S$ existuje právě jedno kritické slovo α, které je prefixem $h(s)$. Definujme $d(s)$ pro $s \in S$ jako délku kritického slova, které je prefixem $h(s)$ a

$$d(S) = \max\{\text{délka}(\alpha) \mid \alpha \text{ je kritické slovo}\} = \max\{d(s) \mid s \in S\}.$$

Množinu S reprezentujeme tak, že je jednoznačná korespondence mezi kritickými slovy a stránkami externí paměti sloužícími k reprezentaci S. Na stránce příslušející kritickému slovu α je reprezentován soubor $h_S^{-1}(\alpha)$.
Problém: jak nalézt stránku kritického slova α?
Řešení: Adresář je funkce, která každému slovu α o délce $d(S)$ přiřadí adresu stránky předpisem

> když kritické slovo β je prefixem α, pak k α je přiřazena stránka korespondující s β, jinak je k α přiřazena stránka NIL – speciální prázdná stránka.

Korektnost: Pro různá kritická slova β a γ platí $h_S^{-1}(\beta) \cap h_S^{-1}(\gamma) = \emptyset$, a tedy pro každé slovo α délky $d(S)$ existuje nejvýše jedno kritické slovo, které je prefixem α. Když α je slovo délky $d(S)$, pak nastane jeden z těchto tří případů:

(1) $h_S^{-1}(\alpha) \neq \emptyset$, pak $0 < |h_S^{-1}(\alpha)| \leq b$ a existuje právě jedno kritické slovo β, které je prefixem α;
(2) $h_S^{-1}(\alpha) = \emptyset$ a existuje prefix α' slova α takový, že $0 < |h_S^{-1}(\alpha')| \leq b$, pak existuje právě jedno kritické slovo, které je prefixem α' (a tedy také prefixem α);
(3) $h_S^{-1}(\alpha) = \emptyset$ a pro každý prefix α' slova α platí buď $h_S^{-1}(\alpha') = \emptyset$ nebo $|h_S^{-1}(\alpha')| > b$ (pak k α je přiřazena stránka NIL.

Mějme slovo α o délce $d(S)$. Označme $c(\alpha)$ nejkratší prefix α' slova α takový, že stránka přiřazená slovu β o délce $d(S)$, které má α' za prefix, je stejná jako stránka přiřazená α.

Všimněme si, že když $h_S^{-1}(\alpha) \neq \emptyset$, pak $c(\alpha)$ je kritické slovo. Platí silnější tvrzení, které tvrdí, že následující podmínky jsou ekvivalentní:

(1) stránka přiřazená slovu α je různá od NIL;
(2) $c(\alpha)$ je kritické slovo;
(3) nějaký prefix α je kritické slovo.

Všimněme si, že znalost adresáře umožňuje nalézt slovo $c(\alpha)$ pro každé slovo o délce $d(S)$.

Lineární uspořádání na slovech délky n nazveme lexikografické, když $\alpha < \beta$, právě když $\alpha = \gamma 0 \alpha'$ a $\beta = \gamma 1 \beta'$ pro nějaká slova γ, α' a β'. Lexikografické uspořádání vždy existuje a je jednoznačné.

Reprezentace adresáře: Je to seznam adres stránek o délce $2^{d(S)}$ takový, že adresa na i-tém místě odpovídá i-tému slovu délky $d(S)$ v lexikografickém uspořádání.
Příklad: U je množina všech slov nad $\{0,1\}$ o délce 5, h je identická funkce a $b = 2$. Reprezentujme množinu $S = \{00000, 00010, 01000, 10000\}$. Pak $d(00000) = d(00010) = d(01000) = 2$, $d(10000) = 1$, kritická slova jsou 00, 01 a 1 a adresář je (místo adresy stránky uvedeme množinu, která je na této stránce uložena)

$$00 \mapsto \{00000, 00010\}, \quad 01 \mapsto \{01000\}, \quad 10 \mapsto 11 \mapsto \{10000\}.$$

Tedy $c(00) = 00$, $c(01) = 01$ a $c(10) = c(11) = 1$. Když odstraníme prvek 10000, pak 1 přestane být kritické slovo a adresář bude mít tvar

$$00 \mapsto \{00000, 00010\}, \quad 01 \mapsto \{01000\}, \quad 10 \mapsto 11 \mapsto NIL.$$

Opět platí $c(00) = 00$, $c(01) = 01$ a $c(10) = c(11) = 1$. V adresáři je také uloženo $d(S)$.

Algoritmy.

Slovní popis operací. Předpokládáme, že adresář je uložen v externí paměti na jedné stránce.

MEMBER(x):
1) Spočítáme $h(x)$ a **načteme** adresář do interní paměti. Vezmeme prefix α slova $h(x)$ o délce $d(S)$ a nalezneme adresu stránky příslušející k α. Když je to stránka NIL, pak $x \notin S$ a konec, jinak pokračujeme krokem 2).
2) **Načteme** stránku příslušející k α do interní paměti. Prohledáme ji a pokud neobsahuje x, pak $x \notin S$ a konec. Když obsahuje x, pak provedeme požadované změny a stránku **uložíme** do externí paměti na její původní místo. Konec.

INSERT(x):
1) Spočítáme $h(x)$ a **načteme** adresář do interní paměti. Vezmeme prefix α slova $h(x)$ o délce $d(S)$ a nalezneme adresu stránky příslušející k α a slovo $c(\alpha)$. Když stránka přiřazená k α je NIL, pokračujeme krokem 3), v opačném případě pokračujeme krokem 2).
2) **Načteme** stránku přiřazenou slovu α. Když x je uloženo na této stránce, pak skončíme. Když x není na této stránce, pak tam přidáme slovo x. Pokud na stránce je nejvýše b prvků, pak **uložíme** stránku na její původní místo a skončíme. Když na stránce je více než b prvků, pak nalezneme nová kritická slova, která nám stránku rozdělí, a vytvoříme dvě stránky – jednu **uložíme** na místo původní stránky a druhou **uložíme** na novou stránku.

Pokračujeme krokem 4).
3) Vytvoříme v interní paměti novou stránku, která obsahuje x, nalezneme novou stránku v externí paměti a tam **uložíme** vytvořenou stránku (všem slovům, která mají $c(\alpha)$ za prefix, bude přiřazena tato stránka) a pokračujeme krokem 4).
4) **Načteme** opět adresář do interní paměti, aktualizujeme adresy přiřazených stránek a případně zvětšíme adresář (to nastane, když nějaké nové kritické slovo má délku větší než $d(S)$, pak nové $d(S)$ je právě délka tohoto slova – obě kritická slova vzniklá v kroku 2) mají stejnou délku). Aktualizovaný adresář **uložíme** do externí paměti. Konec.

DELETE(x):
1) Spočítáme $h(x)$ a **načteme** adresář do interní paměti. Vezmeme prefix α slova $h(x)$ o délce $d(S)$ a nalezneme adresu stránky příslušející k α a slovo $c(\alpha)$. Když stránka přiřazená k α je NIL, pak skončíme. Označme β' slovo, které má stejnou délku jako $c(\alpha)$ a liší se od $c(\alpha)$ pouze v posledním bitu. Když existuje slovo β délky $d(S)$ takové, že $c(\beta) = \beta'$, pak stránka přiřazená k β je kandidát.
2) **Načteme** stránku příslušnou k slovu α do interní paměti. Když tato stránka neobsahuje x, pak skončíme. Když tato stránka obsahuje x, pak odstraníme x z této stránky. Když neexistuje kandidát nebo když nová stránka a stránka kandidáta dohromady obsahují více než b prvků, pak novou stránku **uložíme** na její původní místo a skončíme.
3) Když nová stránka a stránka kandidáta mají dohromady b prvků, pak **načteme** stránku kandidáta do interní paměti. V interní paměti tyto stránky spojíme do jedné a tuto stránku pak **uložíme** do externí paměti.
4) **Načteme** adresář, kde zaktualizujeme adresy stránek. Pokud jsme sloučili dvě stránky, musíme nalézt nové $c(\alpha)$ (je to nejkratší prefix α' slova α takový, že ke každému slovu β o délce $d(S)$, které má α' za prefix, je přiřazena jedna z těchto adres: adresa stránky přiřazená k α, adresa stránky kandidáta, NIL) a každému slovu o délce $d(S)$, které má nové $c(\alpha)$ za prefix, bude přiřazena adresa nové (spojené) stránky. Otestujeme, zda se adresář nemůže zkrátit (to nastane, když adresy stránek přiřazené $(2i+1)$-ímu slovu a $(2i+2)$-ému slovu o délce $d(S)$ jsou stejné pro všechna i, pak se tato slova spojí a $d(S)$ se zmenší o 1). Upravený adresář **uložíme**. Konec.

Následující věta ukazuje, že jsme náš hlavní cíl splnili. Pro jednoduchost předpokládáme, že adresář je také uložen na externí paměti a že v interní paměti nemůže být uložen spolu s nějakou jinou stránkou.

Věta. *Operace* **MEMBER** *vyžaduje nejvýše tři operace s externí pamětí. Operace* **INSERT** *a* **DELETE** *vyžadují nejvýše šest operací s externí pamětí.*

V našem příkladu provedeme operaci **INSERT**(00001). Po přidání prvku stránka původně přiřazená k slovu 00 vypadá takto $\{00000, 00001, 00010\}$. Tuto stránku rozdělíme na stránky $\{00000, 00001\}$ a $\{00010\}$. Přitom kritické slovo první stránky je 0000 a druhé stránky

je 0001. Takže $d(S) = 4$ a adresář vypadá

$$\begin{aligned}
&0000 \mapsto \{00000, 00001\}, 0001 \mapsto \{00010\},\\
&0010 \mapsto 0011 \mapsto NIL,\\
&0100 \mapsto 0101 \mapsto 0110 \mapsto 0111 \mapsto \{0100\},\\
&1000 \mapsto 1001 \mapsto 1010 \mapsto 1011 \mapsto \{10000\},\\
&1100 \mapsto 1101 \mapsto 1110 \mapsto 1111 \mapsto \{10000\}.
\end{aligned}$$

To znamená, že kromě adresy 00 se ostatní slova rozdělila na čtyři slova, ale adresy zůstaly stejné. Jen u slova 00 vzniklá slova dostala různé adresy.

V původním příkladu provedeme operaci **DELETE**(01000). Pak kandidát je 00 a po odstranění prvku 01000 nastane spojení těchto dvou stránek. Po aktualizaci adres dostane adresář tvar

$$00 \mapsto 01 \mapsto \{00000, 00010\}, 10 \mapsto 11 \mapsto \{10000\},$$

tj. k prvnímu a druhému slovu je přiřazena stejná stránka a stejně tak k třetímu a čtvrtému slovu. Takže můžeme adresář zmenšit. Pak $d(S) = 1$ a adresář má podobu

$$0 \mapsto \{00000, 00010\}, 1 \mapsto \{10000\}.$$

Vzniká otázka, jak je tato metoda efektivní. Hlavně jak efektivně využívá paměť. Platí

Věta. *Když velikost reprezentované množiny je n, pak očekávaný počet použitých stránek je $\frac{n}{b \ln 2}$ a očekávaná velikost adresáře je $\frac{e}{b \ln 2} n^{1+\frac{1}{b}}$.*

První tvrzení říká, že očekávaný počet prvků na stránce je $b \ln 2 \approx 0.69b$. Tedy zaplněno je asi 69% míst. Tento výsledek není překvapující a je akceptovatelný. Horší je to s adresářem, jak ukazuje následující tabulka

velikost S	10^5	10^6	10^8	10^{10}
2	$6.2 \cdot 10^7$	$1.96 \cdot 10^8$	$1.96 \cdot 10^{11}$	$1.96 \cdot 10^{14}$
10	$1.2 \cdot 10^5$	$1.5 \cdot 10^6$	$2.4 \cdot 10^8$	$3.9 \cdot 10^{10}$
50	$9.8 \cdot 10^3$	$1.0 \cdot 10^6$	$1.1 \cdot 10^8$	$1.2 \cdot 10^{10}$
100	$4.4 \cdot 10^3$	$4.5 \cdot 10^4$	$4.7 \cdot 10^6$	$4.9 \cdot 10^8$

kde jednotlivé řádky odpovídají hodnotám b uvedených v prvním sloupci. Protože očekávaná velikost adresáře se zvětšuje rychleji než lineárně (exponent u n je $1+\frac{1}{b}$), tak nelze očekávat, že tuto metodu lze vždy použít. Výpočty i experimenty ukazují, že použitelná je do velikosti $|S| = 10^{10}$, když $b \approx 100$. V tomto rozmezí je nárůst adresáře jen kolem 5%. Pro větší n je třeba, aby b bylo ještě větší.